RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ AMAR TELIDJI-LAGHOUAT

FACULTÉ DES SCIENCES
DÉPARTEMENT DES MATHÉMATIQUES ET D'INFORMATIQUE

T H È S E

pour obtenir le titre de

Docteur en Sciences

Spécialité : INFORMATIQUE

Présentée par

Slimane BELLAOUAR

Noyaux de mots et d'arbres : efficacité et unification

Soutenue publiquement le 15/03/2018 devant le jury composé de :

Président :	M.B. YAGOUBI	Prof.	UAT	Laghouat
Examinateurs :	D.E. ZEGOUR	Prof.	ESI	Alger
	A. MOUSSAOUI	Prof.	UFA	Sétif 1
	Y. OUINTEN	Prof.	UAT	Laghouat
Directeur :	D. ZIADI	Prof.	Normandie Université	France
Co-Directeur :	H. CHERROUN	Prof.	UAT	Laghouat

لا يزال المرء عالما ما دام في طلب العلم، فإذا ظن أنه قد علم فقد بدأ جهله.

ابن قتيبة(٩٨٨ م)، أديب فقيه محدث مؤرخ عربي

ملخص

إن التعلم الآلي يستدعي طرقا ذكية لتحليل البيانات من أجل الإستخراج الآلي للمعلومات الهامة من مجموعات البيانات الضخمة. لكن الطرق التقليدية للتعلم الآلي هي عبارة عن طرق خطية غالبا ما تكون مهيأة لمستندات مسطحة. إلا أنه عمليا، نجد أن العديد من التطبيقات لديها بيانات يمكن تمثيلها طبيعيا على شكل مركب (سلاسل، أشجار، مخططات . . .). إن طرق النواة تشكل مقاربات فعالة تستند الى أسس نظرية صلبة لدعم هذا النوع من البيانات. ان هاته الطرق استعملت على نطاق واسع من أجل التعلم الآلي من خلال البيانات المركبة.

إن هدفنا في هاته الرسالة ذو شقين. بادئ بدء، ركزنا على جانب الفعالية الذي يعتبر خاصية أساسية لطرق النواة. من هذا المنظور قمنا باستهداف نواة السلاسل الجزئية للكلمات (string subsequence kernel, SSK) المستعمل بنجاح في عديد من مهام تعلم الآلة. إن الفكرة الأساسية لمقاربتنا تتمثل في تحويل حساب نواة السلاسل الجزئية للكلمات الى مشكل هندسي. على وجه التحديد، قد استخدمنا شجرة مجالات ذات طبقات (layered range tree, LRT) أين طبقنا خوارزميات الهندسة الحسابية المتعلقة بها.

وبهدف تحسين مقاربتنا، قمنا بتوسيع بنية المعطيات، شجرة مجالات ذات طبقات (LRT) الى شجرة مجالات جمع ذات طبقات (layered range sum tree, LRST) مزودة بعمليات التجميع. وبالاضافة الى ذلك، قدمنا تقييمات تجريبية للمقاربة الموسعة سواء على البيانات التركيبية أو بيانات حقيقية مستمدة من المواد الاخبارية. وقد أظهرت النتائج فعالية المقاربة المقترحة بالنسبة للأبجديات الكبيرة باستثناء الكلمات القصيرة جدا.

الهدف الثاني للرسالة يتمثل في المساهمة في تطوير نظرية موحدة للتعلم الآلي. في السنوات الأخيرة كرس جهد كبير لنوى السلاسل حيث تم التركيز على المسائل الفردية، مما أدى الى مجموعة متنوعة من المقاربات. في هذا السياق، اقترحنا اطارا عاما للتعامل مع تقييم نوى السلاسل. في الواقع، إنه بمكن تمثيل كلمة s في فضاء الخصائص ذو الأبعاد الكبيرة بواسطة السلاسل الصورية التي يمكن أن تنجز بواسطة الآلي المرجح A_s الذي يمثل كل السلاسل الجزئية للكلمة s. وعليه فإن حساب نواة كل السلاسل الجزئية $K(s,t)$ بين كلمتين s و t هو سلوك الآلي المرجح $A_{s,t} = A_s \cap A_t$.

من أجل تقييم فعال لنواة كل السلاسل الجزئية اقترحنا تقنية جديدة لتقاطع الآليات تعتمد على البحث المسبق الى الأمام. كشفت نتائج التجارب أن تقييم نواة كل السلاسل الجزئية باستعمال تقنيتنا المقترحة هو أسرع منه باستخدام التقاطع القياسي. علاوة على ذلك، قد تمكنا من تعميم نموذج الآليات المرجحة من أجل انشاء نواة جديدة لمعالجة مجموعة من السلاسل التي يمكن اعتبارهاك كذلك نواة شجرية.

كلمات مفتاحية : التعلم الآلي، نوى منطقية، نوى الكلمات، نوى الأشجار، نوى السلاسل الجزئية للكلمات، فعالية، توحيد، شجرة مجالات جمع ذات طبقات، آليات مرجحة، نواة مجموعة من السلاسل.

Résumé

L'apprentissage automatique fait appel à des méthodes intelligentes d'analyse de données qui consistent à extraire automatiquement de l'information significative à partir des collections de données massives. Cependant, les méthodes classiques de l'apprentissage automatique sont des méthodes linéaires. Elles sont souvent très bien adaptées à des documents plats. Dans la pratique, de nombreuses applications disposent de données qui peuvent être représentées naturellement sous une forme structurée (séquences, arbres, graphes, ...). Les méthodes à noyaux constituent des approches efficaces disposant d'un fondement théorique solide pour prendre en charge ce type de données. Elles ont été largement utilisées pour l'apprentissage automatique à partir des données structurées.

L'objectif de notre travail est double. En premier lieu, nous nous sommes focalisés sur l'aspect efficacité, qui est une propriété clé des méthodes à noyaux. Dans cette perspective nous avons ciblé le noyau sous-séquences de mots (string subsequence kernel, SSK), qui est utilisé avec succès dans plusieurs tâches de l'apprentissage automatique. L'idée de base de notre approche consiste à réduire le calcul du noyau SSK à un problème géométrique. Plus précisément, nous avons fait appel à un arbre d'intervalles en couches (layered range tree, LRT) pour lequel nous avons appliqué les algorithmes de géométrie calculatoire correspondants.

Dans une perspective d'améliorer notre approche, nous avons étendu la structure de données arbre d'intervalles en couches (LRT) à un arbre d'intervalles de somme en couches (layered Range Sum Tree, LRST) doté des opérations d'agrégation. De même, nous avons présenté des évaluations empiriques de l'approche étendue, à la fois sur des données synthétiques et des données réelles extraites des articles de presse. les résultats ont montré l'efficacité de notre approche pour des alphabets de grande taille, à l'exception des mots trop courts.

Le second objectif de la thèse consiste à contribuer au développement d'une théorie d'unification des méthodes liées à l'apprentissage automatique. En fait, dans les dernières années, un effort important a été consacré aux noyaux de séquences en se concentrant sur des problèmes spécifiques conduisant, ainsi, à une variété d'approches. Dans ce contexte, nous avons proposé une plate-forme générale qui s'occupe de l'évaluation des noyaux de séquences. En effet,

la projection d'un mot s dans un espace de redescription de haute dimension peut être modélisée par une série formelle réalisée par un automate pondéré (weighted automaton, WA) A_s représentant toutes les sous-séquences de s. Le calcul du noyau toutes sous-séquences $K(s,t)$ entre deux mots s et t est le comportement de l'automate pondéré $A_{s,t} = A_s \cap A_t$.

Pour une évaluation efficace d'un tel noyau, nous avons proposé une nouvelle technique d'intersection d'automates (intersection par anticipation). les résultats des expérimentations ont révélé que l'évaluation du noyau toutes sous-séquences utilisant notre technique est plus rapide que celle utilisant l'intersection standard. De plus, nous avons pu généraliser notre modèle à base d'automates pondérés pour créer un nouveau *noyau d'ensembles de séquences* qui peut être vu comme un noyau d'arbre.

Mots clés : Apprentissage automatique, noyau rationnel, noyau de mots, noyau d'arbres, noyau sous-séquences de mots, efficacité, unification, arbre d'intervalles de somme en couches, automate pondéré, noyau d' ensembles de séquences.

Abstract

Machine learning requires data analysis intelligent methods for automatic extraction of meaningful information from massive data collection. However, the traditional methods of machine learning are linear. They are often well adapted to flat documents. In practice, many applications have data that can be represented naturally in a structured way (sequences, trees, graphs, ...). Kernel methods are effective approaches with a solid theoretical foundation to support this type of data. They have been widely used for learning from structured data.

The goal of our work is twofold. First, we focused on the efficiency aspect, which is a key property of kernel methods. In this perspective we have targeted the string subsequence kernel (SSK), which is used successfully in several tasks of machine learning. The basic idea of our approach is to reduce the SSK computation problem to a geometric one. More precisely, we have used a layered range tree (LRT) and applied the corresponding computational geometry algorithms.

In order to improve our approach, we extended the LRT data structure to a layered Range Sum Tree (LRST), a range-aggregation data structure. Moreover, we presented empirical evaluations of the extended approach, on both synthetic data and newswire article data. The results showed the efficiency of the proposed approach for large alphabets except for very short strings.

The second objective of the thesis is to contribute in developing a unified theory of machine learning. In recent years, a significant effort has been devoted to sequence kernels focusing on individual problems, and so leading to a variety of approaches. In this context, we have proposed a general framework to deal with sequence kernel evaluation. In fact, the mapping of a string s to a high dimensional feature space can be modeled by a formal power series that can be realized by a weighted automaton (WA) A_s representing all subsequences of s. The computation of the all subsequence kernel $K(s,t)$ between two strings s and t is the behavior of the WA $A_{s,t} = A_s \cap A_t$.

For an efficient kernel evaluation, we proposed a forward lookup automata intersection technique. The results of the experiments revealed that the evaluation of the all-subsequence kernel using our proposed technique is faster than that using the standard intersection. Moreover, we are able to generalize our weighted automaton model to create a new *sequence-set kernel* that can be seen as a tree kernel.

Keywords : Machine learning, rational kernel, string kernel, tree kernel, string subsequence kernel, efficiency, unification, layered range sum tree, weighted automaton, sequence-set kernel.

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce modeste travail.

Une thèse de doctorat est loin d'être un travail individuel. Son élaboration nécessite le concours de plusieurs personnes. Je peux difficilement imaginer l'achèvement de plusieurs années de travail sans les remercier.

Ma dette de reconnaissance va en premier lieu à mes directeurs de thèse, Monsieur Djelloul Ziadi, professeur à Normandie université (France) et Madame Hadda Cherroun, professeur à l'université de Laghouat, pour leurs intérêts, disponibilités, motivations, soutiens, conseils, confiance ainsi que pour les valeurs scientifiques transmises. Je les remercie chaleureusement.

J'adresse aussi mes remerciements à Monsieur Mohamed Bachir Yagoubi d'avoir accepté de présider le jury et pour l'intérêt qu'il porte à mon travail.

Mes vifs remerciements vont à Monsieur Djamel-Eddine Zegour, professeur à l'école supérieure d'informatique à Alger, Monsieur Abdelouahab Moussaoui, professeur à l'université de Sétif 1 et Monsieur Youcef Ouinten, professeur à l'université de Laghouat, d'avoir accepté de lire et de discuter mon travail. Ils m'ont honoré d'être membres de jury de ma thèse.

Je tien aussi à remercier Madame Habiba Drias, professeur à l'Université des Sciences et de la Technologie Houari Boumediene à Alger, de l'intérêt qu'elle a bien voulu porter à mon travail.

Mes remerciements vont également à Monsieur Nacereddine Lagraa, directeur du Laboratoire d'Informatique et de Mathématiques de l'université de Laghouat pour le bon accueil au sein du laboratoire ainsi que pour ses encouragements et soutiens.

Mes sincères remerciements vont à mes collègues du Laboratoire d'Informatique et de Mathématiques de l'université de Laghouat avec qui j'ai partagé de bon moments. Je cite, particulièrement, Slimane Oulad-Naoui, Attia Nehar, Youness Guellouma, Abdellah Lakhdari, Nicha Chorana, Ahlem Belabbassi et Soumia Bougrine.

Ma gratitude s'adresse également à tous les membre du Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes de Haute-Normandie, sous la direction du professeur Jean-Gabriel Luque, pour l'accueil chaleureux.

À Monsieur Bruce W. Watson, professeur à l'université de Stellenbosch en Afrique du sud. Merci pour l'accueil, l'écoute et l'encouragement.

Un grand merci, très particulier, pour mon Collègue Laradj Chellama, enseignant à l'université de Laghouat pour tous ce qu'il a fait, notamment de m'avoir aidé dans mes démêlés administratifs.

Je remercie aussi très vivement Monsieur Slimane Abdelhakem, Dr. à l'université de Ghardaia, de m'avoir accordé de son précieux temps pour la relecture du présent manuscrit.

Je suis très reconnaissant à Monsieur Karim Souffi, ex. cadre de la wilaya de Ghardaia, qui a passé beaucoup de nuits à lire et de relire ma thèse, malgré tous les empêchements. Sa disponibilité, aide et conseils sont inestimables.

Je tiens également à remercier Mademoiselle Habiba Benabderrahmane, ingénieur de labo. à l'université de Ghardaia, d'avoir pris en charge une bonne partie des figures en Latex.

Je remercie aussi tous mes collègues enseignants des universités de Ghardaia, Laghouat, Ouargla et Djelfa pour m'avoir soutenu et encouragé tout au long de la rédaction de cette thèse.

Je remercie, de tout mon coeur, tous mes proches pour la confiance, le soutient, la patience et l'aide qu'ils m'ont apporté. La liste est non exhaustive, néanmoins, je cite, particulièrement, mes frères Messaoud et Belkheir, mes soeurs Fatima, Fadila et Rebha, mes neveux et mes nièces. Je n'oublie pas ma belle mère, mes belles soeurs et mes beaux frères.

À tous ceux dont je n'ai pas pu citer les noms, mais qui m'ont, d'une manière ou d'une autre, de pré ou de loin, apporté leur appui : qu'ils sachent que je leur suis très reconnaissant.

Au terme de ce parcours, je remercie enfin ceux et celles qui me sont chers et que j'ai quelque peu délaissé durant ce long processus de préparation de cette thèse. Je suis redevable à ma très chère mère de m'avoir inculqué de mener les choses à terme, de son attention et de son encouragement constant. Aussi, mes profonds remerciements vont à ma très chère femme, son soutient moral inconditionnel au moments difficiles, sa patience et sa confiance indéfectible m'ont accompagné tout au long de ce travail de longue haleine ! Mes remerciements vont à mes enfants, Mouncif, Anfel, Hatem et Moundhir, qui ont été privés de ma présence de leur coté et pour leur sens de durs sacrifices. Qu'ils sachent tous, qu'en dépit de toutes les méthodes à noyaux, qu'ils sont la meilleure partie dans ma vie. Enfin, j'ai une pensée toute particulière pour mon père et mon frère Elhadj.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations, des sigles et des acronymes

ABRB	**A**rbre **B**inaire de **R**echerche **B**alancé
ACC	**A**nalyse **C**anonique des **C**orrélations
ACP	**A**nalyse en **C**omposantes **P**rincipales
APP	**A**utomate **P**ondéré **P**réfixe
AR	**A**pprentissage par **R**enforcement
ATC	**A**rabic **T**ext **C**lassification
BioMinT	**B**iological **T**ext **M**ining
GASWA	**G**appy **A**ll **S**ubsequence **W**eighted **A**utomaton
GVSM	**G**eneralized **V**ector **S**pace **M**odel
IDC	**I**nternational **D**ata **C**orporation
IE	Information **E**xtraction
idf	**i**nverse **d**ocument **f**requency
iid	**i**ndépendants et **i**dentiquement **d**istribués
KRISP	**K**ernel-based **R**obust **I**nterpretation for **S**emantic **P**arsers
kNN	**k** **N**earest **N**eighbors
LRT	**L**ayered **R**ange **T**ree
LRST	**L**ayered **R**ange **S**um **T**ree

LSI	**L**atent **S**emantic **I**ndexing
MEDLINE	**Med**ical Literature, Analysis, and Retrieval System On**line**
OCR	**O**ptical **C**haracter **R**ecognition
PAC	**P**robablement **A**pproximativement **C**orrect
PT	**P**artial **T**ree kernel
PTK	**P**arse **T**ree **K**ernel
RT	**R**ange **T**ree
SCOP	**S**tructural **C**lassification **O**f **P**roteins
SPA	**S**audi **P**ress **A**gency
SRL	**S**emantic **R**ole **L**abeling
SSK	**S**tring **S**ubsequence **K**ernel
SSK-LP	**S**tring **S**ubsequence **K**ernel **L**ambda **P**runing
SSTK	**S**emantic **S**yntactic **T**ree **K**ernel
STK	**S**ub**T**ree **K**ernel
SVD	**S**ingular **V**alue **D**ecomposition
SVM	**S**éparateurs à **V**aste **M**arge
tf	**t**erm **f**requency
VSM	**V**ector **S**pace **M**odel
WA	**W**eighted **A**utomaton

Chapitre 1

Introduction

1.1 Contexte

« Nous nous noyons dans l'information et nous nous affamons aux connaissances » (Naisbitt 1982). Cette citation reflète fidèlement les effets et les défis imposés par notre ère de déluge informationnel. Par « effet », nous entendons la surabondance d'information résultant d'une digitalisation grandissante, où petit à petit, tout devient information dans une société dite de l'information. La question qui se pose est de savoir si cette société est capable de digérer cette surabondance d'information ? Ici, « digérer » est utilisé dans le sens de nourrir une décision ou enrichir un capital de connaissance. Si la réponse est négative, ceci risque de polluer le climat scientifique et intellectuel et de tendre, par conséquence, à une nouvelle pandémie communément appelée *Infopollution* ou *Infobésité* [1].

Pour des raisons écologiques « informationnelles », la société doit relever le défi et de faire face à cette infopollution, dans un environnement où les modèles classiques d'analyse de données sont adaptés à des volumes d'information restreints. En effet, selon un rapport de l'IDC [2] (International Data Corporation), seulement 0.5% des 643 exaoctets des données estimées utiles, pour l'année 2012, ont été analysées. Les remèdes sont multiples, dans notre contexte, une solution consiste à faire appel à des méthodes intelligentes d'analyse de données

1. http://eduscol.education.fr/numerique/edunum-thematique/edunum_01
2. IDC Digital Universe Study, commandité par EMC, décembre 2012.

qui consistent à extraire de l'information significative à partir des collections de données massives. C'est la motivation principale de l'apprentissage automatique.

L'apprentissage automatique vise à fournir des outils automatiques pour imiter la capacité humaine à améliorer son comportement avec l'expérience. C'est un domaine en plein essor, il est utilisé pour un large spectre d'applications : traitement du langage naturel, bio-informatique, diagnostic médical, reconnaissance de formes, moteurs de recherche, détection de fraudes, analyse des marchés boursiers, génie-logiciel, web adaptatif, robotique, jeux, ...

Cependant, les méthodes classiques d'apprentissage automatique sont des méthodes linéaires. Elles sont souvent très bien adaptées à des documents plats représentés par des modèles vectoriels. Dans la pratique, de nombreuses applications disposent de données qui peuvent être représentées naturellement sous une forme structurée. À titre d'exemple, les documents XML sont naturellement représentés par des arbres ; dans le traitement de la langue naturelle, chaque phrase peut être représentée par un arbre syntaxique. Dans la bioinformatique, les protéines peuvent être représentés comme des séquences d'acides aminés et l'ADN génomique comme une séquence de nucléotides. Ce problème de représentation structurée des données peut être abordé en changeant la représentation des données par le biais de fonctions non linéaires tout en gardant les régularités et les dépendances inhérentes aux données. Les méthodes à noyaux rendent possible cette astuce en projetant les données dans un espace de redescription de haute dimension tout en évitant le calcul explicite de cette projection.

Par définition, les méthodes à noyaux cherchent une relation linéaire dans l'espace de redescription. Ainsi, les données en entrées peuvent être comparées par le biais des produits scalaires de leurs représentations dans l'espace de redescription. Cependant, les méthodes à noyaux évitent l'accès direct à cet espace du moment qu'il est possible de remplacer le produit scalaire par une fonction noyau semi-définie positive qui calcule la similarité entre deux éléments directement dans l'espace d'entrée. L'avantage d'utilisation des fonctions noyaux est qu'il est possible d'utiliser des espaces de redescription de haute dimensions (voire infinis), avec une complexité indépendante de la taille de l'espace de redescription, mais qui dépend seulement de la complexité de la fonction noyau elle même.

1.2 Motivations

Une analyse de la littérature sur les noyaux de mots et d'arbres conclut que la plupart de ces noyaux appartiennent à une famille dite noyaux de convolution (Haussler 1999). Cette famille introduit une méthode pour la construction de noyaux sur des ensembles dont les éléments sont des structures discrètes comme les mots, les arbres et les graphes. Les structures discrètes sont des objets récursifs pouvant être décomposés en sous-objets jusqu'à atteindre une unité atomique. Malheureusement, la complexité des noyaux de convolution est très élevée et ne permet pas le calcul de la fonction noyau sur des structures très complexes. Ceci peut empêcher leurs applications dans des scénarios réels. Le développement des techniques d'évaluation efficaces des fonctions noyaux reste un problème ouvert et c'est l'un des principaux objectifs de la présente thèse.

Une autre constatation que nous pouvons tirer de l'analyse de la littérature sur les noyaux de mots et d'arbres est l'existence de toute une panoplie d'algorithmes qui traite ce type de noyaux, chacun avec sa propre technique. Notre second objectif pour ce travail de recherche consiste à proposer une plate-forme générale pour unifier le calcul des noyaux de séquences et qui peut être généralisée afin de définir un nouveau noyau sur des *ensembles de séquences*, en tenant compte de l'efficacité de calcul qui est une propriété clé des méthodes à noyaux. En effet, ceci s'inscrit dans un cadre plus général marqué par une tendance constante vers l'unification des théories qui demeure un défi pour le domaine informatique en général (Hoare 1996) et pour l'apprentissage automatique en particulier (Yang & Wu 2006).

1.3 Contributions

Les contributions de cette thèse peuvent être regroupés en trois intérêts. Par notre première contribution (Bellaouar *et al.* 2014), nous avons essayé de répondre à notre préoccupation, déjà citée, dans la section précédente. Rappelons qu'elle consiste à développer une technique d'évaluation efficace des fonctions noyaux. Nous avons ciblé le noyau sous-séquences de mots (string subsequence kernel, SSK) (Lodhi *et al.* 2002), qui est utilisé avec succès dans plusieurs tâches d'apprentissage automatique.

Dans la littérature, il existe trois implémentations efficaces du noyau SSK, à savoir, l'approche de la programmation dynamique, celle de la programmation dynamique éparse et à base de trie. L'approche de la programmation dynamique (Lodhi *et al.* 2002) propose un algorithme qui utilise une table de programmation dynamique dont l'analyse conduit à une complexité $O(p\,|s||t|)$, où p désigne la longueur des sous-séquences, et $|s|$ et $|t|$ la longueur des mots à comparer.

Pour sa part, l'approche de la programmation dynamique éparse (Rousu & Shawe-Taylor 2005) utilise un ensemble de listes de correspondances avec un arbre de somme d'intervalles (Range sum tree) pour éviter les calculs inutiles. Le temps de calcul du noyau SSK est $O(p|L_1|\log n)$, où $|L_1|$ est la taille de la liste la plus longue et n est la longueur minimale des deux mots s et t.

En ce qui concerne l'approche à base de trie, il existe plusieurs variantes. Nous avons décrit celle présentée dans Rousu & Shawe-Taylor 2005. Pour des raisons d'efficacité, les auteurs restreignent le nombre de « trous » à un entier donné g_{max}, alors le calcul est approximatif. La complexité achevée est $O(\binom{p+g_{max}}{g_{max}}(|s|+|t|))$.

Notre idée de base trouve ses racines dans le lien entre l'algèbre et la géométrie. Autrement, nous projetons un problème d'apprentissage automatique à un problème de géométrie calculatoire. Plus précisément, le calcul du noyau SSK est réduit à un problème de requête d'intervalle bi-dimensionnelle. Nous commençons par la construction d'une liste de correspondances $L(s,t) = \{(i,j) : s_i = t_j\}$ où s et t sont les mots à comparés. Une telle liste ne contient que les données qui contribuent dans le résultat de calcul du noyau SSK.

Ensuite, pour faire certains pré-traitements efficaces, nous construisons un arbre d'intervalles en couches (layered range tree, LRT) dont nous appliquons les algorithmes de géométrie calculatoire correspondants. Dans une perspective d'améliorer l'efficacité de calcul du noyau SSK, nous étendons notre liste de correspondances pour être une liste de listes. La complexité totale du processus de calcul du noyau SSK de longueur p est alors $O(|L|\log|L| + pK)$, où $|L|$ est la longueur de la liste de correspondances et K est le nombre total de points rapportés pour toutes les entrées de $L(s,t)$.

Cependant, la complexité de notre approche dépend du paramètre K désignant le nombre de points rapportés. Dans le pire des cas, K est $O(|L|^2)$ et $|L|$ est $O(|s||t|)$. Il est clair qu'il sera très utile si nous pouvons calculer la somme des valeurs des points rapportés sans les parcourir ou les emmagasiner. Notre

deuxième contribution (Bellaouar *et al.* 2018) vise à outrepasser ce paramètre K. Pour atteindre cet objectif, nous avons étendu la structure de données arbre d'intervalles en couches (LRT) à un arbre d'intervalles de somme en couches (layered Range Sum Tree, LRST) doté des opérations d'agrégation. En utilisant cette nouvelle structure de données, le noyau SSK de longueur p peut être évalué dans un temps $O(p|L|\log|L|)$, où $|L|$ est la longueur de la liste de correspondances.

Par ailleurs, nous présentons des évaluations empiriques de l'approche étendue contre l'approche à base de programmation dynamique (Lodhi *et al.* 2002) et celle à base de programmation dynamique éparse (Rousu & Shawe-Taylor 2005), à la fois sur des données synthétiques et des données réelles extraites des articles de presse. les résultats des expérimentations ont montré l'efficacité de l'approche proposée pour des alphabets de grande taille, à l'exception des mots trop courts.

La troisième contribution (Bellaouar *et al.* 2017) se focalise sur la proposition d'une plate-forme générale pour unifier le calcul des noyaux de séquences qui peut être généralisée dans l'intention de définir un nouveau noyau pour des *ensembles de séquences*. Ceci en tenant compte de l'efficacité de calcul qui est une propriété clé des méthodes à noyaux.

En effet la projection d'un mot s dans un espace de redescription peut être modélisée par une série formelle qui peut être réalisée par un automate pondéré (weighted automaton) A_s représentant toutes les sous-séquences du mot s. Le calcul du noyau toutes sous-séquences $K(s,t)$ entre deux mots s et t est le comportement de l'automate pondéré $A_{s,t} = A_s \cap A_t$. Ainsi, tel que exprimé, il peut être généralisé pour un ensemble de séquences.

Pour une évaluation efficace du noyau, nous proposons une nouvelle technique d'intersection d'automates (par anticipation) qui interdit l'emprunt des ϵ-chemins qui ne se correspondent pas dans les automates A_s et A_t.

Les résultats des expérimentations utilisant la collection Reuters-21578 révèlent que l'évaluation du noyau toutes sous-séquences utilisant notre technique d'intersection proposée est plus rapide que celle utilisant l'intersection standard.

Nous étudions aussi la relation entre notre plate-forme générale et une variété de noyaux de séquences. Enfin, notre contribution est clôturée par une description détaillée de la création d'un nouveau *noyau d'ensembles de séquences* qui peut être vu comme un d'arbre.

1.4 Organisation

La présente thèse est organisée comme suit. La première partie décrit les concepts de base et fournit une étude de l'état de l'art des noyaux de mots et des noyaux d'arbres. Elle comprend les Chapitres 2 - 4.

Nous commençons par un tour d'horizon de l'apprentissage automatique dans le Chapitre 2. Dans le même chapitre, la Section 2.2 présente les fondements théoriques de l'apprentissage automatique. La Section 2.3 introduit le concept des méthodes à noyaux en établissant le lien avec l'apprentissage automatique.

Le Chapitre 3 s'intéresse aux méthodes à noyaux pour le texte. La taxonomie considérée dépend de la représentation des données, à savoir, sacs de mots, vecteurs de concepts, séquences de symboles et sous forme de transducteurs pour lesquelles nous associons respectivement les noyaux des espaces vectoriels, sémantiques, de séquences et rationnels.

Le Chapitre 4 se propose de faire une investigation des méthodes à noyaux d'arbres. Il commence par une introduction de quelques définitions et notations (Section 4.1) ainsi que les paradigmes de conception des noyaux d'arbres (Section 4.2). La section 4.3 décrit les noyaux d'arbre ordonnés selon qu'ils soient réguliers ou approximatifs. Enfin, la Section 4.4 expose les noyaux d'arbres non ordonnés.

La deuxième partie est consacrée à la présentation de la contribution principale, elle comprend les Chapires 5 et 6.

Une des propriétés clés des fonctions noyaux est l'efficacité d'évaluation. Le Chapitre 5 tente de répondre à cette préoccupation pour le noyau SSK (String Subsequence Kernel) qui est largement utilisé dans plusieurs tâches de l'apprentissage automatique. Après une brève présentation des travaux connexes (Section 5.1), la Section 5.2 se focalise sur notre contribution à base de liste linéaire chaînée conjointement avec une structure de données géométrique. La Section 5.3 clôture le Chapitre 5 en exposant une amélioration substantielle de la première contribution entérinée par une étude expérimentale.

Le Chapitre 6 propose une plate-forme générale pour calculer les noyaux de séquences qui peut être utile pour le calcul des noyaux pour des *ensembles de séquences*, ceci en tenant compte de l'efficacité de calcul. Il présente d'abord

quelques informations préliminaires inhérentes aux automates pondérés (Section 6.1). Le reste des sections (Sections 6.2-6.6) est réservé à notre contribution qui comporte la présentation de l'approche proposée, la construction des automates pondérés représentant toutes les sous-séquences d'un mot donnée, l'évaluation des noyaux de séquences à base d'automates pondérés ainsi que son amélioration, la relation avec d'autres noyaux de séquences et enfin, la généralisation pour un nouveau noyau d'ensembles de séquences.

Enfin, le Chapitre 7 couronne notre thèse en rappelant brièvement la problématique de notre recherche, ses objectifs, les contributions principales ainsi que des perspectives sous forme de questions ouvertes et de travail complémentaire.

Première partie

Concepts de base et état de l'art

« *Si j'ai vu si loin, c'est que j'étais monté sur des épaules de géants.* »

Isaac Newton, Mathématicien,
physicien et astronome

Chapitre 2

Apprentissage automatique et noyaux

Le présent chapitre se propose d'introduire les concepts de base nécessaires à la compréhension du travail présenté dans les chapitres qui suivent. Dans un premier temps, nous faisons un tour d'horizon sur la discipline de l'apprentissage automatique ainsi que les fondements théoriques y afférents. Ensuite, nous soulignons les méthodes à noyaux, la substance de notre recherche.

2.1 Apprentissage automatique

Cette section est dédiée à la présentation de l'apprentissage automatique d'une manière progressive commençant par des généralités et se clôturant par le cadre théorique.

2.1.1 Historique et définitions

Est-ce que les machines peuvent-elles penser ? En effet, cette question a été posée par Alan Turing en 1950 (Turing 1995). L'auteur a considéré que les termes machine et penser portent une certaine ambiguïté. Par conséquent, il a essayé de

reformuler le problème sous forme d'un jeu appelé « le jeu de l'imitation » qui implique trois participants : un être humain agissant en tant que juge, un autre être humain et une machine qui tente de convaincre le juge qu'elle est un être humain. Le juge communique avec les deux participants en posant des questions par le biais d'un programme terminal. La machine et l'homme doivent répondre et le juge décidera quelle réponse provient de la machine. Si le juge se trompe, la machine gagne le jeu, et ainsi se manifeste l'intelligence de la machine.

Dans son papier (Turing 1995), Alan Turing a discuté les différentes objections qui affirment que la machine ne peut pas être l'égale de l'être humain ; ceci en présentant des arguments issus de la théologie, des mathématiques, de la conscience, des divers incapacités, de la continuité dans le système nerveux, de l'informalité du comportement et de la perception extra-sensorielle. Malgré qu'il n'a pas donné une réponse à la question posée et il a avoué qu'il ne dispose pas d'arguments convaincants de son propos, mais il a pu bousculer les préjugés philosophiques, théologiques et scientifiques sur l'homme et la machine.

En 1959, Samuel 1959 définit l'apprentissage automatique comme étant un domaine d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés. Il s'est focalisé sur les jeux comme un moyen moins compliqué pour faire apprendre à l'ordinateur des choses. Dans ce contexte, Samuel a développé le premier programme joueur d'échecs chez IBM, avec une conviction que le jeu d'échecs est une activité dans laquelle les joueurs déploient presque toutes leurs ressources cognitives disponibles. Il n'a jamais programmé explicitement les stratégies du jeu, mais à travers l'expérience de jouer, le programme a appris des comportements complexes qui lui ont permis de battre de nombreux adversaires humains.

En 1997, Mitchell 1997a propose une définition plus formelle couramment citée : « On dit d'un programme informatique qu'il apprend à partir d'une expérience **E** étant donné une classe de tâches **T** et une mesure de performance **P**, si sa performance pour les tâches de **T**, telle que mesurée par **P** est améliorée par l'expérience **E** ».

Par exemple, dans la messagerie électronique, chaque courriel peut être un spam ou un courriel ordinaire. Une tâche de **T** peut être : classifier un courriel comme spam ou non-spam. Un programme peut apprendre à effectuer cette tâche en observant une expérience **E**, des courriels déjà libellés comme spam ou

non. Il peut évaluer sa performance en calculant $\mathbf{P}$, le pourcentage des courriels correctement classifiés.

Après ce bref historique, nous pouvons donner une définition plus générale : l'apprentissage automatique est une branche multidisciplinaire de l'intelligence artificielle, elle fait appel principalement à l'informatique, les statistiques, les mathématiques et l'ingénierie. L'apprentissage automatique s'intéresse à la conception des systèmes capables d'apprendre à partir des données en s'entraînant. De tels systèmes doivent s'améliorer avec l'expérience en raffinant un modèle qui peut être utilisé pour prendre des décisions futurs à des problèmes émanant de divers domaines : de la finance à la biologie, de la médecine à la physique, ...

2.1.2 Pourquoi l'apprentissage automatique ?

Quand est-ce que nous aurons besoin de faire appel à l'apprentissage automatique plutôt que directement utiliser la programmation classique ? Un élément de la réponse consiste à savoir quand est-ce que la programmation classique suffit ? Lorsque l'on connaît le bon modèle de traitement des données à utiliser. Sinon, l'apprentissage automatique peut être utilisé. Pour cerner mieux la réponse, deux aspects d'un problème peuvent faire appel à l'utilisation des programmes qui apprennent et s'améliorent sur la base de leur expérience : la complexité du problème et la nécessité d'adaptativité (Shalev-Shwartz & Ben-David 2014).

Complexité du problème

La complexité du problème se manifeste de diverses manières. Il existe plusieurs tâches accomplies par l'être humain d'une façon régulière, mais nous ne disposons pas de compréhension profonde qui nous permet d'élaborer un programme bien défini. Nous pouvons citer les tâches de conduite, de reconnaissance vocale et de la compréhension des images comme exemples.

Une autre famille des tâches est caractérisée par l'analyse de très grandes et complexes ensemble de données qui dépasse la capacité de l'être humain : la transformation des archives médicales en connaissances médicales, les prévisions météorologiques, l'analyse des données génomiques, les moteurs de recherche sur le web et le commerce électronique.

Autrement dit, nous sommes entrés à l'ère du big data. Par exemple, le nombre de sites web a atteint un milliard en septembre 2014 [1] ; 300 heures de vidéo sont téléchargées à YouTube chaque minute [2] ; l'entreprise multinationale, Wal-mart, de grande distribution gère plus d'un million de transactions par heure et dispose d'une base de données contenant plus de 2.5 pétaoctets (2.5×10^{15}) de données [3]. Par ailleurs, Renée J. James, présidente d'Intel corporation, a estimé que la quantité des données numériques (44 zettaoctets, 10^{21}) dépassera le nombre d'étoiles dans l'univers connu [4]. Elle a aussi rappelé de l'histoire de l'informatique depuis le mainframe où les données sont mesurées en kilooctets, en passant par l'architecture client-serveur en mégaoctets, le web en gigaoctets, le web mobile en téraoctets et en fin l'architecture cloud pour le big data.

« Nous nous noyons dans l'information et nous nous affamons aux connaissances » (Naisbitt 1982), c'est bien le proverbe qui représente notre ère de déluge informationnel qui doit faire appel aux méthodes intelligentes d'analyse de données pour relever le défit. C'est ce que fournit l'apprentissage automatique pour découvrir des motifs cachés dans les données.

Adaptativité

Une des caractéristiques de la programmation classique est la rigidité. Une fois que le programme est élaboré et installé, il demeure inchangeable. Cependant, plusieurs tâches peuvent changer dans le temps ou même entre différents utilisateurs.

Par nature, les programmes de l'apprentissage automatique présentent une solution à cette problématique en s'adaptant aux changements de l'environnement en interaction. Nous pouvons citer plusieurs applications typiques qui répondent aux problèmes d'adaptativité : les systèmes e-learning doivent s'adapter aux profiles variés des utilisateurs, les systèmes de reconnaissance de textes manuscrits tiennent en compte la différence entre les écritures manuscrites des différents utilisateurs, les sytèmes d'extraction de données à partir des sites web adaptatifs, et les systèmes de recommandation pour l'aide à la décision, ...

1. http://www.internetlivestats.com/total-number-of-websites/
2. https://www.youtube.com/yt/press/en-GB/, mise à jour le 12/2/2014.
3. http://www.sas.com/resources/whitepaper/wp_46345.pdf
4. Keynote dans Oracle OpenWorld 2014.

2.1.3 Types d'apprentissage automatique

Il existe plusieurs paramètres pour taxonomiser l'apprentissage automatique. Ces paramètres découlent des réponses à certaines questions : comment l'ordinateur sait s'il s'améliore ou non ? et comment sait-il comment s'améliorer ? Dans ce qui suit, nous considérons la première question. La réponse à une telle question produit différents types d'apprentissage automatique, à savoir, l'apprentissage supervisé, non-supervisé, par renforcement et évolutionnaire (Marsland 2009).

Apprentissage supervisé

En pratique, l'apprentissage supervisé est la forme d'apprentissage la plus utilisée. Ce type d'apprentissage est caractérisé par la fourniture, à l'algorithme d'apprentissage, quelques bonnes réponses pour un problème en espérant qu'il arrivera à généraliser la réponse pour d'autres nouvelles situations.

Autrement dit, l'objectif de l'apprentissage supervisé consiste à apprendre une relation d'entrée/sortie $f(\mathbf{x})$ en utilisant un *ensemble d'apprentissage* $\{(\mathbf{x_i}, \mathbf{y_i}), i = 1, \ldots, n\}$, où les entrées $\mathbf{x_i} \in \mathbb{R}^d$ sont des vecteurs d-dimensionnels contenant différentes caractéristiques et les sorties $\mathbf{y_i}$ sont des étiquettes (fournies par le superviseur). Le rôle de cette relation inférée est de prédire une sortie précise pour une nouvelle entrée $\mathbf{x}^*$. Pour mesurer cette précision, on doit définir une *fonction de perte (Loss function)* $\mathbf{L}$ entre la sortie prédite et la sortie réelle ou inversement une *fonction d'utilité* $\mathbf{U} = -\mathbf{L}$. Dans le cas où les sorties $\mathbf{y_i}$ sont discrètes, on parle de problèmes de *classification* et dans le cas où $\mathbf{y_i}$ sont des valeurs continues, la tâche est dite *régression*.

La Figure 2.1 illustre les deux étapes à suivre pour appliquer l'apprentissage supervisé. Dans la phase d'apprentissage, l'algorithme d'apprentissage conçoit un modèle mathématique de dépendance, en se basant sur l'ensemble d'apprentissage donné. Un tel modèle est dit *classifieur* dans le cas de la classification et *fonction* ou *application* dans le cas de régression.

Par ailleurs, il faut noter quelques difficultés liées au processus de généralisation : Les nouvelles entrées peuvent être différentes de celles des données d'apprentissage. Les entrées peuvent être mesurées avec un certain bruit ainsi

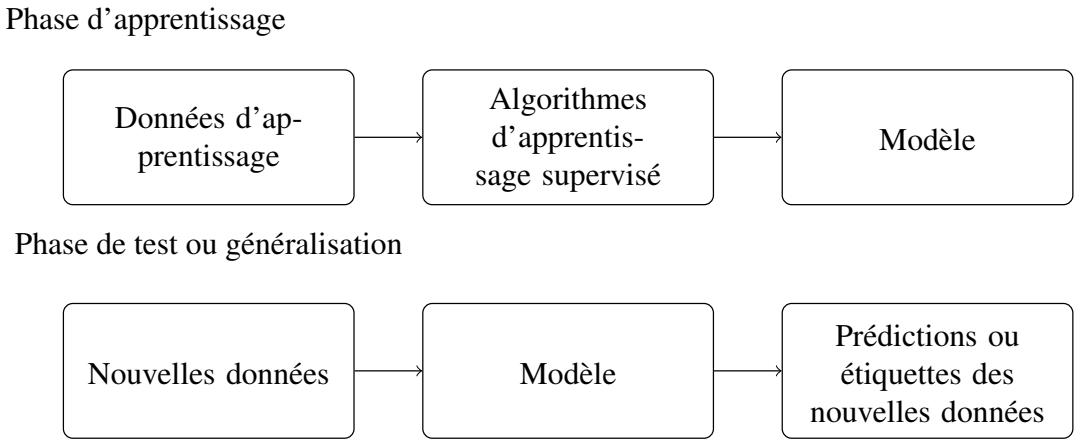


FIGURE 2.1 – Les phases des algorithmes d'apprentissage supervisé. Extrait de Huang 2009.

qu'elles peuvent se présenter sous forme hautement dimensionnelle mais certains composants peuvent être sans intérêt. Dans de telles situations, l'algorithme d'apprentissage ne peut pas prédire des sorties sensibles aux données d'entrées.

- **Classification :** La classification est un problème intéressant dans l'apprentissage supervisé. Elle peut être définie comme étant l'identification de la catégorie ou la classe (discrète) d'une nouvelle donnée observée en se basant sur les données d'apprentissage. Ces dernières doivent avoir les valeurs des attributs (variables indépendantes) ainsi que la classe associée à ces valeurs (variable dépendante). Les algorithmes d'apprentissage de la technique de classification s'appellent *classifieurs*. Formellement, un classifieur peut être défini comme une fonction qui associe un ensemble de valeurs à une classe.

  Typiquement les données d'apprentissage se présentent sous forme de n objets $\mathbf{x_1}, \ldots, \mathbf{x_n}$. Chaque objet $\mathbf{x_i}$ est un vecteur d-dimensionnel fourni avec une étiquette $\mathbf{y_i}$ qui décrira la classe d'appartenance de l'objet $\mathbf{x_i}$. En général, les étiquettes prennent des valeurs entières. Par exemple, s'il existe C classes pour les données, alors $\mathbf{y_i} \in \{1, 2, \ldots, C\}$. Si $C = 2$, alors nous parlons de *classification binaire* ; par contre si $C > 2$, la classification est dite *multiple*.

  Par exemple, pour le diagnostic médical, un patient est décrit par différents descripteurs (âge, taille, poids, sexe, fréquence cardiaque, température corporelle, tension artérielle, ...). Dans un tel cas, le rôle du classifieur consiste à produire un diagnostic sur l'état de santé d'un patient (sain,

grippe, pneumonie, . . .).
Ceci dit, il existe plusieurs types de classifieurs, les plus connus sont les arbres de décision, les règles de décision, le classifieur bayésien naïf, les plus proches voisins, la régression logistique, la machine à vecteurs de support, les réseaux de neurones artificiels et les fonctions discriminantes. Ces classifieurs ont été utilisés avec succès dans plusieurs domaines qui s'étalent depuis le diagnostic automatique des malades où le coût d'erreur est trop conséquent jusqu'à la classification du texte où les données sont relativement complexe.

- **Régression :** Comme dans la classification, nous disposons d'un ensemble de données d'apprentissage décrites par plusieurs descripteurs (variables indépendantes) continues ou discrètes. Ces données d'apprentissage sont dotées d'une variable dépendante continue agissant comme étiquette. La tâche de la régression est de déterminer d'abord une fonction continue en apprenant à partir des données d'apprentissage. Cette fonction s'appelle prédicteur régressionnel, son rôle consiste à prédire des valeurs pour de nouveaux exemples. Par exemple, la prédiction de la température, la pression atmosphérique, la vitesse du vent en se basant sur les données météorologiques antérieures.
  Dans la littérature, il existe plusieurs types de prédicteurs régressionnels qui diffèrent selon la représentation de leurs fonctions. Les plus communs sont la régression linéaire, les arbres de régression, la régression localement pondérée, la machine à vecteurs de support et les réseaux de neurones à propagation avant multi-couches pour la régression.

Apprentissage non supervisé

L'apprentissage non supervisé est caractérisé par l'absence d'un superviseur qui guide le processus d'apprentissage en fournissant une réponse correcte ou même un degré d'erreur pour chaque donnée d'entrée. Autrement dit, les données d'entrée sont dépourvues d'une étiquette, ce qui rend le processus de construction d'un modèle à partir des données une tâche plus difficile. En effet, l'objectif de l'apprentissage non supervisé peut être considéré comme une approximation de la distribution de probabilité ayant généré les données d'entrée ou de découvrir une structure intéressante cachée dans les données.

Il existe plusieurs raisons pour s'intéresser à l'apprentissage non supervisé (Duda *et al.* 2001) :

- Collecter et étiqueter un ensemble de données d'apprentissage de taille importante peut être très coûteux,
- procéder à un apprentissage moins coûteux avec des données non étiquetées, et ensuite utiliser la supervision pour étiqueter les segments trouvés,
- utiliser les méthodes non supervisées pour trouver des descripteurs qui peuvent être utiles pour la classification.

Dans la littérature, il existe plusieurs approches d'apprentissage non supervisé. Les techniques de réduction de dimensions fournissent des informations concernant les relations entre les variables et si elles peuvent être considérées comme des fonctions d'un ensemble petit de variables latentes. L'analyse des clusters tente de regrouper des individus qui se ressemblent en classes. Les règles d'association sont utilisées pour trouver des relations potentiellement importantes entre les données.

- **Analyse des clusters :** Intuitivement, l'analyse des clusters ou clustering est la tâche de regroupement d'un ensemble d'objets de telle manière que les objets similaires se trouvent dans le même groupe et ceux dissimilaires sont séparés dans différents groupes (Shalev-Shwartz & Ben-David 2014). Il existe plusieurs applications pratiques de l'analyse des clusters. À titre d'exemple, elle est souvent utilisée dans l'analyse d'images, segmentation d'images, analyse des réseaux sociaux, classification biologique, analyse des gènes et analyse des crimes.
  Dans la littérature, il existe plusieurs modèles de clustering. Néanmoins, nous présentons la configuration commune suivante : Nous disposons en entrée d'un ensemble d'éléments, $\mathbf{X}$, doté d'une métrique (mesure) pour mesurer la similarité/dissimilarité des éléments à regrouper, $d : \mathbf{X} \times \mathbf{X} \to \mathbb{R}^+$. Cette fonction peut être une distance symétrique satisfaisant $d(\mathbf{x}, \mathbf{x}) = 0$ pour tout $\mathbf{x} \in \mathbf{X}$. Elle doit aussi satisfaire l'inégalité triangulaire $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$. Alternativement, cette métrique peut être une fonction de similarité $s : \mathbf{X} \times \mathbf{X} \to [0, 1]$ symétrique satisfaisant $s(\mathbf{x}, \mathbf{x}) = 1$ pour tout $\mathbf{x} \in \mathbf{X}$. En outre, quelques algorithmes de clustering auront besoin également de spécifier à l'avance un paramètre k indiquant le nombre de clusters.

La sortie du modèle se présente sous forme d'une partition de l'ensemble $\mathbf{X}$ en sous ensembles : $C = (C_1, \dots C_k)$ où $\bigcup_{i=1}^{k} C_i = \mathbf{X}$ et pour tout $i \neq j$, $C_i \cap C_j = \phi$. Par ailleurs, il existe un autre type de sortie où les sous ensembles du domaine $\mathbf{X}$ se présentent sous forme hiérarchique.
En ce qui concerne les méthodes de clustering, deux classes peuvent être identifiées : les méthodes de partitionnement (k-moyennes, k-médoïdes, carte-auto-organisatrice) et les méthodes hiérarchiques (agglomérative et de division).

- **Règles d'association :** L'analyse des règles d'association est un outil populaire dont l'utilisation s'étale de la fouille des bases de données commerciales jusqu'à la fouille d'usage des données du web (web usage mining). Cette technique est fréquemment utilisée pour analyser des transactions. Le but consiste à trouver les valeurs conjointes apparaissant le plus fréquemment dans la collection de données en entrée. Le scénario de l'analyse du panier de la ménagère est un exemple typique pour introduire le principe des règles d'association.
  Ceci dit, il existe plusieurs algorithmes pour l'apprentissage des règles d'association. Les plus prévalents sont les algorithmes *apriori* et *FP-Growth*.

Apprentissage par renforcement

L'apprentissage par renforcement (AR) se situe entre l'apprentissage supervisé et l'apprentissage non supervisé. Un extrait de l'ouvrage de Sutton & Barto 1998 semble une bonne introduction de l'AR. Ce type d'apprentissage consiste à apprendre ce qu'il faut faire, comment transformer des situations à des actions, afin de maximiser quantitativement une récompense. On ne dit pas à l'agent (personne, animal, robot, ...) quelle action à entreprendre, mais il doit découvrir quelles actions donnent plus de récompenses en les essayant. Dans les cas les plus intéressants et stimulants, les actions peuvent affecter non seulement la récompense immédiate, mais aussi la situation suivante et par conséquence, toutes les récompenses à plus long terme. Ces deux propriétés : recherche par essai-erreur et récompense à long terme sont les deux caractéristiques les plus importantes de l'AR.

L'un des défis de l'AR est le compromis entre exploration et exploitation. L'agent préfère exploiter des actions efficaces qu'il a essayé dans le passé, mais

pour les découvrir, il doit les explorer. Le dilemme est que ni l'exploitation ni l'exploration peuvent être poursuivies exclusivement sans faillir à la tâche.

Formellement, le problème de l'AR peut se définir comme suit :

- Un ensemble d'état $\mathbf{S}$ correspondant à la perception que l'agent dispose de son environnement,
- un ensemble d'actions possibles $\mathbf{A}$,
- une fonction de récompense $R : \{\mathbf{S}, \mathbf{A}\} \rightarrow \mathbb{R}$.

L'agent et l'environnement interagissent par pas de temps discrets $(t = 0, 1, 2, \ldots)$. À chaque pas de temps t, l'agent reçoit une certaine représentation de l'état de l'environnement $s_t \in \mathbf{S}$. En fonction de cet état, il choisit une action $a_t \in \mathbf{A}$. Un pas de temps plus tard, l'apprenant reçoit une récompense $r_{t+1} \in \mathbb{R}$ et se trouve dans un nouveau état s_{t+1}. L'agent utilise cette récompense pour évaluer la qualité de sa décision.

Une approche communément utilisée pour l'AR est le Q-learning. C'est une forme d'apprentissage par renforcement dans laquelle l'agent apprend de manière itérative une fonction d'évaluation sur les états et les actions.

L'AR est fréquemment utilisé pour le contrôle des systèmes dynamiques (contrôle de robots) et dans la résolution des problèmes d'optimisation et de jeux (échecs, backgammon, divers jeux de cartes).

Apprentissage évolutionnaire

L'apprentissage évolutionnaire est inspiré de l'évolution biologique qui peut être vu comme un processus d'apprentissage. Les organismes biologiques s'adaptent pour améliorer le taux de leur survie et la chance d'avoir une progéniture dans leur environnement (Marsland 2009). Les algorithmes génétiques modélisent le processus génétique qui peut être considéré comme un signe très marquant de l'évolution. En effet, l'algorithme génétique imite la capacité d'une population d'organismes vivants à s'adapter à son environnement à l'aide des mécanismes de sélection et d'héritage.

La plupart des méthodes des algorithmes génétiques ont au moins les éléments en commun suivants : une population de chromosomes (d'individus ou de séquences), la sélection selon le critère d'adaptation, le croisement pour produire des progénitures et la mutation aléatoire (Mitchell 1998). Chaque chromosome

dans une population des algorithmes génétiques prend, en général, la forme d'une chaînes de bits. Il est considéré comme un point dans l'espace de recherche de solutions candidates. L'algorithme génétique traite les populations en remplaçant une par une autre. Il nécessite, le plus souvent, une fonction d'adaptation (*fitness*) qui associe un score (d'adaptation) pour chaque chromosome dans la population actuelle. L'adaptation du chromosome dépend de la façon dont ce chromosome résout le problème.

En ce qui concerne l'utilisation des algorithmes génétiques, de nombreux chercheurs prétendent que leur utilisation soit bonne dans les cas suivants :

- L'espace de recherche est important,
- l'espace de recherche n'est pas parfaitement appréhendé,
- la fonction d'adaptation est bruitée,
- la tâche à accomplir ne requiert pas un optimum global.

Les applications des algorithmes génétiques sont diverses, à titre non exhaustif, nous pouvons citer les jeux adaptatifs, les simulations biologiques, la reconnaissance des motifs ainsi que la dynamique macroéconomique et de marchés.

2.2 Fondements théoriques de l'apprentissage automatique

Dans la section précédente, nous avons présenté des généralités sur l'apprentissage automatique. Dans la présente, nous nous focalisons sur ses fondements théoriques dont le but consiste à identifier quels types de tâches que nous pouvons espérer apprendre et à partir de quel type de données. Cette espérance est accompagnée avec un type de garantie.

Ces fondements théoriques font appel à plusieurs disciplines, à savoir, l'algorithmique, la théorie de la complexité, la théorie de l'information, ...

2.2.1 Modèle génératif vs. discriminatif

Nous avons déjà introduit dans la Section 2.1.3 une taxonomie de l'apprentissage automatique qualifiant l'apprentissage supervisé comme étant la forme la

plus utilisée. Dans la présente, nous décrivons deux écoles de pensées de ce style d'apprentissage. Ceci, par le biais des modèles génératifs et ceux discriminatifs.

Modèle génératif

Nous rappelons que dans le contexte d'un apprentissage supervisé, l'objectif consiste à apprendre une relation entre les entrées $\mathbf{x_i} \in \mathbb{R}^d$ et les sorties $\mathbf{y_i}, i = 1, \dots n$. L'approche générative vise à élaborer un modèle pour la distribution de la probabilité conjointe $Pr(\mathbf{x}, \mathbf{y})$. Une fois que le modèle soit entraîné, nous pouvons l'utiliser, soit pour la prédiction des $\mathbf{y}$ pour une nouvelle donnée $\mathbf{x}$ ou l'utiliser pour générer artificiellement des échantillons de données.

Formellement, la prédiction peut être décrite en conditionnant et/ou en marginalisant la probabilité conjointe :

$$Pr(\mathbf{y}|\mathbf{x}) = \frac{Pr(\mathbf{x}, \mathbf{y})}{Pr(\mathbf{x})} = \frac{Pr(\mathbf{x}, \mathbf{y})}{\sum_{i=1}^{n} Pr(\mathbf{x}, \mathbf{y} = \mathbf{y}_i)},$$

avec la considération $Pr(\mathbf{x}) = \sum_{i=1}^{n} Pr(\mathbf{x}, \mathbf{y} = \mathbf{y}_i)$. Il est à noter que la probabilité des données $Pr(\mathbf{x})$ est indépendante de $\mathbf{y}$, ainsi, dans la plupart des cas peut être ignorée.

Alternativement, le modèle génératif peur être décrit par le biais de la fonction de vraisemblance $Pr(\mathbf{x}|\mathbf{y})$ et la distribution à priori $Pr(\mathbf{y})$ (car $Pr(\mathbf{x}, \mathbf{y}) = Pr(\mathbf{x}|\mathbf{y})Pr(\mathbf{y})$) comme suit :

$$Pr(\mathbf{y}|\mathbf{x}) = \frac{Pr(\mathbf{x}|\mathbf{y})Pr(\mathbf{y})}{Pr(\mathbf{x})}.$$

Elle suggère la prédiction de la classe avec la *probabilité postérieure maximale* :

$$\mathbf{y}_{ppm} = \arg\max_{\mathbf{y}} Pr(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \frac{Pr(\mathbf{x}|\mathbf{y})Pr(\mathbf{y})}{Pr(\mathbf{x})} = \arg\max_{\mathbf{y}} Pr(\mathbf{x}|\mathbf{y})Pr(\mathbf{y}).$$

Plusieurs méthodes adoptent cette démarche générative. À titre non exhaustif, nous pouvons citer la méthode naïve bayes et la famille des modèles de mixtures.

Modèle discriminatif

L'approche discriminative ne s'intéresse pas à modéliser explicitement les distributions sous-jacentes des variables et des descripteurs du système. Elle tente de modéliser directement la probabilité postérieure $Pr(\mathbf{y}|\mathbf{x})$ ou même apprendre une application directe de l'espace d'entrée $\mathbf{X}$ à l'ensemble de toutes les classes $\mathbf{Y}$. De cette manière, l'approche discriminative évite de modéliser les étapes intermédiaires. Autrement dit, elle évite la modélisation du processus de génération de données. Cette focalisation sur une tâche particulière s'inscrit dans une perspective d'amélioration des performances.

Le processus de modélisation discriminative peut être exprimé formellement comme suit : On se limite à l'ensemble des concepts candidats qu'on appel ensemble d'hypothèses $\mathbf{H} = \{h_w : \mathbf{X} \rightarrow \mathbf{Y} | w \in W\}$, où w est la paramétrisation d'une hypothèse particulière h_w. Notez que ceci peut être interprété comme un modèle de la distribution conditionnelle $Pr_{\mathbf{Y}|\mathbf{X}=\mathbf{x},\mathbf{H}=h} = 1_{\mathbf{y}=h(\mathbf{x})}$, où $1_{\mathbf{y}=h(\mathbf{x})}$ est la fonction indicatrice de l'événement $\mathbf{y} = h(\mathbf{x})$.

Les méthodes populaires de cette approche comprennent la régression logistique, le processus Gaussien, la méthode de séparateurs à vaste marge (SVM) et les réseaux de neurones traditionnels.

2.2.2 Apprentissage PAC

La conception et l'analyse des algorithmes d'apprentissage à partir d'une quantité limitée d'exemples soulève une question fondamentale inhérente à la faisabilité de l'apprentissage (Abu-Mostafa *et al.* 2012). Plusieurs questions dérivent de cette question fondamentale : Ce que peut être appris efficacement ? Ce qui est intrinsèquement difficile à apprendre ? Combien d'exemples sont nécessaires pour apprendre avec succès ? Est ce qu'il existe un modèle général pour l'apprentissage ? (Mohri *et al.* 2012).

La présente section s'intéresse à une formalisation qui tente à répondre aux questions ci-dessus. Cette formalisation est baptisée PAC, pour Probablement Approximativement Correct.

Modèle d'apprentissage PAC

Nous dénotons par $\mathbf{X}$ l'ensemble des *exemples* ou *instances*, $\mathbf{X}$ est parfois dénommé *espace d'entrée*. Un exemple est l'objet à classer. Par exemple dans l'OCR (optical character recognition), les images constituent des exemples. Le *libellé* ou la *classe* est la catégorie que nous essayons de prédire. Pour le problème d'OCR, les libellés sont les lettres ou les chiffres étant représentés. L'ensemble de toutes les classes est dénoté par $\mathbf{Y}$. Pour des raisons de simplicité, nous nous limitons, dans notre contexte, pour $\mathbf{Y} = \{0, 1\}$. Un *concept* $c : \mathbf{X} \to \mathbf{Y}$ est une application à partir des exemples aux libellés. L'application c induit, alors, une partition de $|\mathbf{Y}|$ sous-ensembles de $\mathbf{X}$. Ainsi, dans la littérature, un concept c à apprendre peut être référé en tant que application $c : \mathbf{X} \to \mathbf{Y}$ ou comme un sous ensemble de $\mathbf{X}$. Chaque sous-ensemble est dit *classe de concepts* dénoté par $\mathbf{C}$.

Les exemples observés sont générés par une certaine distribution de probabilité $\mathbf{D}$ sur $\mathbf{X} \times \mathbf{Y}$, fixe mais inconnue. Nous supposons que ces exemples sont *indépendants et identiquement distribués* (iid), c'est-à-dire que chaque exemple est généré par la même distribution $\mathbf{D}$ et la probabilité d'observer un exemple est indépendante des autres exemples observés. Cette propriété iid assure que les exemples générés soient représentatifs.

Le problème d'apprentissage peut être formulé comme suit : l'algorithme d'apprentissage considère un ensemble de concepts candidats que l'on appel ensemble d'*hypothèses* $\mathbf{H}$ qui pourra ne pas coïncider avec $\mathbf{C}$. À titre d'exemple, $\mathbf{H}$ peut être l'ensemble de toutes les fonctions linéaires, l'ensemble de tous les arbres de décision, ...

L'algorithme reçoit un échantillon $\mathbf{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ iid tiré selon $\mathbf{D}$ ainsi que les libellés $(c(\mathbf{x}_1), \ldots, c(\mathbf{x}_m))$ qui sont basés sur un concept cible $c \in \mathbf{C}$ à apprendre. Autrement dit, les libellés sont déterministes. La tâche de l'algorithme d'apprentissage consiste à utiliser l'échantillon $\mathbf{S}$ libellé, qualifié de données d'apprentissage $\mathbf{T} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)\}$, pour choisir une hypothèse $h_s \in \mathbf{H}$ approximative à c, qui constitue le meilleur ajustement aux données.

L'objectif consiste à minimiser l'*erreur de généralisation* ou tout simplement le *risque*. Étant donné $h \in \mathbf{H}$, l'erreur de généralisation de h en ce qui concerne le concept c est défini comme suit : $err\,(h) = Pr_{\mathbf{x}\sim\mathbf{D}}\,[h(\mathbf{x}) \neq c(\mathbf{x})) =$

$E_{\mathbf{x}\sim\mathbf{D}}\left[\mathbf{1}_{h(\mathbf{x})\neq c(\mathbf{x})}\right]$, où $\mathbf{1}_\omega$ est la fonction indicatrice de l'événement ω. Malheureusement, l'erreur de généralisation n'est pas calculable (la distribution $\mathbf{D}$ et le concept cible c sont inconnus). Cependant, l'algorithme peut mesurer l'*erreur empirique* d'une hypothèse sur les données d'apprentissage $\mathbf{T}$: $\hat{err}\,(h) = \frac{1}{m}\left|\{t_i \in \mathbf{T} : h(\mathbf{x}_i) \neq \mathbf{y}_i\}\right| = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}_{h(\mathbf{x}_i)\neq\mathbf{y}_i}$.

Il découle de ce qui précède que l'erreur empirique constitue l'erreur moyenne sur les données d'apprentissage $\mathbf{T}$ alors que l'erreur de généralisation représente l'espérance mathématique de $h(\mathbf{x}) \neq c(\mathbf{x})$ sur la distribution $\mathbf{D}$. Il est facile de noter que pour un $h \in \mathbf{H}$ donné, l'espérance de l'erreur empirique sur un échantillon $\mathbf{S}$ iid est égale l'erreur de généralisation : $E\left[\hat{err}(h)\right] = err(h)$.

Ceci étant dit, le cadre d'apprentissage PAC a été introduit par Valiant 1984. Intuitivement, un problème sera dit PAC-apprenable s'il existe un algorithme capable, avec une probabilité élevée, d'induire une hypothèse dont les prédictions incorrectes sont peu fréquentes. D'une manière plus formelle :

Définition 2.1. Une classe de concepts $\mathbf{C}$ sur l'espace $\mathbf{X}$ est dite PAC-apprenable s'il existe un algorithme $\mathbf{A}$ et une fonction polynomiale $poly(.,.,.,.)$ telle que pour tout $\epsilon > 0$ et $\delta > 0$, pour toute distribution $\mathbf{D}$ sur $\mathbf{X}$, pour tout concept cible $c \in \mathbf{C}$ et pour tout échantillon de taille $m \geq poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n, |c|)$, $\mathbf{A}$ est capable de générer une hypothèse $h_s \in \mathbf{H}$ telle qu'avec une probabilité au moins $1-\delta$, l'erreur de h_s soit inférieur à ϵ : $Pr_{\mathbf{S}\sim\mathbf{D}^m}\left[err\,(h_s) \leq \epsilon\right] \geq 1-\delta$, où $O(n)$ est la borne supérieur de la représentation de tout élément $\mathbf{x} \in \mathbf{X}$ et $|c|$ est le coût maximal de la représentation de $c \in \mathbf{C}$.

Si de plus, l'algorithme $\mathbf{A}$ s'exécute en temps polynomial $poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n, |c|)$, alors $\mathbf{C}$ est dite efficacement PAC-apprenable. Quand un tel algorithme existe, il est qualifié de PAC-apprenable pour $\mathbf{C}$.

Un concept c est PAC-apprenable si l'hypothèse est renvoyé par l'algorithme après l'observation d'un nombre polynomial d'exemples en $1/\epsilon$ et $1/\delta$ est approximativement correct (erreur au plus ϵ) avec une probabilité élevée (au moins $1-\delta$). Ceci justifie la terminologie PAC. L'utilisation de $\delta > 0$ définit la confiance $1-\delta$ et $\epsilon > 0$ la précision.

Borne d'erreur pour un ensemble fini d'hypothèses - cas consistant

Nous commençons par l'analyse d'un cas simple dans lequel l'ensemble des hypothèses est fini. Nous considérons aussi que les hypothèses sont consistantes, c'est-à-dire qu'elles ne commettent pas d'erreurs sur les données d'apprentissage. Formellement, la notion d'hypothèse consistante peut être exprimée ainsi :

Définition 2.2. Une hypothèse h est consistante sur les données d'apprentissage $\mathbf{T}$ avec un concept c, si et seulement si $h(\mathbf{x}) = c(\mathbf{x})$ pour chaque exemple $(\mathbf{x}, c(\mathbf{x}))$ de $\mathbf{T}$.

$$consistant(h, \mathbf{T}) \equiv (\forall(\mathbf{x}, c(\mathbf{x})) \in \mathbf{T})\ h(\mathbf{x}) = c(\mathbf{x}).$$

Les bornes sur l'erreur empirique et l'erreur de généralisation sont données par le théorème suivant :

Théorème 2.3. *Soit* $\mathbf{H}$ *un ensemble fini de fonctions de* $\mathbf{X}$ *à* $\mathbf{Y}$. *Soit* $\mathbf{A}$ *un algorithme qui pour tout concept cible* $c \in \mathbf{H}$ *et un échantillon* $\mathbf{S}$ *iid renvoyant une hypothèse consistante* h_s *:* $e\hat{r}r\,(h_s) = 0$. *Alors pour tout* $\epsilon, \delta > 0$, *l'inégalité* $Pr_{\mathbf{S}\sim\mathbf{D}^m}\left[err\,(h_s \leq \epsilon)\right] \geq 1 - \delta$ *est vérifiée si* $m \geq \frac{1}{\epsilon}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right)$. *Ce résultat général de la complexité sur un échantillon admet la déclaration équivalente suivante en tant que borne de généralisation : pour* $\epsilon, \delta > 0$, *avec une probabilité au moins égale à* $1 - \delta$, $err(h_s) \leq \frac{1}{m}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right)$.

Démonstration. Il faut noter, d'abord, que nous ne savons pas quelle hypothèse consistante est choisie par l'algorithme $\mathbf{A}$. En outre, cette hypothèse dépend de l'échantillon d'apprentissage $\mathbf{S}$. Alors, il faut donner une borne convergente uniforme qui soit valide pour l'ensemble des hypothèses consistantes, dont h_s est incluse. Nous allons donc majorer la probabilité que certaine $h \in \mathbf{H}$ soit consistante et possède une erreur de généralisation supérieure à ϵ :

$$\begin{aligned}
&Pr\left[\exists h \in \mathbf{H} : e\hat{r}r(h) = 0 \wedge err(h) > \epsilon\right] \qquad (2.1)\\
&= Pr[(h_1 \in \mathbf{H} : e\hat{r}r(h_1) = 0 \wedge err(h_1) > \epsilon)\\
&\quad \vee (h_2 \in \mathbf{H} : e\hat{r}r(h_2) = 0 \wedge err(h_2) > \epsilon) \vee \ldots]\\
&\leq \sum_{h\in\mathbf{H}} Pr\left(e\hat{r}r(h) = 0 \wedge err(h) > \epsilon\right) \qquad \text{(Borne de la réunion)}\\
&\leq \sum_{h\in\mathbf{H}} Pr\left(e\hat{r}r(h) = 0 | err(h) > \epsilon\right) \qquad \text{(Probabilité conditionnelle)}
\end{aligned}$$

Pour une hypothèse donnée $h \in \mathbf{H}$ avec $err(h) > \epsilon$:

$$\begin{aligned} Pr\left(err(h) = 0\right) &= Pr\left(h\left(\mathbf{x}_1\right) = c\left(\mathbf{x}_1\right) \wedge \ldots \wedge h\left(\mathbf{x}_m\right) = c\left(\mathbf{x}_m\right)\right) \\ &= \prod_{i=1}^{m} Pr\left(h\left(\mathbf{x}_i\right) = c\left(\mathbf{x}_i\right)\right) \\ &< \left(1-\epsilon\right)^m. \end{aligned}$$

La probabilité exprimée par (2.1) devient alors :

$$\begin{aligned} Pr\left[\exists h \in \mathbf{H} : e\hat{r}r(h) = 0 \wedge err(h) > \epsilon\right] &\leq |\mathbf{H}|\left(1-\epsilon\right)^m \\ &\leq |\mathbf{H}|\,\mathrm{e}^{-\epsilon m} \qquad (\forall x \in \mathbb{R}, 1+x \leq \mathrm{e}^x) \end{aligned}$$

Il suffit de mettre le membre de droite égale à δ :

$$|\mathbf{H}|\,\mathrm{e}^{-\epsilon m} = \delta, \text{ ce qui donne } m = \frac{1}{\epsilon}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right).$$

Nous obtenons donc :

$$Pr\left[\exists h \in \mathbf{H} : e\hat{r}r(h) = 0 \wedge err(h) > \epsilon\right] \leq \delta.$$

La négation de $Pr\left[\exists h \in \mathbf{H} : e\hat{r}r(h) = 0 \wedge err(h) > \epsilon\right]$ nous amène à conclure que :

$$Pr\left[err(h) \leq \epsilon\right] \geq 1-\delta, \text{ si } m \geq \frac{1}{\epsilon}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right).$$

Et en découle :

$$err(h) \leq \frac{1}{m}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right).$$

□

Le Théorème 2.3 montre que dans le cas où l'ensemble des hypothèses est fini, un algorithme consistant $\mathbf{A}$ est PAC-apprenable car la taille de l'échantillon $m \geq \frac{1}{\epsilon}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right)$ est linéaire en $1/\epsilon$ et $1/\delta$. De même, l'erreur de généralisation $err(h) \leq \frac{1}{m}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right)$ est majorée par une borne supérieure qui décroit en fonction de la taille de l'échantillon m. Néanmoins, le coût d'un tel algorithme consistant est l'utilisation d'un ensemble d'hypothèses de taille importante. Cependant la dépendance est logarithmique.

Borne d'erreur pour un ensemble fini d'hypothèses - cas inconsistant

Dans la pratique, les problèmes d'apprentissage peuvent être un peu difficiles ou même les classes de concepts peuvent être plus compliquées que l'ensemble des hypothèses. Ceci implique qu'il peut y avoir aucune hypothèse consistante aux données d'apprentissage. D'où, le recours à des hypothèses inconsistantes avec un nombre réduit d'erreurs sur l'échantillon d'apprentissage pourra être utile.

Dans ce qui suit, nous présentons des garanties pour un ensemble fini d'hypothèses dans le cas inconsistant. Pour ce faire, nous aurons besoin d'un outil plus puissant, l'inégalité de Hoeffding qui met en relation l'erreur de généralisation et l'erreur empirique.

Corollaire 2.4. *Pour $\epsilon > 0$, soit un échantillon $\mathbf{S}$ iid de taille m. Alors pour n'importe quelle hypothèse $h : \mathbf{X} \to \{0, 1\}$, les inégalités suivantes sont vérifiées :*

$$Pr_{\mathbf{S}\sim\mathbf{D}^m}\left(\hat{err}\left(h\right) - err\left(h\right) \geq \epsilon\right) \leq \exp^{-2m\epsilon^2},$$
$$Pr_{\mathbf{S}\sim\mathbf{D}^m}\left(err\left(h\right) - \hat{err}\left(h\right) \geq \epsilon\right) \leq \exp^{-2m\epsilon^2}.$$

Par application de de la propriété de la borne d'union :

$$Pr_{\mathbf{S}\sim\mathbf{D}^m}\left(\hat{err}\left(h\right) - err\left(h\right) \geq \epsilon\right) \leq 2\exp^{-2m\epsilon^2}. \tag{2.2}$$

Nous pouvons utiliser le Corollaire 2.4 pour donner un résultat qui porte sur une hypothèse donnée. Il suffit de mettre $\delta = 2\exp^{-2m\epsilon^2}$ et de résoudre pour ϵ :

Corollaire 2.5. *Pour une hypothèse donnée $h : \mathbf{X} \to \{0, 1\}$, pour tout $\delta > 0$, l'inégalité suivante est vérifiée avec une probabilité au moins égale à $1 - \delta$:*

$$err\left(h\right) \leq \hat{err}\left(h\right) + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}.$$

Comme dans le cas consistant, nous avons besoin d'établir une borne convergente uniforme qui doit être vérifiée pour toutes les hypothèses.

Théorème 2.6. *Soit* $\mathbf{H}$ *un ensemble fini d'hypothèses. Alors, pour tout* $\delta > 0$*, avec une probabilité au moins* $1 - \delta$*, l'inégalité suivante est vérifiée :*

$$\forall h \in \mathbf{H}, err(h) \leq e\hat{r}r(h) + \sqrt{\log \frac{|\mathbf{H}| + \log \frac{2}{\delta}}{2m}}.$$

Démonstration. soient $h_1, \ldots, h_m$ les éléments de $\mathbf{H}$.

$$\begin{aligned}
&Pr\left[\exists h \in \mathbf{H} : |e\hat{r}r(h) - err(h)| > \epsilon\right] \\
&= Pr\left[(e\hat{r}r(h_1) - err(h_1) > \epsilon) \vee \ldots \vee (e\hat{r}r(h_m) - err(h_m) > \epsilon)\right] \\
&\leq \sum_{h \in \mathbf{H}} Pr\left[|e\hat{r}r(h) - err(h)| > \epsilon\right] && \text{(borne de la réunion)} \\
&\leq 2\,|\mathbf{H}| \exp^{-2m\epsilon^2}. && \text{(Corollaire 2.5)}
\end{aligned}$$

Mettons le membre de droite égale à δ et résolvons pour ϵ, nous obtenons :

$$err(h) \leq e\hat{r}r(h) + \sqrt{\log \frac{|\mathbf{H}| + \log \frac{2}{\delta}}{2m}}.$$

□

Nous pouvons constater que cette garantie suggère la recherche d'un compromis entre la réduction de l'erreur empirique et le contrôle de la taille de l'ensemble des hypothèses. Un ensemble d'hypothèses de taille importante est pénalisé par le second terme, mais il peut être utile pour réduire l'erreur empirique (le premier terme). Par contre, pour une erreur empirique similaire, elle suggère l'utilisation d'un ensemble d'hypothèses de taille plus petite.

2.2.3 Généralisation

Nous avons discuté, dans la Section 2.2.2, la faisabilité de l'apprentissage pour un ensemble fini d'hypothèses. Cependant, les ensembles des hypothèses typiques dans l'apprentissage automatique sont infinis.

Dans la présente section, nous nous focalisons sur la généralisation des garanties d'apprentissage sur les ensembles infinis des hypothèses. L'idée principale consiste à réduire le cas infini à l'analyse de l'ensemble fini des hypothèses.

Dans la littérature, il existe plusieurs techniques pour une telle réduction. Dans notre contexte, nous nous limitons à la notion de dimension-VC introduite par Vapnik & Chervonenkis 1971, et depuis, largement étudiée par Vapnik & Chervonenkis 1974; Vapnik 1982. Avant d'entamer la dimension-VC, nous aurons besoin d'introduire les notions de fonction de croissance, éclatement et dichotomie (Abu-Mostafa *et al.* 2012).

Définition 2.7. La **fonction de croissance** $\Pi_{\mathbf{H}} : \mathbb{N} \rightarrow \mathbb{N}$ pour un ensemble d'hypothèses $\mathbf{H}$ est défini par :

$$\exists m \in \mathbb{N}, \Pi_{\mathbf{H}} = max_{\{\mathbf{x}_1,\ldots,\mathbf{x}_m \subseteq \mathbf{H}\}} \left|\{h(\mathbf{x}_1), \ldots, h(\mathbf{x}_m)\}\right|. \tag{2.3}$$

La fonction de croissance $\Pi_{\mathbf{H}}$ est donc le nombre maximum de manières distinctes dans lequel m exemples peuvent être classifiés en utilisant les hypothèses de l'ensemble $\mathbf{H}$.

Définition 2.8. Une **dichotomie** d'un ensemble d'instances $\mathbf{S}$ est une partition de $\mathbf{H}$ en deux sous ensembles disjoints.

Définition 2.9. Un ensemble d'instances $\mathbf{S}$ est **éclaté** par les hypothèses de $\mathbf{H}$ si et seulement si pour chaque dichotomie de $\mathbf{S}$, il existe certaines hypothèses consistantes de $\mathbf{H}$ vis-à-vis cette dichotomie. Autrement dit, quand $\mathbf{H}$ réalise toutes les dichotomies possibles de $\mathbf{S}$, i.e. quand $\Pi(m) = 2^m$.

La Dimension Vapnik-Chervonenkis

La question qui se pose, maintenant, si l'ensemble des hypothèses est infini, quelle mesure de complexité devons-nous utiliser à la place de $\mathbf{H}$ dans l'inégalité $m \geq \frac{1}{\epsilon}\left(\log|\mathbf{H}| + \log\frac{1}{\delta}\right)$? En harmonie, avec le principe suscité, le plus grand sous ensemble $\mathbf{X}$ dont lequel $\mathbf{H}$ peut garantir une erreur d'apprentissage nulle peut constituer une complexité de remplacement. C'est bien cette mesure qui représente la dimension-VC pour l'ensemble des hypothèses $\mathbf{H}$.

Définition 2.10. La dimension-VC d'un ensemble d'hypothèses $\mathbf{H}$ défini sur l'espace d'entrée $\mathbf{X}$ est la taille du plus grand sous ensemble fini de $\mathbf{X}$ **éclaté** par $\mathbf{H}$:

$$VC(\mathbf{H}) = max\left\{m : \Pi(m) = 2^m\right\}.$$

Si $\Pi(m) = 2^m$ pour toute m, alors $VC(\mathbf{H}) = \infty$.

De plus, Le nombre d'exemples d'apprentissage, tirés aléatoirement, suffisant pour PAC-apprendre un concept $c \in \mathbf{C}$ est majoré comme suit (Mitchell 1997b) :

$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(2/\delta\right)\right) + 8VC(\mathbf{H})\log_2\left(13/\epsilon\right).$$

À titre d'exemple, dans le cas où l'ensemble des hypothèses se présente sous forme d'un ensemble d'hyperplans de séparation linéaire $\mathbf{H_n}$ de n dimensions, alors $VC(\mathbf{H_n}) = n + 1$.

2.3 Méthodes à noyaux

Nous commençons par l'établissement du lien entre l'apprentissage automatique et les méthodes à noyaux en les introduisant comme des méthodes d'analyse de données non linéaires. Par la suite, les propriétés des fonctions noyaux ainsi que leurs méthodes de construction sont dépeintes.

2.3.1 Analyse de données non linéaires

Les méthodes classiques de l'apprentissage automatique sont des méthodes linéaires qui se basent sur une classe de fonctions linéaires. Elles sont intensivement étudiées car elles se prêtent à des modèles mathématiques. Dans la littérature, il existe une panoplie de méthodes linéaires, à titre d'exemples, on peut citer : l'analyse en composantes principales (ACP), l'analyse canonique des corrélations (ACC), la régression linéaire, l'analyse discriminante, la régression logistique, ...

Cependant, de nombreuses applications disposent de données avec des régularités et des dépendances non linéaires ; d'où la nécessité de modèles non linéaires. Une astuce consiste à changer la représentation des données par le bias de fonctions non linéaires tout en gardant les régularités et les dépendances inhérentes aux données.

Les méthodes à noyaux rendent possible un telle astuce en projetant les données depuis un espace d'entrée dans un espace de redescription de haute dimension tout en évitant le calcul explicite de cette projection (Figure 2.2).

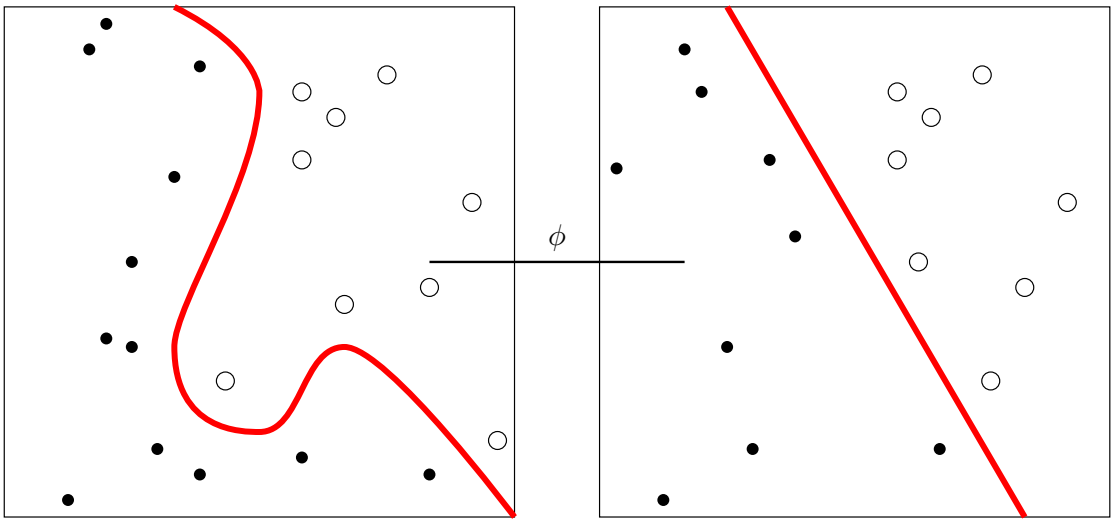


FIGURE 2.2 – Transformation de la recherche d'une régularité non linéaire **(a)** à la recherche d'une régularité linéaire **(b)** via une projection ϕ.

2.3.2 Noyaux et modularité

Nous avons montré dans la section précédente que l'astuce noyau permet de transformer la recherche des régularités non linéaires en régularités linéaires via la projection (virtuelle) de l'espace d'entrée dans un espace de redescription.

Formellement, pour un point $\mathbf{x}$ de l'espace $\mathbf{X}$, nous considérons une fonction de redescription Φ vers un espace de redescription $\mathbf{F}$ doté d'un produit scalaire :

$$\Phi : \mathbf{x} \mapsto \Phi(\mathbf{x}) \in \mathbf{F} \subseteq \mathbb{R}^N.$$

Les algorithmes d'apprentissage n'ont pas besoin de connaître les coordonnées de projections dans l'espace de redescription $\mathbf{F}$. Par contre, ils doivent calculer les produits scalaires des images de points dans un tel espace. La complexité de calculer chaque produit scalaire est proportionnelle à la dimension N de $\mathbf{F}$ qui peut être grande (voire infinie). Cependant, il est possible de calculer efficacement ce produit scalaire en utilisant l'espace d'entrée grâce aux *fonctions noyaux* :

$$k(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle . \tag{2.4}$$

Ceci étant dit, les méthodes à noyaux offrent une plate forme modulaire telle qu'illustré dans la Figure 2.3. Dans un premier temps, les données sont traitées en utilisant un noyau pour créer une *matrice de noyau* ou *matrice de Gram*. Ensuite,

une variété d'algorithmes d'apprentissage peuvent être utilisés pour produire une *fonction d'apprentissage*. Autrement dit, tout noyau peut être utilisé avec tout algorithme d'apprentissage.

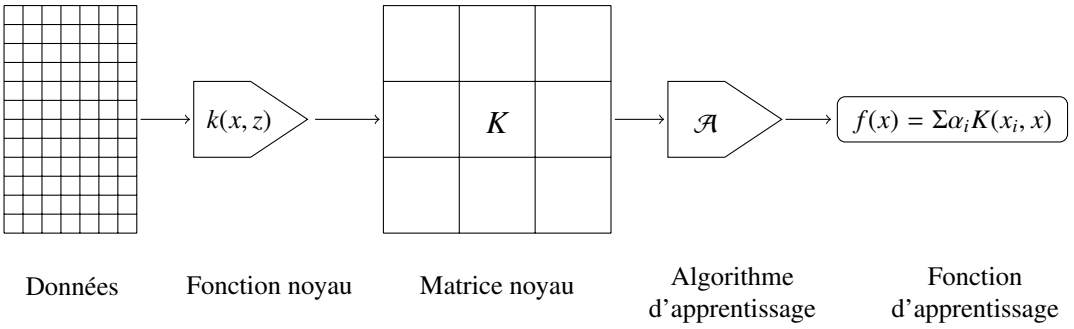


FIGURE 2.3 – Modularité dans les applications des méthodes à noyaux. Extrait de Shawe-Taylor & Cristianini 2004.

2.3.3 Propriétés des noyaux

L'Équation 2.4 a défini la fonction noyau comme étant le produit scalaire des images de deux points dans un espace de redescription. Dans la présente section, nous discutons les propriétés des noyaux en considérant les propriétés du produit scalaire et sa relation avec la notion semi-définie positive des matrices de Gram.

Notions de l'algèbre linéaire

Cette section s'intéresse à rappeler quelques notions émanant de l'algèbre linéaire et qui constituent les fondements des propriétés des fonctions noyaux (Strang 2009; Mohri *et al.* 2012).

Définition 2.11. Un espace de produit scalaire est un espace vectoriel $\mathbf{E}$ doté d'une fonction $\langle ., . \rangle : \mathbf{E} \times \mathbf{E} \to \mathbb{R}$ tels que :

- Pour tous $\mathbf{x}, \mathbf{y} \in \mathbf{E}$: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ (symétrie),
- pour tous $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{E}$ et $\alpha \in \mathbb{R}$: $\langle \alpha \mathbf{x}, \mathbf{y} \rangle = \alpha \langle \mathbf{x}, \mathbf{y} \rangle$ et $\langle \mathbf{x} + \mathbf{z}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$ (linéarité), et
- pour tous $\mathbf{x} \in \mathbf{E}, \mathbf{x} \neq \mathbf{0} \Rightarrow \langle \mathbf{x}, \mathbf{x} \rangle > 0$ (positivité).

Par ailleurs, l'espace de produit scalaire est dit strict si $\langle \mathbf{x}, \mathbf{x} \rangle = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$.

La fonction $\langle ., . \rangle$ est dite produit scalaire. Chaque espace de produit scalaire est un espace linéaire normé avec la norme euclidienne $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$

et par conséquent un espace métrique avec la distance $d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\| = \sqrt{\langle \mathbf{x} - \mathbf{z}, \mathbf{x} - \mathbf{z} \rangle}$.

Un espace de produit scalaire est parfois appelé espace de Hilbert. Dans la littérature, d'autres définitions exigent les propriétés de complétude et de séparabilité.

Définition 2.12. Un espace de Hilbert $\mathbf{H}$ est un espace de produit scalaire avec les propriétés de complétude et de séparation. La complétude fait référence à la propriété que chaque séquence de Cauchy $\{h_n\}_{n \geq 1}$des éléments de $\mathbf{H}$ converge à un élément $h \in \mathbf{H}$, où une séquence de Cauchy qui satisfait la propriété :

$$\sup_{m>n} \|h_n - h_m\| \to 0, \text{ quand } n \to \infty.$$

Un espace $\mathbf{H}$ est séparable si pour tous $\epsilon > 0$, il existe un ensemble fini d'éléments $h_1, \ldots, h_N$ tel que pour tous $h \in \mathbf{H}$:

$$\min_i \|h_i - h\| < \epsilon.$$

Définition 2.13. Matrice de Gram : Considérons l vecteurs $\mathbf{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ dans l'ensemble d'espace d'entrée $\mathbf{X}$. La matrice $\mathbf{G}$, $l \times l$, des produits scalaires entre ces vecteurs (les entrées $G_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$) est appelée matrice de *Gram* associée à $\mathbf{S}$.

Dans le cas des fonctions noyaux, $G_{ij} = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = K(\mathbf{x}_i, \mathbf{x}_j)$, la matrice de Gram est souvent dénommée matrice de *noyau*. Cette matrice contient toute l'information utile pour calculer la distance entre toutes les paires de données. De plus, elle est symétrique : $G_{ij} = G_{ji}$.

Il est clair que les algorithmes d'apprentissage utilisent exclusivement les informations contenues dans la matrice de Gram. Par conséquent, il est possible d'estimer certaines propriétés de l'apprentissage à partir des propriétés d'une telle matrice. Il est possible, également, de modifier l'apprentissage en effectuant certaines opérations sur la matrice de Gram.

Ceci dit, le choix de la fonction noyau peut influencer fortement l'opération d'apprentissage. Il doit être fait de telle manière que la matrice de Gram résultante doit refléter les régularités et les dépendances des données. Ainsi, il est important d'étudier les propriétés des matrices de Gram.

Définition 2.14. Valeurs propres, vecteurs propres et spectre : Soit $\mathbf{A}$ une matrice carrée $n \times n$, un scalaire λ est une *valeur propre* de $\mathbf{A}$ s'il existe un vecteur $\mathbf{v}$ non nul tel que $\mathbf{Av} = \lambda\mathbf{v}$. Dans ce cas, $\mathbf{v}$ est dit *vecteur propre* associé à la valeur λ. L'ensemble des valeurs propres d'une matrice $\mathbf{A}$, noté $\lambda(\mathbf{A})$ est dit *spectre* de $\mathbf{A}$.

Plusieurs théorèmes sont associés aux valeurs propres :

Théorème 2.15. *Si $\mathbf{A}$ est une matrice triangulaire, triangulaire supérieure, triangulaire inférieure ou diagonale, les valeurs propres de $\mathbf{A}$ sont les entrées de la diagonale de $\mathbf{A}$.*

Théorème 2.16. *$\lambda = 0$ est une valeur propre de $\mathbf{A}$ si $\mathbf{A}$ est une matrice singulière (non invertible).*

Théorème 2.17. *Les valeurs propres d'une matrice symétrique sont orthogonales, mais seulement pour des valeurs propres distinctes.*

Théorème 2.18. *Les valeurs propres d'une matrice symétrique sont réelles.*

Théorème 2.19. *Le déterminant de la matrice $\mathbf{A}$, $|det(\mathbf{A})|$, est le produit des valeurs absolues des valeurs propres de $\mathbf{A}$.*

Matrices semi-définies positives

Nous commençons par la définition d'une matrice semi-définie positive, puis nous démontrons que la matrice de Gram et celle du noyau sont semi-définies positives (Shawe-Taylor & Cristianini 2004).

Définition 2.20. Matrice semi-définie positive : Une matrice symétrique est semi-définie positive, si toutes ses valeurs propres sont non négatives.

Pour reconnaître ce type de matrices, il suffit de calculer les valeurs propres et tester si $\lambda > 0$. Cependant, il faut éviter ce type de solutions car le calcul des valeurs propres n'est pas aussi facile, notamment, pour les matrices de dimensions importantes. Dans le cas où nous aurons besoin de connaître, seulement, si les valeurs propres sont positives, il existe des méthodes très rapides :

Définition 2.21. Une matrice $\mathbf{A}$ est semi-définie positive si $\mathbf{v}^T\mathbf{Av} \geq 0, \mathbf{v} \neq 0$.

Proposition 2.22. *Les matrices de Gram et de noyau sont semi-définies positives.*

Démonstration. Nous considérons le cas des matrices de noyau :

$$\mathbf{G}_{ij} = \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \text{ , pour } j = 1, \ldots, l.$$

Pour chaque vecteur $\mathbf{v}$, nous avons :

$$\begin{aligned} \mathbf{v}^T \mathbf{G} \mathbf{v} &= \sum_{i,j=1}^{l} v_i v_j \mathbf{G}_{ij} = \sum_{i,j=1}^{l} v_i v_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{l} v_i \Phi(\mathbf{x}_i), \sum_{j=1}^{l} v_j \Phi(\mathbf{x}_j) \right\rangle \\ &= \left\| \sum_{i=1}^{l} v_i \Phi(\mathbf{x}_i) \right\|^2 \geq 0. \end{aligned}$$

□

Caractérisation des fonctions noyaux

Nous ne disposons jusqu'à présent que d'un seul moyen pour vérifier si une fonction est un noyau. Il consiste, d'abord, de construire l'espace de redescription en effectuant la projection des descripteurs et ensuite calculer le produit scalaire des images correspondantes. Par la présente section, nous introduisons une méthode alternative pour démontrer qu'une fonction est un noyau. Cette méthode repose sur la relation entre le produit scalaire et la propriété semi-définie positive des matrices induites.

Définition 2.23. Fonction semi-définie positive finie : Une fonction $k : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$ est semi-définie positive finie si elle est symétrique, i.e., $\forall\, \mathbf{x}, \mathbf{z} \in \mathbf{X} / \; k(\mathbf{x}, \mathbf{z}) = k(\mathbf{z}, \mathbf{x})$ et la matrice construite par application de la fonction k à un sous ensemble fini de l'espace d'entrée $\mathbf{X}$ est semi-définie positive.

Définition 2.24. Fonction noyau reproduisant : Soit $\mathbf{H}$ un espace de Hilbert de fonctions sur $\mathbf{X}$. La fonction $k : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$ est un noyau reproduisant de $\mathbf{H}$ si pour chaque $\mathbf{x} \in \mathbf{X}$, la fonction $h_{\mathbf{x}}(.) = k(., \mathbf{x})$ est dans $\mathbf{H}$ et pour chaque $\mathbf{z} \in \mathbf{X}$ et chaque $h_{\mathbf{x}}(.) \in \mathbf{H}$ $h(\mathbf{z}) = \langle h, k(., \mathbf{z}) \rangle_{\mathbf{H}}$.

Le Théorème 2.25 montre que la propriété semi-définie positive est une caractérisation des fonctions noyaux :

Théorème 2.25. *Une fonction $k : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ (soit continue ou avec un domaine dénombrable) est semi-définie positive finie si et seulement s'il existe un espace de Hilbert $\mathbf{H}$ avec une application $\Phi : \mathbf{X} \to \mathbf{H}$ tel que $k(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle$.*

Dans ces conditions, le noyau semi-défini positif est dit noyau de Mercer (Murphy 2012).

2.3.4 Construction de Noyaux

Nous avons vu dans la Section 2.3.3 que la propriété semi-définie positive finie est une caractérisation des fonctions noyaux. Bien que cette caractérisation est utile pour décider si une fonction candidate est un noyau valide ou non, une de ses principales conséquences est qu'elle peut être utilisée pour construire de nouvelles fonctions noyaux à partir de celles déjà connues en utilisant des opérations qui préservent cette caractérisation (semi-définie positive finie).

La proposition 2.26 montre que les fonctions noyaux satisfassent à un certain nombre de propriétés de fermetures (Shawe-Taylor & Cristianini 2004; Cornuéjols & Miclet 2010) :

Proposition 2.26. *(**Propriétés de fermeture**) Soient k_1 et k_2 deux fonctions noyaux sur $\mathbf{X} \times \mathbf{X}$, $\mathbf{X} \in \mathbb{R}^d$, $c \in \mathbb{R}^+$, $f(.)$ une fonction réelle sur $\mathbf{X}$, $poly(.)$ un polynôme avec des coefficients positifs ou nuls, $\Phi(.)$ une fonction de $\mathbf{X}$ sur $\mathbb{R}^D$, , $\mathbf{A}$ une matrice semi-définie positive, $\mathbf{x_a}$ et $\mathbf{x_b}$ des variables avec $\mathbf{x} = (\mathbf{x_a}, \mathbf{x_b})$ et k_a et k_b des fonctions noyaux dans leur espace respectif. Alors les fonctions suivantes sont des fonctions noyaux :*

(i) $k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}')$

(ii) $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\, k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$

(iii) $k(\mathbf{x}, \mathbf{x}') = poly(k_1(\mathbf{x}, \mathbf{x}'))$

(iv) $k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$

(v) $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$

(vi) $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$

(vii) $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}'$

(viii) $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x_a}, \mathbf{x}'_\mathbf{a}) + k_b(\mathbf{x_b}, \mathbf{x}'_\mathbf{b})$

(ix) $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x_a}, \mathbf{x}'_\mathbf{a}) k_b(\mathbf{x_b}, \mathbf{x}'_\mathbf{b})$

Nous laissons le soin au lecteur de voir la preuve de la Proposition 2.26 dans (Shawe-Taylor & Cristianini 2004).

Une alternative pour construire des fonctions noyaux consiste à suivre les principes suivants (Lampert 2009) :

(i) Pour chaque $\Phi : \mathbf{X} \to \mathbb{R}^D$, $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$ est un noyau.

(ii) Si $d : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ est une fonction distance, i.e.
- $d(\mathbf{x}, \mathbf{x}') \geq 0$ pour tous $\mathbf{x}, \mathbf{x}' \in \mathbf{X}$,
- $d(\mathbf{x}, \mathbf{x}') = 0$ seulement pour $\mathbf{x} = \mathbf{x}'$,
- $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}', \mathbf{x})$ pour tous $\mathbf{x}, \mathbf{x}' \in \mathbf{X}$,
- $d(\mathbf{x}, \mathbf{x}') \leq d(\mathbf{x}, \mathbf{x}'') + d(\mathbf{x}'', \mathbf{x}')$ pour tous $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathbf{X}$,

alors $k(\mathbf{x}, \mathbf{x}') = \exp(-d(\mathbf{x}, \mathbf{x}'))$ est un noyau.

À titre d'illustration, en utilisant une combinaison des principes de construction des noyaux sus-cités, nous pouvons vérifier que les fonctions suivantes sont des noyaux valides :

- Toute combinaison linéaire $\sum_j \alpha_j k_j$ avec k_j des noyaux et $\alpha_j \geq 0$,
- tout noyau polynomial $k(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^m$, $m \in \mathbb{N}$,
- le noyau Gaussien $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2})$.

2.4 Conclusion

Ce chapitre a introduit les concepts de base nécessaire à la compréhension de la suite de ce manuscrit.

Nous avons commencé par la présentation de l'apprentissage automatique selon deux angles de vues. La première s'est intéressée aux généralités, notamment les scénarios où il est nécessaire de faire appel à l'apprentissage automatique ainsi que ses types. tandis que, la seconde s'est focalisée sur les fondements théoriques de l'apprentissage automatique, en particulier la formalisation PAC (Probablement Approximativement Correct) et la notion de dimension-VC.

Le chapitre est clôturé par l'établissement d'un lien entre l'apprentissage automatique et les méthodes à noyaux en les présentant comme des méthodes d'analyse de données non linéaires. Ensuite, les propriétés des fonctions noyaux et les méthodes de leurs constructions sont dépeintes.

Il convient maintenant d'entamer, dans les deux chapitres qui suivent, une revue de la littérature concernant les noyaux de mots et d'arbres.

Chapitre 3

Noyaux pour le texte

LES trois dernières décennies sont marquées par le phénomène de l'explosion de données, notamment celle textuelle qui est considérée parmi les types de données les plus importants. La production de cette immense quantité de données a rendu les opérations d'analyse et de classification manuelles de ces ressources très difficile, voire impossible. Ceci a capturé l'intérêt de la communauté informatique, en générale, et celle de l'intelligence artificielle, en particulier. Plusieurs algorithmes d'apprentissage automatique ont été développés dans une perspective d'obtenir des résultats satisfaisants en matière d'analyse de données.

Par ailleurs, les algorithmes d'apprentissage automatique sont développés pour être appliqués sur des données vectorielles. Cependant, pour plusieurs types de données, cette représentation n'est pas disponible à l'état brut. Il faut capturer, implicitement ou explicitement, un espace de descripteurs plus approprié. Un tel espace est souvent de haute dimension où les algorithmes d'apprentissage automatiques ne peuvent pas être appliqués, essentiellement, pour des raisons calculatoires. Les méthodes à noyaux offrent une reformulation permettant d'éviter la construction explicite des vecteurs de descripteurs permettant ainsi aux algorithmes d'être exécutés efficacement.

Dans le présent chapitre, nous nous focalisons sur les méthodes à noyaux qui sont couramment utilisées pour le texte. Notre démarche consiste à introduire les noyaux pour le texte selon la représentation des données. Nous considérons

quatre représentations essentielles, à savoir, sacs de mots, vecteurs de concepts, séquences de symboles et sous forme de transducteurs pour lesquelles nous associons respectivement les noyaux des espaces vectoriels, sémantiques, de séquences et rationnels.

3.1 Représentation de texte

Les données textuelles doivent être pré-traitées avant de servir d'entrées pour un système d'apprentissage. La phase de représentation de texte est la plus importante. En effet, il existe plusieurs façons de représenter une entité textuelle, cela dépend souvent, de l'application l'utilisant. Dans notre contexte, nous considérons une liste non exhaustive : représentation en « sac de mots », en vecteur de concepts, en séquences de symboles et sous forme de transducteurs.

3.1.1 Représentation en « sac de mots »

La représentation en « sac de mots » ou modèle d'espace vectoriel (VSM, Vector Space Model en anglais) est introduite par Salton *et al.* 1975. L'idée principale consiste à représenter une entité textuelle notée d (pour document) sous forme d'un vecteur indexé par les mots (termes) qu'elle contient. L'élément $tf(t_i, d)$ est le nombre d'occurrences du terme t_i dans le document d. La nature du terme t_i est, en général, le résultat des opérations linguistiques lors de la phase de sélection des attributs dans le processus du prétraitement (Tokenisation, lemmatisation, filtrage des mots vides, ...).

Formellement, cette représentation peut être modélisée comme étant une projection d'un document d dans un espace de haute dimension :

$$d \mapsto \Phi(d) = (tf(t_1, d), tf(t_2, d), \ldots, tf(t_N, d)) \in \mathbb{R}^N$$

La taille N du vecteur est, en général, assez grande. Ceci implique que la représentation VSM engendre des vecteurs creux dont la plupart des entrées sont nulles. En outre, Il est clair que, dans cette représentation, l'ordre des mots et les informations grammaticales sont ignorés, ainsi, il est impossible de reconstruire le document original à partir de sa représentation en « sac de mots ».

3.1.2 Représentation en vecteur de concepts

L'inconvénient majeur de la représentation en « sac de mots » est qu'elle ignore le contenu sémantique des mots. À titre d'exemples, les mots synonymes sont assignés à des composantes distinctes alors qu'ils désignent la même chose. Tandis que pour les homonymes (polysémie), la représentation VSM ne pourra pas faire face à cette situation, car ce modèle ne prend pas en compte l'information contextuelle nécessaire pour la désambiguïsation du sens.

Une alternative consiste à garder la même représentation vectorielle mais cette fois-ci on considère les concepts à la place des mots. Les concepts sont une représentation plus compacte et plus sémantique d'une entité textuelle. Cette représentation de concepts peut être obtenue en appliquant différentes opérations sur la représentation VSM :

- appliquer différents schémas de pondération w_i aux termes. Une forme de cette pondération est le filtrage des mots vides ;
- utiliser des réseaux sémantiques comme les thésaurus et les ontologies pour extraire des concepts ;
- utiliser des méthodes de réduction de dimensions qui conservent le maximum d'information de la collection de données.

La manière la plus simple de concevoir cette représentation est de penser à une transformation linéaire de la représentation VSM :

$$\tilde{\Phi}(d) = \Phi(d)S, \tag{3.1}$$

où S est une matrice $N \times k$ dite sémantique et k est le nombre de concepts.

3.1.3 Représentation en séquences de symboles

Les deux représentations précédentes ne tiennent pas compte de l'ordre dans les entités textuelles. Pour la représentation en séquence de symboles, l'aspect séquentiel reprend son importance. Un symbole désigne une lettre sur un alphabet. Il peut être un caractère, un mot ou un concept. Cette représentation est adoptée dans plusieurs domaines : Dans les applications bio-informatiques, les protéines peuvent être représentées comme une séquence d'acides aminés ; l'ADN génomique comme des séquences de nucléotides. Dans le domaine de

traitement automatique de la langue naturelle, un document peut être représenté comme une séquence de caractères, de mots, de phrases ou de paragraphes. Ceci étant dit, l'aspect formel des séquences de symboles est discuté dans la section réservée aux noyaux de séquences (Section 3.4.2).

3.1.4 Représentation sous forme de transducteurs

Une autre alternative de la représentation des entités textuelles en « sac de mots » est la représentation sous forme d'automates ou de transducteurs. Dans la présente section, nous nous intéressons aux transducteurs car ils sont utilisés par les noyaux rationnels que nous allons présenter ultérieurement.

Les transducteurs offrent un formalisme de traitement des entités textuelles et la description du comportement dynamique. Ils s'adaptent mieux aux applications récentes de traitement de texte, de la parole et du calcul biologique nécessitant l'analyse des séquences de longueur variables (Cortes *et al.* 2004). Dans la suite, nous présentons les notions de base inhérentes aux transducteurs abordées essentiellement dans Kuich & Salomaa 1986; Sakarovitch & Thomas 2009.

Un monoïde est une structure algébrique avec une signature $\langle M, \circ, 1\rangle$, où $\circ$ est une opération binaire associative sur un ensemble M et 1 est l'élément neutre. i.e. que les axiomes suivants sont valables pour tout $a, b, c \in M$: $a \circ (b \circ c) = (a \circ b) \circ c$ et $a \circ 1 = 1 \circ a = a$. Un monoïde est dit commutatif si et seulement si $a \circ b = b \circ a$ pour tout a et $b \in M$.

Dans notre contexte, le type le plus important des monoïdes est le monoïde libre $\langle \Sigma^*, \text{concaténation}, \epsilon\rangle$ généré par un alphabet Σ muni de la concaténation. L'élément neutre est le mot vide dénoté ϵ.

Un semi-anneau est une structure algébrique avec une signature $\langle \mathbb{S}, +, \cdot, 0, 1\rangle$ qui satisfait les conditions suivantes :

i, $\langle \mathbb{S}, +, 0\rangle$ est un monoïde commutatif,

ii, $\langle \mathbb{S}, \cdot, 1\rangle$ est un monoïde,

iii, la seconde loi $(\cdot)$ est distributive sur la première $(+)$,

iv, $0 \cdot a = a \cdot 0 = 0$ pour tout $a \in \mathbb{S}$.

Un semi-anneau est dit commutatif si et seulement si $a \cdot b = b \cdot a$ pour tous $a, b \in \mathbb{S}$.

Un transducteur pondéré sur un semi-anneau $\mathbb{S}$, ou $\mathbb{S}$-automate est un 8-uplet $T = (\Sigma, \Delta, Q, E, I, F, \lambda, \rho)$, où Σ est un alphabet d'entrée fini, Δ est un alphabet de sortie fini, Q est un ensemble fini d'états, $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{S} \times Q$ est un ensemble fini de transitions, I (resp. F) $\subseteq Q$ est un ensemble d'états initiaux (resp. finaux) et $\lambda : I \to \mathbb{S}$ (resp $\rho : F \to \mathbb{S}$) est la fonction de pondération pour les états initiaux (resp. finaux). Dans cette définition, on admet les ϵ-transitions.

Nous pouvons considérer les transducteurs pondérés selon deux angles de vues. La première considère le transducteur comme un $\mathbb{S}$-automate accepteur disposant de deux bandes pour lire une paire de mots $\Sigma^* \times \Delta^*$. Dans ce cas le transducteur accepte un ensemble de paires de mots et produit une valeur dans $\mathbb{S}$. La seconde considère un transducteur comme un automate sur une seule bande produisant un poids dans le semi-anneau $\mathbb{S}_{Rat}\Delta^*$ des séries formelles.

3.2 Noyaux pour « sac de mots »

Les noyaux pour « sac de mots » ou noyaux VSM reposent sur la représentation des entités textuelles en « sac de mots ». Dans cette représentation, un dictionnaire désigne un ensemble permanent de termes prédéfinis. Dans la pratique, on considère les termes apparaissant dans le corpus qui contient tous les documents en cours de traitement. Un corpus de l documents peut être représenté par une matrice document-terme dont les lignes sont indexées par les documents du corpus et les colonnes sont indexées par les termes du dictionnaire :

$$D = \begin{bmatrix} tf(t_1, d_1) & \cdots & tf(t_N, d_1) \\ \vdots & \ddots & \vdots \\ tf(t_1, d_l) & \cdots & tf(t_N, d_l) \end{bmatrix}$$

La matrice terme-document D^T est la transposée de la matrice D. La matrice $D^T D$ est la matrice terme-terme. Tandis que la matrice DD^Test la matrice

document-document, en effet, c'est bien la matrice noyau. Le noyau VSM correspondant entre deux documents d_1 et d_2 est :

$$\begin{aligned} k(d_1, d_2) &= \langle \Phi(d_1), \Phi(d_2) \rangle \\ &= \sum_{j=1}^{N} tf(t_j, d_1) tf(t_j, d_2) \end{aligned}$$

Pour calculer efficacement un tel noyau, on doit considérer la version « creuse » du produit scalaire. Cette version consiste à convertir un document d en une liste $L(d)$ à la place d'un vecteur $\Phi(d)$ en définissant un ordre sur les mots. De cette manière, il est simple de calculer le noyau comme suit :

$$k(d_1, d_2) = A(L(d_1), L(d_2)),$$

où l'algorithme A parcourt les listes $L(d_1)$ et $L(d_2)$ en calculant le produit des fréquences des termes qui correspondent. Ainsi, bien que nous travaillons dans un espace de haute dimension, le calcul de la projection et le produit scalaire peuvent être implémentés dans un temps proportionnel à la longueur des deux documents $O(|d_1| + |d_2|)$, où $|d_i|$ désigne le nombre de termes différents dans le document d_i.

Le noyau VSM est souvent utilisé dans des applications d'apprentissage automatique telles que le filtrage des spams (Drucker *et al.* 1999), la reconnaissance des entités nommées (Isozaki & Kazawa 2002) et la classification des hypertextes (Joachims & Cristianini 2001).

3.3 Noyaux sémantiques

Nous avons vu que la stratégie des noyaux sémantiques repose sur l'extension de la représentation VSM telle que introduite par l'Équation (3.1). Cette extension est justifiée par le fait que les composantes de la représentation vectorielle sont toujours plus au moins corrélées, en effet c'est la nature du langage

humain. Le noyau correspondant prend la forme suivante :

$$\begin{aligned}\tilde{k}(d_1, d_2) &= \left\langle \tilde{\Phi}(d_1), \tilde{\Phi}(d_2) \right\rangle \\ &= \tilde{\Phi}(d_1)\ \tilde{\Phi}(d_2)^T \\ &= \Phi(d_1) S\ S^T \Phi(d_2)^T\end{aligned}$$

Différents choix de la matrice S mènent à des variantes du noyau VSM. Elle est définie dans Shawe-Taylor & Cristianini 2004 comme $S = RP$, où R est une matrice diagonale des poids des termes ou de pertinence, alors que P est une matrice de proximité exprimant l'écart sémantique entre les différents termes du dictionnaire.

Dans la représentation VSM, la fréquence d'un terme qui est une propriété locale d'un document n'a pas l'aptitude d'établir une thématique du document. Pour ce faire, elle a besoin de porter une information absolue par rapport à tous les documents du corpus. Plusieurs mesures ont été proposées dans ce contexte, à savoir, l'entropie ou la fréquence d'un mot dans les documents du corpus. Elle peut être utilisée pour quantifier la quantité d'information portée par un mot. L'information mutuelle, une autre mesure absolue, est largement utilisée pour les tâches de la classification. En fin, à titre d'illustration, nous considérons la mesure absolue idf (inverse document frequency) qui pèse les termes en fonction de leurs fréquences de documents inverses :

$$w(t) = idf(t) = \ln \frac{l}{df(t)},$$

où l désigne le nombre de documents dans le corpus et $df(t)$ est le nombre de documents contenant le terme t. Si nous considérons juste la matrice diagonale de pondération des termes R avec $R_{tt} = idf(t)$, le noyau sémantique associé peut être exprimé comme suit :

$$\begin{aligned}\tilde{k}(d_1, d_2) &= \Phi(d_1) R\ R^T \Phi(d_2)^T \\ &= \sum_t idf(t)^2\ tf(t, d_1)\ tf(t, d_2).\end{aligned}$$

L'évaluation d'un tel noyau implique tf et idf. C'est pour cette raison qu'il est référé en tant que représentation tf-idf. Cette représentation est en mesure de mettre en évidence les termes potentiellement discriminants. Aussi, elle affecte une pondération faible aux termes non pertinents. Cependant, elle est toujours

incapable de reconnaître le lien sémantique entre deux mots, voire deux documents. Cette relation sémantique peut être introduite dans le noyau sémantique par le biais d'une matrice de proximité P dont les entrées en dehors de la diagonale principale sont non nulles, $P_{ij} > 0$ quand le terme t_i est sémantiquement corrélé au terme t_j.

Étant donnée P, le noyau sémantique sera exprimé ainsi :

$$\bar{k}(d_1, d_2) = \Phi(d_1)P\ P^T\Phi(d_2)^T \tag{3.2}$$

Par conséquent, un document sera représenté par un vecteur moins creux $\Phi(d_1)P$. Il contient des éléments non nuls pour des termes ayant des relations sémantiques avec d'autres termes. Cette forme de noyau rejoint l'idée de base du noyau proposé dans Kandola *et al.* 2003 :

$$\begin{aligned}\bar{k}(d_1, d_2) &= \Phi(d_1)\ Q\ \Phi(d_2)^T \\ &= \sum_{i,j} \Phi(d_1)_i\ Q_{ij}\ \Phi(d_2)_j,\end{aligned}$$

où $Q = PP^T$ et l'entrée Q_{ij} encode la corrélation sémantique entre les termes t_i et t_j. Toutefois, la définition de Q requiert une contrainte additionnelle, Q doit être semi-définie positive pour que le noyau soit valide et par conséquent on peut forcément la décomposer en $Q = PP^T$. Ceci témoigne que le processus qui commence par la construction de la matrice P pour réaliser le noyau sémantique est, en général, plus simple.

Matrice de proximité avec des réseaux sémantiques

Une autre méthode de conception des matrices de proximité consiste à utiliser des ressources externes encodant, à priori, des connaissances sémantiques sur les mots. Siolas & d'Alche Buc 2000 ont construit la matrice de similarité terme-terme Q, où la similarité entre deux termes correspond au nombre d'ancêtres communs de leurs nœuds représentatifs dans l'arbre hiérarchique du réseau sémantique WordNet.

Cette nouvelle métrique peut être incorporée directement dans les noyaux de la méthode SVM ou utilisée directement dans l'algorithme des K-proches voisins. Les deux méthodes SVM et kNN (k Nearest Neighbors) sont testées et

comparées sur la base de données 20-newsgroups. Les résultats de comparaison montrent que SVM fournit la meilleure précision.

Dans la même optique, il est possible de choisir une proximité sémantique égale à l'inverse de la distance entre deux termes dans l'arbre. Autrement dit, la longueur du plus court chemin les reliant (Shawe-Taylor & Cristianini 2004).

Le modèle d'espace vectoriel généralisé

Le modèle d'espace vectoriel généralisé (GVSM, Generalized Vector Space Model) est une variante du VSM classique. Il a été proposé dans Wong *et al.* 1985 afin d'introduire une similarité sémantique entre les termes du dictionnaire. L'idée principale stipule que deux termes sont sémantiquement reliés s'ils apparaissent fréquemment ensemble dans les mêmes documents. Ainsi, deux documents peuvent être considérés comme similaires même s'ils ne partagent aucun terme, mais les termes qu'ils contiennent apparaissent ensemble dans d'autres documents.

Formellement, un document est représenté comme suit :

$$\tilde{\Phi}(d) = \Phi(d)D^T$$

où D est la matrice document-terme.

Il se peut que cette définition n'illustre pas clairement la similarité sémantique, mais avec la définition du noyau correspondant :

$$\begin{aligned}\tilde{k}(d_1, d_2) &= \left\langle \tilde{\Phi}(d_1), \tilde{\Phi}(d_2) \right\rangle \\ &= \Phi(d_1)D^T \; D\Phi(d_2)^T,\end{aligned}$$

l'entrée $(D^T D)_{ij}$ de la matrice terme-terme $D^T D$ est non nulle si et seulement s'il existe un document dans le corpus pour lequel les termes t_i et t_j apparaissent ensemble. Ceci peut être interprété par le fait que l'importance de la relation sémantique entre deux termes est déterminée par la somme des fréquences de leurs co-occurrences :

$$(D^T D)_{ij} = \sum_d tf(t_i, d)tf(t_j, d).$$

Dans le cas où le nombre de documents est inférieur aux nombre de termes, l'approche GVSM pourra être utilisée comme réduction de dimension.

Par ailleurs, la manière de pondération de l'approche GVSM affecte des fréquences de co-occurrences élevées pour les termes synonymes utilisées par un auteur pour éviter les répétitions, alors que les fréquences de co-occurrences seront faibles pour les termes sémantiquement proches dans le sens linguistique utilisés d'une façon indépendante dans les différents documents. Cela implique que GVSM peut détecter le premier cas de synonymie. Par contre, le deuxième cas et le problème de polysémie sont traités par l'approche de l'indexation sémantique latente.

Noyaux à sémantique latente

L'indexation sémantique latente (LSI, Latent Semantic Indexing) consiste à découvrir des motifs sémantiques latents dans un corpus de documents. La LSI utilise le même principe que la GVSM (information de co-occurrence des termes) mais avec une technique d'extraction plus perfectionnée. Intuitivement, la LSI définit un espace de concepts dans lequel les termes ayant une co-occurrence importante sont projetés (par une combinaison linéaire de ces termes) sur un même axe représentant un concept.

La LSI se base sur la technique de décomposition de la matrice terme-terme, D^T, en valeurs singulières (SVD, Singular Value Decomposition) :

$$D^T = U\Sigma V^T,$$

où U et V sont respectivement des matrices unitaires dont les colonnes sont les vecteurs propres des matrices $D^T D$ et DD^T. U est la matrice de passage de l'espace VSM à l'espace de concepts, elle relie les termes aux concepts. Alors que V est la matrice de passage de l'espace des termes à l'espace des concepts, elle relie les documents aux concepts.

La technique SVD considère les k colonnes des matrices des vecteurs propres (U et V) correspondant aux k valeurs propres les plus importantes. La technique LSI projette les documents dans *un espace co*ncerné par les k premières colonnes

de U (U_k) :

$$d \mapsto \Phi(d)U_k$$

Le noyau à sémantique latente (Cristianini *et al.* 2002) est alors défini :

$$\tilde{k}(d_1, d_2) = \Phi(d_1)U_k U_k^T \Phi(d_2)^T.$$

Il en découle que la matrice de proximité P est égale à U_k.

Cristianini *et al.* 2002 ont fourni les résultats expérimentaux démontrant que cette approche peut améliorer les performances des noyaux sur des données textuelles ou autre. Toutefois, la complexité de calcul de la décomposition en valeurs propres de la matrice noyau est un inconvénient majeur de la LSI.

3.4 Noyaux de séquences

Les noyaux pour le texte étudiés dans les sections précédentes reposent sur la représentation vectorielle qu'elle soit de mots ou de concepts. Dans la présente section, nous nous intéressons aux noyaux de séquences ou de mots qui reposent sur la représentation d'une entité textuelle en tant que séquence de symboles sur un alphabet (Section 3.1.3).

Avant d'aborder une collection de noyaux de séquences, nous présentons les noyaux de convolution, dont les noyaux de séquences font partie, ainsi que quelques définitions afférentes aux mots et aux séquences jugées indispensables pour la compréhension de la suite de ce chapitre.

3.4.1 Noyaux de convolution

Le principe de convolution (Haussler 1999) introduit une méthode pour la construction de noyaux sur des ensembles dont les éléments sont des structures discrètes comme les mots, les arbres et les graphes. Ces structures discrètes sont des objets récursifs pouvant être décomposés en sous-objets jusqu'à atteindre une unité atomique.

L'idée du noyau de convolution consiste à adopter une démarche récursive dans le calcul de similarité. Formellement, soit x une structure discrète appartenant à un ensemble $\mathbf{X}$ de même type. La structure x peut être décomposée en sous-objets $x_1, \ldots, x_D$ où x_d appartient à l'ensemble $\mathbf{X_d}$ pour $1 \leq d \leq D$ et D est un entier positif.

Nous pouvons représenter la relation « $\vec{x} = x_1, \ldots, x_D$ sont des parties de x » par la relation $R : X_1, \ldots, X_D \mapsto X$ avec $R(\vec{x}, x)$ vraie si et seulement si $x_1, \ldots, x_D$ sont des parties de x. Soit $R^{-1}(x) = \{x : R(\vec{x}, x)\}$ qui retourne pour x l'ensemble de toutes les décompositions possibles.

Supposons que $x, y \in \mathbf{X}$ et que $\vec{x} = x_1, \ldots, x_D$ et $\vec{y} = y_1, \ldots, y_D$ sont respectivement des décompositions de x et y. Nous supposons aussi que, pour chaque $1 \leq d \leq D$, nous disposons d'un noyau k_d sur X_d que nous pouvons utiliser pour mesurer la similarité $k_d(x_d, y_d)$ entre la partie x_d et la partie y_d. Alors nous définissons le noyau de convolution (R-noyau ou R-convolution) pour deux éléments x, y de $\mathbf{X}$ comme suit :

$$k(x, y) = \sum_{\vec{x} \in R^{-1}(x),\ \vec{y} \in R^{-1}(y)} \prod_{d=1}^{D} k_d(x_d, y_d). \tag{3.3}$$

Il est facile de montrer que si les k_d sont des noyaux valides, le noyau de convolution k est valide. Si k_d est valide, alors sa matrice Gram est semi-définie positive. Il en est de même pour le produit et la somme des matrices semi-définies positives.

Par ailleurs, les définitions récursives pour le calcul d'un noyau de convolution nécessite un temps de calcul important. Par conséquent le recours à des techniques de programmation itératives et de structures de données sophistiquées est nécessaire.

3.4.2 Mots et séquences

Soit Σ un alphabet d'un ensemble fini de symboles. Nous dénotons le nombre de symboles de Σ par $|\Sigma|$. Un mot $s = s_1 \ldots s_{|s|}$ est une séquence finie de symbole de Σ de longueur $|s|$, où s_i désigne le i^{e} élément de s. Un mot peut être éventuellement vide, il est représenté conventionnellement par le symbole ϵ ; il correspond à la suite de symboles vides. Le mot vide est de longueur 0.

Nous utilisons Σ^n pour représenter l'ensemble de tous les mots de longueur n et Σ^* pour dénoter l'ensemble de tous les mots que l'on peut construire sur l'alphabet Σ :

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n.$$

La notation $[s = t]$ est utilisée pour représenter la fonction booléenne qui renvoi :

$$\begin{cases} 1 & \text{si les mots } s \text{ et } t \text{ sont identiques ;} \\ 0 & \text{sinon.} \end{cases}$$

La concaténation de deux mots s et t, notée $s \cdot t$ ou simplement st est le mot obtenu en juxtaposant les symboles de t suite à ceux de s. Le mot s^n représente la concaténation de s avec lui même n fois.

Par ailleurs, étant donné quatre mots s, u, t et v définis sur un alphabet Σ, on dit que t est un sous-mot (facteur) de s si on peut écrire $s = utv$ (u et v peuvent être vides). Si $u = \epsilon$, on dit que t est un préfixe de s. Par contre, si $v = \epsilon$, t est dit suffixe de s. Le symbole ϵ est un sous-mot, préfixe et suffixe de tout mot. Le mot $s(i : j)$ dénote le sous-mot $s_i s_{i+1} \dots s_j$ de s. Les sous-mots de longueur k sont également appelés k-grams ou k-mers.

Le mot t est une sous-séquence du mot s s'il existe une séquence d'indices croissante $I = (i_1, \dots, i_{|t|})$ dans s, ($1 \leq i_1 < \dots < i_{|t|} \leq |s|$) tel que $t_j = s_{i_j}$, pour $j = 1, \dots, |t|$. Dans la littérature, on utilise $t = s(I)$ si t est une sous-séquence de s dans les positions données par I. Un mot vide est indexé par le tuple vide. La valeur absolue $|t|$ représente la longueur de la sous-séquence t qui est le nombre des indices de I ($|I|$). Cependant, $l(I) = i_{|t|} - i_1 + 1$ fait référence au nombre de symboles de s couverts par la sous-séquence t. Par convention, si I est le tuple vide (associé au mot vide) alors $l(I) = 0$.

À titre d'exemple, pour l'alphabet $\Sigma = \{a, b, \dots, z\}$, la longueur du mot $s =$ 'noyaux' est $|s| = 6$, les mots $s(1 : 3) =$ 'noy' et $s(4 : 6) =$ 'aux' sont, respectivement, un préfixe et un suffixe de s. Le préfixe $s(1 : 3)$, le suffixe $s(4 : 6)$ et $s(3 : 5) =$ 'yau' sont des sous-mots de s. Le mot $s(2, 4, 6) =$ 'oax' est une sous-séquence de longueur 3, alors que $l(2, 4, 6) = 5$.

3.4.3 Noyau p-spectre

Le noyau p-spectre ou p-gram (Leslie *et al.* 2002) est un noyau de séquences de caractères. Il utilise un espace de redescription indexé par tous les sous-mots de longueur p (p-spectres ou p-grams). La composante d'un p-spectre u représente le nombre d'occurrences de u dans le mots s :

$$\Phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, \ u \in \Sigma^p.$$

Le noyau p-spectre évalue le nombre de sous-mots de taille p que deux entités textuelles ont en commun. Ceci est réalisé en effectuant le produit scalaire dans cet espace (donc noyau valide) :

$$k_p(s,t) = \langle \Phi^p(s), \Phi^p(t) \rangle = \sum_{u \in \Sigma^p} \Phi_u^p(s) \Phi_u^p(t).$$

Pour évaluer le noyau p-spectre, on peut définir un noyau auxiliaire connu sous le nom k-suffixe (Shawe-Taylor & Cristianini 2004) :

$$k_k^S(s,t) = \begin{cases} 1 & \text{si } s = s_1 u \text{ et } t = t_1 u, \text{ pour } u \in \Sigma^p ; \\ 0 & \text{sinon.} \end{cases}$$

Par la suite, le noyau p-spectre peut être exprimé en fonction de la version k-suffixe :

$$k_p(s,t) = \sum_{i=1}^{|s|-p+1} \sum_{j=1}^{|t|-p+1} k_p^S(s(i : i+p-1), t(j : j+p-1)). \tag{3.4}$$

Il est clair que l'évaluation du noyau k-suffixe, $k_p^S(s,t)$, nécessite $O(p)$ comparaisons. Il en résulte de l'Équation (3.4) que l'évaluation complète du noyau p-spectre est $O(p\,|s|\,|t|)$. Il est à noter que dans la littérature, il existe des méthodes de calcul du noyau p-spectre plus efficaces qui reposent essentiellement sur des structures de données plus appropriées comme les tries.

Le noyau p-spectre est utilisé avec la méthode SVM dans une approche discriminative pour le problème de la classification de la protéine. Les résultats expérimentaux, sur la base de données SCOP (Structural Classification of Proteins) (Lo Conte *et al.* 2000), ont révélé que le noyau fonctionne remarquablement bien par rapport aux méthodes de l'état de l'art. Ceci qualifie les méthodes

à noyau à base de mots en conjonction avec la méthode SVM à fournir une alternative simple et efficace par rapport à d'autres méthodes de la classification de la protéine et de la détection d'homologie.

3.4.4 Noyau toutes sous-séquences

Le noyau toutes sous-séquences (Shawe-Taylor & Cristianini 2004) utilise un espace de redescription indexé par toutes les sous-séquences possibles. La composante d'une sous-séquence u représente le nombre d'occurrences de u comme sous-séquence dans le mot s :

$$\Phi_u(s) = |\{I : u = s(I)\}|, \ u \in \Sigma^*.$$

Le noyau associé est :

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle = \sum_{u \in \Sigma^*} \Phi_u(s)\Phi_u(t). \tag{3.5}$$

Le calcul explicite du produit scalaire dans cet espace devient rapidement coûteux. D'où la nécessité d'une autre méthode de nature récursive plus efficace que celle explicite. Pour des raisons d'illustration des calculs ultérieurs, essayons de montrer la contribution d'une composante u dans le calcul du produit scalaire :

$$\Phi_u(s)\Phi_u(t) = \sum_{I:u=s(I)} 1 \sum_{J:u=t(J)} 1 = \sum_{(I,J):u=s(I)=t(J)} 1 \tag{3.6}$$

En utilisant le résultat de l'Équation (3.6), Le noyau de l'$\acute{E}$quation (3.5) peut être reformulé ainsi :

$$k(s,t) = \sum_{u \in \Sigma^*} \sum_{(I,J):u=s(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1. \tag{3.7}$$

Maintenant, on peut définir la récursion qui se base sur le raisonnement suivant : si l'on ajoute un symbole a au mot s, le nombre de sous-séquences communes peut être donné par la somme finale de l'Équation (3.7) comme suit :

$$\sum_{(I,J):sa(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1 + \sum_{u:t=uav} \sum_{(I,J):s(I)=u(J)} 1. \tag{3.8}$$

La première somme de l'Équation (3.8) fait référence aux sous-séquences de s, alors que la somme finale fait référence aux sous-séquences dont le symbole finale est a, d'où la version récursive du noyau toutes sous-séquences :

$$\begin{aligned} k(s, \epsilon) &= 1 \\ k(sa, t) &= k(s, t) + \sum_{j:t_j=a} k(s, t(1 : j-1)). \end{aligned} \tag{3.9}$$

La propriété de la symétrie des noyaux permet une récursion analogue :

$$\begin{aligned} k(\epsilon, t) &= 1 \\ k(s, ta) &= k(s, t) + \sum_{j:s_j=a} k(s(1 : j-1), t). \end{aligned}$$

Une évaluation récursive de ce noyau est très coûteuse, d'où le recours à la technique de programmation dynamique qui consiste à sauvegarder les résultats intermédiaires dans une table de programmation dynamique (Table 3.1) dont chaque ligne correspond à un symbole du mot s et chaque colonne correspond à un symbole du mot t :

TABLE 3.1 – Table de programmation dynamique pour le calcul d'un noyau.

DP	ϵ	t_1	t_2	$\dots$	t_m
ϵ	1	1	1	$\dots$	1
s_1	1	k_{11}	k_{12}	$\dots$	k_{1m}
s_2	1	k_{21}	k_{22}	$\dots$	k_{2m}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_n	1	k_{n1}	k_{n2}	$\dots$	k_{nm}

La première ligne et la première colonne expriment le cas de base de la récurrence. Les entrées $k_{ij} = k(s(1 : i), t(1 : j))$ donnent les évaluations des noyaux pour les préfixes des deux mots s et t. La formule de récurrence (3.9) nous permet de calculer l'entrée (i, j) comme la somme de l'entrée $(i-1, j)$ avec toutes les entrées $(i-1, k-1)$ avec $1 \leq k < j$ dont $t_k = s_i$.

Le nombre d'opérations nécessaires pour évaluer une entrée (i, j) est $O(j)$, car on doit parcourir tous les symboles de t jusqu'à la position j. Ainsi, pour renseigner complètement la table dynamique, nous aurons besoin de $O(|s|\,|t|^2)$ opérations, une complexité qui nécessite toujours d'être améliorée.

Amélioration

Il est possible d'améliorer la complexité d'évaluation du noyau toutes sous-séquences en observant que lors du remplissage de la ligne i, la somme :

$$\sum_{k \leq j : t_k = s_i} k(s(1:i-1), t(1:k-1)),$$

requise lorsque nous atteignons la position j aurait pu être pré-calculée dans un tableau P (Algorithme 1).

Algorithm 1: Calcul et stockage des sommes intermédiaires dans un tableau P.

```
1 m ← length(t)
2 last ← 0
3 P[0] ← 0
4 for k = 1 : m do
5 |   P[k] ← P[last]
6 |   if t_k = s_i then
7 |   |   P[k] ← P[last] + DP[i − 1, k − 1]
8 |   |   last ← k
```

Le fragment de code de l'Algorithme 2 illustre comment renseigner la ligne suivante dans la table dynamique en utilisant le tableau P.

Algorithm 2: Remplissage de la ligne suivante dans la table dynamique.

```
1 for k = 1 : m do
2 |   DP[i, k] ← DP[i − 1, k] + P[k]
```

Pour évaluer la complexité de l'algorithme de l'évaluation complète du noyau toutes sous-séquences (Algorithme 3), nous pouvons considérer deux blocs en séquentiel, à savoir la boucle de 3 à 4 dont le temps d'exécution est $O(|t|)$ et la boucle de 5 à 14 qui comporte deux blocs en séquentiel. Les deux boucles internes de 8 à 12 et de 13 à 14 s'exécutent dans un temps $O(|s|)$, d'où une complexité totale de l'algorithme en $O(|s|\,|t|)$.

En plus, il est à noter que cette manière d'évaluation du noyau toutes sous-séquences permet de résoudre un problème plus général qui consiste à calculer les noyaux $k(s(1:i), t(1:j))$ pour tout $1 \leq i \leq |s|$ et pour tout $1 \leq j \leq |t|$. Autrement dit, à la fin de l'évaluation du noyau toutes sous-séquences, on peut

Algorithm 3: Évaluation du noyau toutes sous-séquences.

Input: Les mots s et t.
Output: Valeur du noyau $k(s,t) = DP\,[n,m]$.

$n \leftarrow length(s)$
$m \leftarrow length(t)$
for $j = 1 : m$ **do**
    $DP[0,j] \leftarrow 1$
/* Traitement des lignes */
for $i = 1 : n$ **do**
    /* pré-calcul et remplissage du tableau P */
    $last \leftarrow 0$
    $P\,[0] \leftarrow 0$
    for $k = 1 : m$ **do**
        $P\,[k] \leftarrow P\,[last]$
        if $t_k = s_i$ **then**
            $P\,[k] \leftarrow P\,[last] + DP[i-1,k-1]$
            $last \leftarrow k$
    /* renseignement de la ligne suivante */
    for $k = 1 : m$ **do**
        $DP[i,k] \leftarrow DP[i-1,k] + P[k]$

consulter la table de la programmation dynamique pour l'évaluation d'un noyau entre n'importe quel préfixe de s avec n'importe quel préfixe de t.

3.4.5 Noyau sous-séquences de longueur fixe

Nous avons déjà constaté que l'espace de redescription du noyau toutes sous-séquences est de très haute dimension. Une adaptation possible consiste à réduire cette dimension en imposant une longueur fixe des sous-séquences. En effet, le noyau sous-séquences de longueur fixe (Shawe-Taylor & Cristianini 2004) est un compromis entre le noyau p-spectre et le noyau toutes sous-séquences.

Pour ce noyau, l'espace de redescription associé est indexé par toutes les sous-séquences possibles de longueur p (Σ^p). La composante d'une sous-séquence u de longueur p représente le nombre d'occurrences de u dans le mots s :

$$\Phi_u^p(s) = |\{I : u = s(I)\}|\,,\ u \in \Sigma^p.$$

Le noyau correspondant évalue le nombre de sous-séquences de taille p que deux entités textuelles ont en commun. Ceci est réalisé en effectuant le produit scalaire dans cet espace (donc noyau valide) :

$$k_p(s,t) = \langle \Phi^p(s), \Phi^p(t) \rangle = \sum_{u \in \Sigma^p} \Phi^p_u(s) \Phi^p_u(t).$$

De même que le noyau toutes sous-séquences, on peut exprimer ce noyau comme dans l'Équation (3.7), mais cette fois-ci en tenant compte des séquences d'indices I_p de longueur p :

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle = \sum_{u \in \Sigma^p} \sum_{(I,J):u=s(I)=t(J)} 1 = \sum_{(I,J) \in I_p \times I_p : s(I)=t(J)} 1.$$

Avec le même raisonnement récursif exprimé par l'Équation (3.8), nous aurons :

$$\sum_{(I,J) \in I_p \times I_p : sa(I)=t(J)} 1 = \sum_{(I,J) \in I_p \times I_p : s(I)=t(J)} 1 + \sum_{u:t=uav} \sum_{(I,J) \in I_{p-1} \times I_{p-1} : s(I)=u(J)} 1. \quad (3.10)$$

Par conséquent, la version récursive du noyau toutes sous-séquences de longueur fixe p est donnée par :

$$\begin{aligned} k_0(s,t) &= 1, \\ k_p(s,\epsilon) &= 0, \text{ pour } p > 0, \\ k_p(sa,t) &= k_p(s,t) + \sum_{j:t_j=a} k_{p-1}(s, t(1:j-1)). \end{aligned}$$

L'évaluation récursive de ce noyau est très coûteuse. Comme pour le noyau toutes sous-séquences, la technique de la programmation dynamique peut être proposée comme une alternative pour améliorer le temps d'exécution d'une telle évaluation. Cependant, une autre contrainte se pose. Au delà de la récursion sur les préfixes de mots, une autre récursion sur la longueur des sous-séquences s'ajoute. D'où, la nécessité du rajout d'une table de programmation dynamique, DP_{prec}, qui sert à stocker l'évaluation du noyau de l'itération précédente. L'Algorithme 4 propose une solution itérative à base de programmation dynamique.

Une analyse de l'Algorithme 4 montre que la complexité du noyau toutes sous-séquences de longueur p est $O(p|s||t|)$.

Algorithm 4: Évaluation du noyau sous-séquences de longueur p.

Input: Les mots s et t et la longueur p des sous-séquences.
Output: Valeur du noyau $k_p(s,t) = DP\,[n,m]$.

```
1  n ← length(s)
2  m ← length(t)
3  for j = 0 : m do
4      DP[0, j] ← 1
5  for i = 0 : n do
6      DP[i, 0] ← 1
7  for l = 1 : p do
8      DP_prec ← DP
9      for j = 1 : m do
10         DP[0, j] ← 1
11     for i = 1 : n-p+1 do
12         last ← 0
13         for k = 1 : m do
14             P [k] ← P [last]
15             if t_k = s_i then
16                 P [k] ← P [last] + DP_prec[i − 1, k − 1]
17                 last ← k
           /* renseignement de la ligne suivante          */
18         for k = 1 : m do
19             DP[i, k] ← DP[i − 1, k] + P[k]
```

3.4.6 Noyau sous-séquences de mots

L'un des inconvénients des noyaux traitant les sous-séquences est qu'ils les traitent de la même manière, sans tenir compte des distances séparant les éléments non contiguës (trous). Le noyau sous-séquences de mots (SSK, String Subsequence Kernel) (Lodhi *et al.* 2002) adopte une nouvelle méthode de pondération qui reflète le degré de contiguïté de la sous-séquence dans le mot.

Pour mesurer la distance des éléments non contiguës des sous-séquences, un facteur de pénalisation $\lambda \in]0,1]$ est introduit. Ainsi, l'espace de redecsription associé à ce noyau est indexé par toutes les sous-séquences de longueur p dont la composante est donnée par :

$$\phi_u^p(s) = \sum_{I:u=s(I)} \lambda^{l(I)},\ u \in \Sigma^p.$$

Le noyau associé peut être écrit comme suit :

$$\begin{aligned} K_p(s,t) &= \langle \phi^p(s), \phi^p(t) \rangle \\ &= \sum_{u \in \Sigma^p} \phi^p_u(s).\phi^p_u(t) \\ &= \sum_{u \in \Sigma^p} \sum_{I:u=s(I)} \sum_{J:u=t(J)} \lambda^{l(I)+l(J)}. \end{aligned} \tag{3.11}$$

Pour le noyau SSK, nous allons le traiter avec un soin particulier, car c'est bien le sujet essentiel de notre contribution. Commençons par une illustration de l'idée de ce noyau avec un exemple très répandu dans la littérature. Considérons les mots bar, bat, car et cat, pour des sous-séquences de longueur $p = 2$. La Table 3.2 montre la projection de ces mots dans l'espace de redescription.

TABLE 3.2 – Projection des mots bar, bat, car et cat dans un espace de redescription pour des sous-séquences de longueur $p = 2$.

ϕ^2_u	ar	at	ba	br	bt	ca	cr	ct
bar	λ^2	0	λ^2	λ^3	0	0	0	0
bat	0	λ^2	λ^2	0	λ^3	0	0	0
car	λ^2	0	0	0	0	λ^2	λ^3	0
cat	0	λ^2	0	0	0	λ^2	0	λ^3

La valeur du noyau entre bar et bat est $K_2(bar, bat) = \lambda^4$, tandis que, la version normalisée est obtenue par :

$$\widehat{K}_2(bar, bat) = K_2(bar, bat)/\sqrt{K_2(bar, bar).K_2(bat, bat)} = \lambda^4/(2\lambda^4+\lambda^6) = 1/(2+\lambda^2).$$

Une implémentation directe de ce noyau mène à une complexité temporelle et spatiale de $O(|\Sigma^p|)$, du moment que $|\Sigma^p|$ est la dimension de l'espace de redescription. Dans une perspective d'amélioration d'une telle complexité, on peut faire recours à la récursivité. Comme pour les noyaux précédents, ceci passe par la définition d'un noyau de suffixe dont la projection est définie comme suit :

$$\phi^{p,S}_u(s) = \sum_{I \in I_p^{|s|}:u=s(I)} \lambda^{l(I)},\ u \in \Sigma^p,$$

où I^k_p dénote l'ensemble des p-tuples des indices I avec $i_p = k$. Autrement dit, nous considérons seulement les sous-séquences de longueur p dont le dernier

symbole est identique au dernier symbole du mot s. Le noyau associé peut être défini comme suit :

$$\begin{aligned} K_p^S(s,t) &= \langle \phi^{p,S}(s), \phi^{p,S}(t) \rangle \\ &= \sum_{u \in \Sigma^p} \phi_u^{p,S}(s).\phi_u^{p,S}(t). \end{aligned}$$

Nous reprenons l'exemple précédent, pour illustrer cette astuce de comptage du noyau suffixe (Table 3.3). Par exemple $K_2^S(bar, bat) = 0$ et $K_2^S(bat, cat) = \lambda^4$.

TABLE 3.3 – Version suffixe de la projection des mots bar, bat, car et cat dans l'espace de redescription pour les sous-séquences de longueur $p = 2$.

$\phi_u^{2,S}$	ar	at	ba	br	bt	ca	cr	ct
bar	λ^2	0	0	λ^3	0	0	0	0
bat	0	λ^2	0	0	λ^3	0	0	0
car	λ^2	0	0	0	0	0	λ^3	0
cat	0	λ^2	0	0	0	0	0	λ^3

Ceci dit, le noyau SSK peut être exprimé en fonction de sa version suffixe comme suit :

$$K_p(s,t) = \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} K_p^S(s(1:i), t(1:j)), \tag{3.12}$$

avec $K_1^S(s,t) = [s_{|s|} = t_{|t|}]\ \lambda^2$.

Le calcul de la similarité entre deux mots (sa et tb) est conditionné par leurs symboles finaux. Dans ce cas, quand $a = b$, nous devons sommer les noyaux de tous les préfixes de s et t. Ainsi une récursion peut être conçue :

$$K_p^S(sa, tb) = [a = b] \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} \lambda^{2+|s|-i+|t|-j} K_{p-1}^S(s(1:i), t(1:j)). \tag{3.13}$$

L'implémentation naïve de cette récursion conduit à une complexité $O(p(|s|^2|t|^2))$ qui reste toujours non satisfaisante, d'où la nécessité de considérer d'autres issues conduisant à des implémentations plus efficaces.

Implémentations efficaces

Dans notre contexte, nous présentons trois approches qui calculent efficacement le noyau SSK, à savoir, les approches de la programmation dynamique, la programmation dynamique éparse et celle à base de trie. Pour décrire ces approches, nous utilisons les mots $s =$ gatta et $t =$ cata comme un exemple qu'on va dérouler tout au long de notre description.

Approche de la programmation dynamique

Le point de départ de l'approche de la programmation dynamique (Lodhi *et al.* 2002) est la récursion suffixe donnée par l'Équation (3.13). À partir de cette équation, on peut considérer une table de programmation dynamique DP_p pour emmagasiner la double somme :

$$DP_p(k,l) \quad = \quad \sum_{i=1}^{k} \sum_{j=1}^{l} \lambda^{k-i+l-j} \, K_{p-1}^{S}(s(1:i), t(1:j)). \tag{3.14}$$

Suite aux Équations (3.13 et 3.14), il est facile de voir que :

$$K_p^S(sa, tb) = [a = b] \, \lambda^2 \, DP_p(|s|, |t|)). \tag{3.15}$$

Cependant, l'évaluation ordinaire de la table DP_p pour chaque entrée (k, l) serait inefficace. Alors, nous pouvons concevoir une version récursive de l'Équation (3.14) avec un dispositif de calcul simple :

$$\begin{aligned} DP_p(k,l) = K_{p-1}^{S}(s(1:k), t(1:l)) + \lambda DP_p(k-1,l) + \\ \lambda DP_p(k,l-1) - \lambda^2 DP_p(k-1,l-1). \end{aligned} \tag{3.16}$$

La traduction algorithmique des Équations (3.15 et 3.16) se manifeste par (Algorithme 5) dont l'analyse conduit à une complexité de calcul SSK en $O(p\,|s||t|)$. À titre d'illustration, la Table 3.4 montre le calcul des tables de la programmation dynamique pour notre exemple et pour $p = 1, 2$. Les valeurs du noyau SSK correspondant sont données par :

$$\begin{aligned} K_1(s,t) &= 6\lambda^2, \\ K_2(s,t) &= 2\lambda^4 + 2\lambda^5 + \lambda^7. \end{aligned}$$

Algorithm 5: Évaluation du noyau SSK par la programmation dynamique.

Input: Les mots s et t, La longueur p des sous-séquences et le facteur de pénalisation λ.
Output: Valeurs du noyau $K_q(s,t) = K(q) : q = 1, \ldots, p$.

$m \leftarrow length(s)$
$n \leftarrow length(t)$
$K(1:p) \leftarrow 0$
/* Calcul de $K_1(s,t)$ */
for *i = 1 :m* **do**
  for *j = 1 :n* **do**
    if *s[i] = t[j]* **then**
      $KPS[i,j] \leftarrow \lambda^2$
      $K[1] \leftarrow K[1] + KPS[i,j]$

/* Calcul de $K_q(s,t) : q = 2, \ldots, p$ */
for *q = 2 :p* **do**
  for *i = 1 :m* **do**
    for *j = 1 :n* **do**
      $DP[i,j] \leftarrow$
      $KPS[i,j] + \lambda DP[i-1,j] + \lambda DP[i,j-1] - \lambda^2 DP[i-1,j-1]$
      if *s[i] = t[j]* **then**
        $KPS[i,j] \leftarrow \lambda^2 DP[i-1,j-1]$
        $K[q] \leftarrow K[q] + KPS[i,j]$

Approche de la programmation dynamique éparse

L'approche de la programmation dynamique éparse est construite sur le fait que, dans de nombreux cas, la plus part des entrées de la matrice de la programmation dynamique DP sont nuls et ne contribuent pas au résultat. Rousu & Shawe-Taylor 2005 ont proposé une solution utilisant la technique de la programmation dynamique éparse, pour éviter les calculs inutiles.

Pour ce faire, deux structures de données ont été proposées. La première est un arbre de somme d'intervalles (Range sum tree), un arbre binaire qui remplace la table de la programmation dynamique DP. Il est utilisé pour renvoyer la somme de n valeurs dans un intervalle en un temps $O(\log n)$. La seconde est un ensemble de listes de correspondances à la place de la matrice des suffixes.

$$L_q(i) = \{(j_1, \overline{K_p^S}(s(1:i), t(1:j_1)), (j_2, \overline{K_p^S}(s(1:i), t(1:j_2)), ...\},$$

TABLE 3.4 – Les tables de suffixe et la table de la programmation dynamique pour calculer le noyau SSK pour $p = 1,\ 2$.

KPS_1	g	a	t	t	a
c					
a		λ^2			λ^2
t			λ^2	λ^2	
a		λ^2			λ^2

DP_2	g	a	t	t	a
c	0	0	0	0	
a	0	λ^2	λ^3	λ^4	
t	0	λ^3	$\lambda^2 + \lambda^4$	$\lambda^2 + \lambda^3 + \lambda^5$	
a					

KPS_2	g	a	t	t	a
c					
a					
t		λ^4		λ^5	
a					$\lambda^4 + \lambda^5 + \lambda^7$

où $\overline{K_p^S}(s(1:i), t(1:j)) = \lambda^{m-i+n-j} K_p^S(s(1:i), t(1:j))$. Cette pondération de l'écart fictif ($\lambda^{m-i+n-j}$) permet de régler le problème de la mise en échelle des valeurs du noyau lors de la progression des calculs. Par conséquent la récursion de l'Équation (3.13) devient :

$$\overline{K_p^S}(sa, tb) = [a = b]\lambda^2 \sum_{i \le |s|} \sum_{j \le |t|} \overline{K_{p-1}^S}(s(1:i), t(1:j)). \tag{3.17}$$

Et la table de la programmation dynamique de l'Équation (3.14) peut être exprimée comme suit :

$$\overline{DP_p}(k, l) = \sum_{i \le k} \sum_{j \le l} \overline{K_{p-1}^S}(s(1:i), t(1:j)). \tag{3.18}$$

Par la suite, les auteurs élaborent une version récursive de l'Équation (3.18) :

$$\overline{DP_p}(k, l) = \overline{DP_p}(k-1, l) + \sum_{j \le l} \overline{K_{p-1}^S}(s(1:k), t(1:j)). \tag{3.19}$$

Ceci peut être interprété comme une réduction de l'évaluation d'une requête

d'intervalle orthogonale (orthogonal range query) (3.18) à une évaluation d'une requête d'intervalle simple autant de fois que de nombre de lignes de la matrice des suffixes K_p^S.

Pour évaluer efficacement la requête de somme d'intervalles, les auteurs utilisent un arbre de somme d'intervalles pour emmagasiner un ensemble $S = \{(j, v_j)\} \subset \{1, \ldots, n\} \times \mathbb{R}$ de paires clé-valeur. Un arbre de somme d'intervalles est un arbre binaire de hauteur $h = \lceil \log n \rceil$ où chaque nœud dans une profondeur $d = 0, 1, \ldots, h-1$ contient une clé j avec la somme des valeurs associée à un sous intervalle $[j - 2^{h-d} + 1, j]$. La racine est étiquetée par 2^h, le fils gauche d'un nœud j est $j - j/2$ et le fils droit s'il existe est $j + j/2$. Les clés impaires étiquettent les feuilles de l'arbre.

Pour évaluer la somme d'intervalle des valeurs dans un intervalle $[1, j]$, il suffit de parcourir le chemin allant du nœud j vers la racine et faire les sommes sur les sous-arbres gauches comme suit :

$$\text{Rangesum}([1, \text{j}]) \quad = \quad v_j + \sum_{h \in \text{Ancestors(j)} | h < j} v_h.$$

En plus, pour mettre à jour la valeur d'un nœud j, nous aurons besoin de mettre à jour toutes les valeurs de ses parents ($h \in \text{Ancestors(j)} | h > j$). Ces deux opérations sont effectuées dans un temps $O(\log n)$ car dans le pire des cas, on parcourt un chemin dont la longueur est la hauteur de l'arbre.

Pour l'algorithme de la programmation dynamique éparse (Algorithme 6), l'arbre de somme d'intervalles est utilisé d'une façon incrémentale lors du calcul de l'Équation (3.19). Alors que lors du traitement de la liste de correspondance $L_p(k)$, l'arbre contiendra les valeurs v_j satisfaisant $\sum_{i=1}^{k} \overline{K_{p-1}^S}(s(1:i), t(1:j)), 1 \leq j \leq l$. Par conséquent, l'évaluation de l'Équation (3.19) est exécutée en impliquant une requête d'intervalle unidimensionnelle :

$$\begin{aligned} \text{Rangesum}([1, \text{j}]) \quad &= \quad \sum_{j=1}^{l} v_j \\ &= \quad \sum_{i=1}^{k} \sum_{j=1}^{l} \overline{K_{p-1}^S}(s(1:i), t(1:j)) \\ &= \quad \overline{DP_p}(k, l). \end{aligned}$$

Concernant le coût de calcul de cette approche, l'ensemble des listes de correspondance est crée dans un temps et espace de $O(m+n+|\Sigma|+|L_1|)$, tandis que le temps de calcul du noyau est $O(p|L_1|\log n)$, sachant que $|L_1| \geq |L_2| \geq \ldots \geq |L_p|$.

Pour illustrer le mécanisme de l'algorithme de la programmation dynamique éparse, la Figure 3.1 représente l'état de l'arbre de somme d'intervalles durant le calcul de $K_2^S(s,t)$. Initialement l'ensemble des listes de correspondance est crée comme suit :

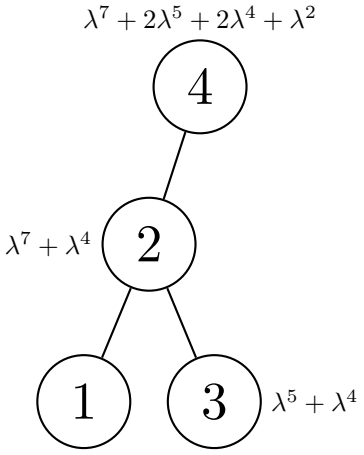


FIGURE 3.1 – L'état de l'arbre de somme d'intervalles durant le calcul de $K_2^S(s,t)$.

$$
\begin{aligned}
L_1(1) &= () \\
L_1(2) &= ((2,\lambda^7),(4,\lambda^5)) \\
L_1(3) &= ((3,\lambda^5)) \\
L_1(4) &= ((3,\lambda^4)) \\
L_1(5) &= ((2,\lambda^4),(4,\lambda^2)).
\end{aligned}
$$

Pendant le maintien de l'arbre de somme d'intervalles, l'algorithme met à jour l'ensemble des listes de correspondance tel que présenté ci-dessous :

$$
\begin{aligned}
L_2(1) &= () \\
L_2(2) &= () \\
L_2(3) &= ((3,\lambda^7)) \\
L_2(4) &= ((3,\lambda^7)) \\
L_2(5) &= ((4,\lambda^7+\lambda^5+\lambda^4)).
\end{aligned}
$$

Enfin, la somme des valeurs de la liste de correspondance mise à jour après l'écartement de la pondération fictive donne la valeur du noyau $K_2(s,t)$:

$$\begin{aligned} K_2(s,t) &= \lambda^7 \cdot \lambda^{-9+3+3} + \lambda^7 \cdot \lambda^{-9+4+3} + (\lambda^7 + \lambda^5 + \lambda^4) \cdot \lambda^{-9+5+4} \\ &= \lambda^7 + 2\lambda^5 + 2\lambda^4. \end{aligned}$$

Algorithm 6: Évaluation du noyau SSK par l'approche de la programmation dynamique éparse.

Input: Les mots s et t, la longueur p des sous-séquences et le facteur de pénalisation λ.
Output: Kernel value $K_p(s,t) = K$.

```
1  m ← length(s)
2  n ← length(t)
3  Création de l'ensemble des listes de correspondance L_1
4  for q = 2 :p do
5      Rangesum(1 : n) ← 0 (Initialisation de l'arbre de somme d'intervalles)
6      for i = 1 :m do
7          foreach (j_h, v_h) ∈ L_{q−1}(i) do
8              S ← Rangesum([1, j_h − 1])
9              if S > 0 then
10                 appendlist(L_q(i), (j_h, S))
           /* Mise à jour de l'arbre de somme d'intervalles
              */
11         foreach (j_h, v_h) ∈ L_{q−1}(i) do
12             update(Rangesum, (j_h, v_h))
   /* Calcul de la valeur du noyau pour le niveau final
      */
13 K ← 0
14 for i = 1 :m do
15     foreach (j_h, v_h) ∈ L_p(i) do
16         K ← K + v_h λ^{−m−n+i+j_h}
```

Approche à base de trie

Cette approche se base sur des arbres de recherche appelés *tries*, introduits par E. Fredkin en 1960. L'idée principale de l'approche à base de trie est que

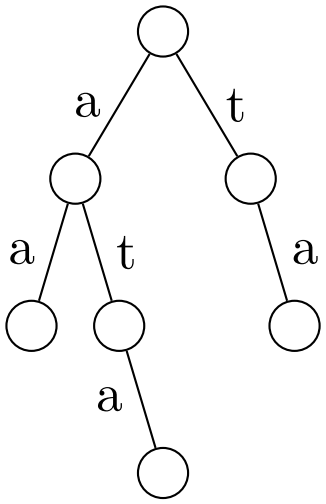


FIGURE 3.2 – La structure de données trie pour l'exemple de déroulement $s = gatta$, $t = cata$.

les feuilles de l'arbre jouent le rôle de l'espace de redescription indexé par l'ensemble Σ^p des mots de longueur p sur un alphabet Σ.

Dans la littérature, il existe des variantes des noyaux SSK à base de trie. Par exemple, le noyau de mots (p, m)-mismatch (Leslie *et al.* 2003) et le SSK restreint (Shawe-Taylor & Cristianini 2004). Dans la présente section, nous essayons de décrire une variante présentée dans Rousu & Shawe-Taylor 2005 qui diffère légèrement de celles citées ci-dessus (Leslie *et al.* 2003; Shawe-Taylor & Cristianini 2004).

TABLE 3.5 – Les indices actives pour toutes les sous-séquences de l'exemple de déroulement pour $p = 1,\ 2,\ 3$ avec le nombre de trous de 0 à 3.

g	$A_s('a', g)$	$A_t('a', g)$	$A_s('t', g)$	$A_t('t', g)$	$A_s('aa', g)$	$A_t('aa', g)$
0	$\{2, 5\}$	$\{2, 4\}$	$\{3, 4\}$	$\{3\}$		
1						$\{4\}$
2					$\{5\}$	
3						
g	$A_s('at', g)$	$A_t('at', g)$	$A_s('ta', g)$	$A_t('ta', g)$	$A_s('ata', g)$	$A_t('ata', g)$
0	$\{3\}$	$\{3\}$	$\{5\}$	$\{4\}$		$\{4\}$
1	$\{4\}$		$\{5\}$		$\{5, 5\}$	
2						
3						

La Figure 3.2 illustre la structure de données trie pour notre exemple de déroulement. Chaque noeud dans le trie correspond à une co-occurrence entre les mots. L'algorithme maintient pour toutes les correspondances $u = s(I) = u_1 \cdots u_q$, $I = i_1 \cdots i_q$ une liste de correspondance actives $A_s(u, g)$ telle que présentée dans la Table 3.5 qui enregistre le dernier indice i_q où $g = l(I) - |I|$ est le nombre de « trous » dans la correspondance.

Notons que dans la même liste on est en mesure d'enregistrer plusieurs occurrences avec différents « trous ». Les listes actives pour les correspondances longues $uc, (c \in \Sigma)$, peuvent être construites d'une façon incrémentale en étendant la liste active associée à u. De même, l'algorithme est appliqué au mot t. Le processus se poursuivra jusqu'à atteindre la profondeur p. Pour des raisons d'efficacité, le nombre de « trous » est restreint à un entier donné g_{max}, alors le calcul est approximatif.

Le noyau est évalué comme suit :

$$K_p(s,t) = \sum_{u \in \Sigma^p} \phi_u^p(s)\phi_u^p(t) = \sum_{g_s, g_t} \lambda^{g_s+p}|L_s(u,g_s)| \cdot \lambda^{g_t+p}|L_t(u,g_t)|.$$

Etant donné qu'il existe $\binom{p+g_{max}}{g_{max}}$ combinaisons possibles pour assigner p symboles et g_{max} « trous » dans une fenêtre de longueur $p + g_{max}$, la complexité temporelle de l'algorithme dans le pire des cas est $O(\binom{p+g_{max}}{g_{max}}\ (|s| + |t|))$. La valeur du noyau SSK pour notre exemple de déroulement et pour $p = 1$ est :

$$\begin{aligned} K_1(s,t) &= \lambda^{0+1}|A_s(\text{'a'},0)| \cdot \lambda^{0+1}|A_t(\text{'a'},0)| + \lambda^{0+1}|A_s(\text{'t'},0)| \cdot \lambda^{0+1}|A_t(\text{'t'},0)| \\ &= 4 \cdot \lambda^2 + 2 \cdot \lambda^2 = 6 \cdot \lambda^2. \end{aligned}$$

Des calculs similaires sont effectués pour évaluer K_2 et K_3 :

$$\begin{aligned} K_2(s,t) &= (1 \cdot \lambda^{2+2}) \cdot (1 \cdot \lambda^{1+2}) + (1 \cdot \lambda^{0+2} + 1 \cdot \lambda^{1+2}) \cdot (1 \cdot \lambda^{0+2}) \\ &\quad + (1 \cdot \lambda^{0+2} + 1 \cdot \lambda^{1+2}) \cdot (1 \cdot \lambda^{0+2}) \\ &= \lambda^7 + 2 \cdot \lambda^5 + 2 \cdot \lambda^4. \end{aligned}$$

et

$$K_3(s,t) = (2 \cdot \lambda^{1+3}) \cdot (1 \cdot \lambda^{0+3}) = 2 \cdot \lambda^7.$$

Applications du noyau sous-séquences de mots

Les noyaux sous-séquences de mots ont été déployés avec succès dans de nombreuses applications, telle que la classification, l'analyse sémantique, l'extraction de l'information relationnelle et la détection de logiciels malveillants. Sans être exhaustif, nous essayons de passer en revue certains travaux connexes inhérents aux applications SSK.

Classification de texte

Le noyau sous-séquences de mots (Lodhi *et al.* 2002) a été proposé comme alternative aux approches qui considèrent des descripteurs à base de mots, généralement, utilisées par les applications de classification de texte conjointement avec la méthode SVM (Joachims 1998).

Les auteurs analysent empiriquement leur approche sur un sous-ensemble de l'ensemble de données Reuters. Ils comparent l'approche proposée aux techniques de représentation de texte à base de sac de mot et de n-gram. Les expérimentations indiquent que l'approche offre des performances de l'état de l'art sur des jeux de données de taille modeste. En raison de la complexité du temps $O(p|s||t|)$, l'application du SSK sur des ensembles de données de taille importante est trop coûteuse. Pour surmonter ce problème, les auteurs ont proposé une approximation SSK qui permet un calcul rapide des matrices des noyaux.

Plus tard, Cancedda *et al.* 2003 a proposé d'utiliser le SSK avec une séquence de mots plutôt que de caractères. Cette transition augmente le nombre de symboles (mots) à considérer. Par conséquent, le nombre de dimensions de l'espace de redescription augmente. Cependant, la longueur moyenne du document diminue. Alors que le calcul de SSK dépend de la longueur de mot, une telle transition conduit à une amélioration du calcul de l'efficacité du noyau de séquences de mots.

Pour les besoins de leurs expérimentations, les auteurs ont utilisé l'ensemble de données standard pour les systèmes de classification de texte : Reuters-21578. Ils ont utilisé la méthode SVM conjointement avec divers noyaux sac à mots. Les résultats des expérimentations révèlent que les performances du noyau de séquence de mots sont comparables aux résultats de l'état de l'art.

Récemment, Nehar *et al.* 2014 ont utilisé des noyaux de sous-séquences pour la classification de texte arabe (ATC, Arabic Text Classification), y compris le noyau SSK. Ils ont mis en place un système qui prend en charge les aspects de l'ordre et les co-occurrences de mots dans un texte. Ils ont commencé par transformer les documents de la granularité caractère à la granularité mot. Par la suite, les documents sont représentés par des transducteurs, une telle représentation conduit à une évaluation particulière des noyaux sous-séquences, connue sous le nom de noyau rationnel (Cortes *et al.* 2004).

Les auteurs analysent empiriquement le système développé sur le corpus SPA (Saudi Press Agency) (Althubaity *et al.* 2008). Pour le noyau SSK, ils ont limité le nombre de trous à 5. Les résultats obtenus montrent que l'utilisation du SSK n'améliore pas les performances des systèmes ATC.

Classification des séquences protéiques

Un problème important dans la classification des séquences biologiques est l'annotation d'une nouvelle séquence protéique avec des propriétés structurelles et fonctionnelles. Peut-être que la conception de méthodes capables de regrouper toutes les protéines qui partagent les mêmes fonctions en familles permet de classer les protéines nouvellement découvertes sans connaissance préalable.

Dans cette perspective, Zaki *et al.* 2005 ont proposé une approche à base de noyau de mots dans la classification des séquences protéiques. La méthode consiste à extraire toutes les sous-séquences possibles de longueur p sur un alphabet de vingt acides aminés. Ensuite, la matrice du noyau est implicitement calculée. L'objectif consiste à évaluer le noyau SSK en conjonction avec la méthode SVM. Les expérimentations ont été réalisées sur trois familles sélectionnées à partir de la base de données SCOP (Structural Classification of Proteins) (Lo Conte *et al.* 2000). Elles comparent l'approche proposée aux méthodes de détection d'homologie les plus réussies sur des jeux de données SCOP.

Les résultats des expérimentations révèlent que l'approche proposée a bien fonctionné dans la classification des protéines. En outre, la méthode a surpassé toutes les approches génératives. Cependant, dans la plupart des cas, la méthode SVM-Fisher a surpassé l'approche proposée.

Extraction des informations relationnelles

Les systèmes traditionnels d'extraction de l'information (IE, Information Extraction) ont été conçus pour identifier les instances d'une classe d'entités particulière, de relations et d'événements dans le traitement du langage naturel (Piskorski & Yangarber 2013). Malheureusement, ces systèmes traditionnels (IE) ne fonctionnent pas bien sur les corpus biomédicaux comme sur les corpus sur lesquels ils ont passé la phase d'apprentissage (Mooney & Bunescu 2006).

Un remède à ce défi est de considérer les techniques de morceaux « chunk » au lieu de celles d'analyse. Dans ce contexte, Mooney & Bunescu 2006 ont proposé une généralisation de la méthode SSK pour extraire les interactions et les relations protéine-protéine. Ici, les sous-séquences sont composées par combinaison de mots et de classe de mots. L'espace de redescription est réduit de sorte qu'il contient un nombre limité de sous-séquences contraintes à contenir trois motifs prédéfinis.

Le noyau SSK généralisé a été évalué sur la tâche d'extraction des relations à partir des corpus biomédicaux et de journaux. Le nouveau noyau a été utilisé conjointement avec la méthode SVM. Les expérimentations sur l'extraction des interactions protéiques montrent de meilleures performances que les systèmes à base de règles de Blaschke & Valencia 2001; Valencia & Blaschke 2002. Aussi, sur la tâche d'extraction des relations de haut niveau, les résultats démontrent l'avantage du nouveau noyau SSK généralisé par rapport au noyau d'arbre de dépendance utilisé dans Culotta & Sorensen 2004.

Analyse sémantique

L'analyse sémantique est la tâche de traduire le langage naturel en une représentation ayant une signification dans la machine. En raison de la flexibilité des langages naturels, il est difficile pour les méthodes fondées sur des règles ou sur les statistiques d'énumérer tous les contextes du langage naturel qui correspondent à un concept sémantique. Les noyaux de séquences présentent également un mécanisme qui peut implicitement capturer cette gamme de contextes. Selon cette vision, Kate & Mooney 2006 ont proposé une approche basée sur le noyau de sous-séquences de mots pour apprendre les analyseurs sémantiques appelés KRISP (Kernel-based Robust Interpretation for Semantic Parsers).

Les données d'apprentissage comprennent des phrases en langage naturel associées à leurs représentations sémantiques formelles. KRISP traite les productions grammaticales du langage de la représentation sémantique comme étant des concepts sémantiques. Le système est entraîné avec la méthode d'apprentissage SVM en utilisant le noyau sous-séquences de mots comme mesure de similarité entre les productions. Ici, les productions sont considérées comme des chaînes de mots. KRISP a été évalué sur deux ensembles de données réels en utilisant une validation croisée 10-fold. Les résultats montrent que KRISP se compare favorablement à la performance de l'état de l'art et il est particulièrement robuste au bruit.

Lambda pruning pour la classification et le clustering SVM

L'efficacité du calcul est une propriété clé des méthodes à noyaux. Pour remédier à ce problème, Seewald & Kleedorfer 2007 ont introduit une approximation du noyau SSK, appelée SSK-LP pour l'élagage Lambda du noyau sous-séquences de mots (String Subsequence Kernel Lambda Pruning). L'idée derrière cette approximation est que la contribution des correspondances trop étirées est faible au résultat global et nécessite plus d'effort de calcul. Ainsi, le calcul de la correspondance peut être arrêté lorsque ses effets sont très faibles. Pour ce faire, les auteurs ont introduit une borne sur la somme de $l(I) + l(J)$ dans l'Équation 3.11.

De plus, les auteurs ont créé des modèles de consommation maximale de la mémoire et de temps d'exécution moyen pour les méthodes SSK et SSK-LP. Ces modèles sont utilisés pour prévoir le temps d'exécution et l'utilisation de la mémoire pour les deux variantes SSK. Une telle prévision permet le choix automatique entre les variantes SSK dans les applications pratiques.

Les méthodes SSK et SSK-LP ont été évaluées dans plusieurs tâches d'apprentissage : texte mining, filtrage de spam et clustering de redondances.

Pour le texte mining, les auteurs ont tenté de classer les entrées bibliographiques collectées à partir des références MEDLINE (Medical Literature, Analysis, and Retrieval System Online). La méthode SSK n'a pas été prise en compte en raison de son temps d'exécution estimé élevé. Les résultats des expérimentations indiquent que SSK-LP est compétitif en précision par rapport à la méthode SVM

linéaire avec un vecteur de mot en entrée. Mais, en terme de temps d'exécution, les résultats ne sont pas encourageants.

La deuxième tâche d'apprentissage est le filtrage de spam. La tâche est réduite à la classification de mots. Les auteurs impliquent 3902 courriels en tant que ensemble de données. En raison de la petite taille des mots, SSK et SSK-LP peuvent être exécutées sur cet ensemble de données. Les résultats de l'expérimentation indiquent que le temps d'exécution de SSK-LP est d'un ordre de grandeur par rapport à SSK et que leurs précisions sont très similaires.

La dernière tâche d'apprentissage est le clustering de redondances. Il est utilisé pour déterminer et supprimer des phrases redondantes d'un jeu de résultats pour donner une présentation concise à l'utilisateur. Cela peut être vu comme une tâche de résumé. Les auteurs utilisent le corpus BioMinT (Biological Text Mining) avec des phrases redondantes, où la tâche est réduite à la similarité par paires entre les phrases. Pour mener des expérimentations, les auteurs ont défini cinq façons de déterminer les mesures de similarité, y compris SSK et SSK-LP. Les résultats des expérimentations attestent que SSK fonctionne mieux que l'approche de base qui considère la similarité comme le nombre de mots communs entre deux phrases. En outre, SSK-LP peut être utilisé comme une mesure de similarité compétitive pour le clustering de redondances.

Nous pouvons conclure des évaluations ci-dessus que les noyaux de séquences, au delà d'être des mesures de similarité, ils peuvent être utiles dans le cas des tâches de classification de mots où la sélection des descripteurs demeure difficile. De plus, même si les améliorations de SSK-LP en termes de complexité temporelle sont évidentes, plus d'efforts sont nécessaires pour remédier aux insuffisances de la méthode SSK, en particulier sur des machines avec des ressources limitées.

Détection de malwares

La détection de malwares (logiciels malveillants) est l'un des sujets les plus intéressants de la sécurité informatique. La détection de malwares basés sur la signature fait référence à une liste noire contenant des modèles identifiables connus sous le nom de signature pour déterminer si un objet est infecté par un

malware. Ceci implique que les malwares inconnus ne peuvent pas être détectés par cette technique.

Pour remédier à cet inconvénient, l'analyse basée sur l'apprentissage automatique considère le diagnostic de comportement plutôt que l'analyse de l'objet lui-même. Néanmoins, la classification se fait fréquemment hors ligne ce qui rend cette technique impraticable. Pour combler cette lacune, une détection de malwares basée sur le noyau de mots a été proposée (Pfoh *et al.* 2013). Les auteurs modélisent le comportement du programme par le biais de traces d'appel système tels que les opérations sur les fichiers, les communications réseaux et les communications inter-processus. Ils représentent des traces d'appel système en tant que séquences et le noyau SSK en tant que mesure de similarité.

Le problème du noyau SSK sur l'ensemble des traces d'appel système est le nombre croissant d'appariements des sous-séquences entraînant un temps d'exécution important. Le problème a été résolu par l'introduction d'une technique de fenêtre glissante (Pfoh *et al.* 2013).

L'approche proposée utilise la méthode SVM en conjonction avec la métrique SSK pour apprendre un classifieur capable de séparer les traces d'appel système. Le deux phases d'apprentissage et de test ont été réalisées sur des données réelles collectées à partir de Windows XP SP3. Les expérimentations révèlent que la méthode proposée montre des résultats prometteurs. De plus, par rapport à d'autres approches basées sur l'apprentissage automatique, l'approche basée sur les noyaux de mots fonctionne très bien.

3.4.7 Noyaux rationnels de mots

Dans cette section, nous présentons une famille de noyaux à base de transducteurs pondérés, appelés noyaux rationnels (Cortes *et al.* 2004; Cortes *et al.* 2002; Mohri *et al.* 2012). L'évaluation d'un noyau rationnel fait appel à l'opération de composition de transducteurs pondérés ainsi qu'aux algorithmes du plus court chemin.

Nous avons déjà introduit quelques notions de base concernant les transducteurs pondérés dans la Section 3.1.4, néanmoins, pour la compréhension du processus d'évaluation d'un noyau rationnel, nous aurons besoin de détailler la notion de chemin ainsi que l'opération de composition des transducteurs.

Chemins

Soit un transducteur pondéré $T = (\Sigma, \Delta, Q, E, I, F, \lambda, \rho)$ sur un semi-anneau $\mathbb{S}$, un chemin π de T est une suite de transitions de telle sorte que l'origine de chacune est la destination de la précédente. Un chemin avec cycle est un chemin dont lequel son origine coïncide avec sa destination. Un transducteur est dit acyclique s'il n'admet pas de cycle. Un chemin réussi (aussi appelé calcul) de T est un chemin à partir d'un état initial à un état final. L'étiquette d'entrée d'un chemin π est un mot de Σ^* obtenue par la concaténation des étiquettes des transitions qui le constituent. De même, l'étiquette de sortie d'un chemin π est un mot de Δ^*.

Un transducteur pondéré peut être considéré comme une fonction définie sur $\Sigma^* \times \Delta^*$ à valeurs dans le sous-anneau $\mathbb{S}$. Le poids associé par T à une paire de mot (x, y) est dénoté par $T(x, y)$ et obtenu en additionnant (opération d'addition du semi-anneau) les poids de tous les chemins réussis avec l'entrée x et la sortie y. Plus formellement :

$$T(x, y) = \sum_{\pi \in P(I,x,y,F)} \lambda(p[\pi]) \cdot w[\pi] \cdot \rho(n[\pi]),$$

où $p[\pi]$ (resp. $n[\pi]$) est l'état source (resp. destination) du chemin π ; $w[\pi]$ est la pondération du chemin calculée comme étant le produit des poids des transitions qui le constituent ; $P(I, x, y, F)$ est l'ensemble des chemins avec un état initial de I, un état final de F, une étiquette d'entrée x et une étiquette de sortie y.

Composition

La composition est une opération intéressante des transducteurs. Elle peut être utilisée dans de nombreuses applications pour créer des transducteurs pondérés complexes à partir de ceux plus simples. Dans le contexte des noyaux rationnels, elle est utile pour créer et évaluer des noyaux de séquences. En effet, sa définition découle de la composition des relations.

Étant donné deux transducteurs pondérés $T_1 = (\Sigma, \Delta, Q_1, E_1, I_1, F_1, \lambda_1, \rho_1)$ et $T_2 = (\Delta, \Omega, Q_2, E_2, I_2, F_2, \lambda_2, \rho_2)$ sur un semi-anneau $\mathbb{S}$ de telle sorte que l'alphabet de sortie de T_1 coïncide avec celle d'entrée de T_2. On définit le transducteur

pondéré $T = (\Sigma, \Omega, Q, E, I, F, \lambda, \rho)$ de composition de T_1 et T_2 comme suit :

$$T(x, y) = T_1 \circ T_2(x, y) = \sum_{z \in \Delta^*} T_1(x, z) \cdot T_2(z, y). \tag{3.20}$$

On peut déduire à partir de la formule (3.20) que la composition est similaire à la multiplication des matrices dans le cas infini.

Afin d'éclaircir cette notion de composition, nous commençons par la description d'un algorithme de composition standard introduit dans Cortes *et al.* 2004; Mohri *et al.* 1996; Pereira & Riley 1997. Les états $Q = Q_1 \times Q_2$ du transducteur pondéré T sont des paires d'un état de T_1 et d'un état de T_2. De même pour les états initiaux et les états finaux de T, $I = I_1 \times I_2$ et $F = F_1 \times F_2$. Sans tenir compte des transitions ϵ, chaque transition $e = ((q_1, q_2), a, c, w_1 \cdot w_2, (q_1', q_2')) \in E$ est construite à partir de $e_1 = (q_1, a, b, w_1, q_1') \in E_1$ et $e_2 = (q_2, b, c, w_2, q_2') \in E_2$.

Dans le pire des cas, toutes les transitions de T_1 partant de q_1 correspondent à celles de T_2 partant de q_2. Ainsi, la composition standard requiert un temps d'exécution et un espace mémoire quadratique, $O((|Q_1| + |E_1|)(|Q_2| + |E_2|))$.

Cependant, une généralisation naïve de cet algorithme pour les cas des transitions ϵ peut générer des chemins-ϵ redondants, ce qui conduit à des résultats incorrects pour les semi-anneaux non idempotents. En effet, le problème surgira lors de la composition d'une transition $e_1 = (q_1, a, \epsilon, w_1, q_1')$ de T_1 avec une transition $e_2 = (q_2, \epsilon, b, w_2, q_2')$ de T_2. Ceci donne trois chemins différents, $\pi_1 = (((q_1, q_2), a, \epsilon, w_1, (q_1', q_2)), ((q_1', q_2), \epsilon, b, w_2, (q_1', q_2')))$, $\pi_2 = (((q_1, q_2), \epsilon, b, (q_1, q_2')), (q_1, q_2', a, \epsilon, w_1, (q_1', q_2')))$, $\pi_3 = (((q_1, q_2), a, b, w_1 \cdot w_2, (q_1', q_2')))$.

Une solution (Mohri *et al.* 1996; Pereira & Riley 1997) consiste à éviter ce type de correspondance des ϵs. Ceci peut être réalisé par la dérivation de deux transducteurs pondérés $\tilde{T}_1$ (resp. $\tilde{T}_2$) à partir de T_1 (resp. T_2) en remplaçant les étiquettes de sortie ϵ de T_1 par ϵ_2 et les étiquettes d'entrée ϵ de T_2 par ϵ_1. Cependant, on doit garder un seul chemin-ϵ parmi d'autres. Ce mécanisme de filtrage est mis en œuvre par le biais d'un transducteur pondéré. Un tel transducteur filtre F (Figure 3.3) est inséré entre $\tilde{T}_1$ et $\tilde{T}_2$. Ainsi $\tilde{T}_1 \circ F \circ \tilde{T}_2 = T_1 \circ T_2$. Étant donné que les deux compositions $\tilde{T}_1 \circ F \circ \tilde{T}_2$ n'impliquent pas les transitions-ϵ, l'algorithme de composition standard sus-décrit pourra être appliqué.

En terme de complexité, il est facile de voir que la taille du transducteur filtre F est constante. Alors la complexité de composition $(\tilde{T}_1 \circ F \circ \tilde{T}_2)$ rejoint celle de l'algorithme de composition standard, soit $O((|Q_1| + |E_1|)(|Q_2| + |E_2|))$.

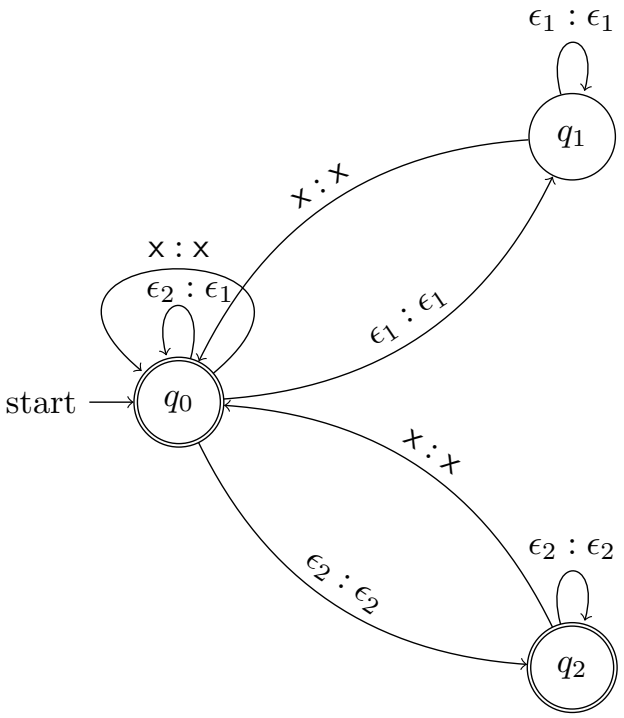


FIGURE 3.3 – Transducteur filtre, x représente un élément de Σ. Extrait de Mohri *et al.* 2012.

Noyaux rationnels

Les noyaux rationnels (Cortes *et al.* 2004; Mohri *et al.* 2012; Cortes *et al.* 2002) sont des mesures de similarité sur des ensembles de séquences. Étant donné que les ensembles de séquences peuvent être représentés d'une manière compacte par des automates, les noyaux rationnels peuvent être considérés comme des noyaux sur des automates pondérés.

Définition 3.1. Un noyau K est dit rationnel s'il existe un transducteur pondéré $T = (\Sigma, \Delta, Q, E, I, F, \lambda, \rho)$ sur un semi-anneau $\mathbb{S}$ et une fonction $\Psi : \mathbb{S} \to \mathbb{R}$ tel que pour tout $x, y \in \Sigma^*$:

$$K(x, y) = \Psi(T(x, y)). \tag{3.21}$$

Le noyau est défini donc par le couple (Ψ, T).

Dans la littérature (Cortes *et al.* 2004; Mohri *et al.* 2012), il existe un algorithme simple pour l'évaluation des noyaux rationnels. Soit un transducteur pondéré T définissant un noyau rationnel K. Le transducteur T n'admet aucun cycle-ϵ avec un poids non nul. Sinon la valeur du noyau est infinie pour toute séquence x. Soit T_x un transducteur pondéré représentant la séquence x. Il comporte un seul chemin réussi avec un poids 1 et une étiquette d'entrée et de sortie x. Il est clair que T_x peut être construit en un temps linéaire $O(|x|)$. Pour tous $x, y \in \Sigma^*$, $T(x, y)$ peut être évalué en suivant les étapes suivantes :

- Calculer $T_k = T_x \circ T \circ T_y$ en utilisant l'algorithme de composition des transducteurs pondérés décrit auparavant. Ceci est réalisé en un temps $O(|T_x|\,|T|\,|T_y|)$, où $|T_x|$, $|T|$ et $|T_y|$ désignent respectivement la taille des transducteurs $T_x,\ T$ et T_y,
- calculer la somme des poids de tous les chemins réussis en utilisant un algorithme de plus courte distance. La complexité de cette opération est $O(|T_k|)$,
- calculer $\Psi(T_k(x, y))$. Ceci peut être effectué dans un temps $O(\Psi)$, où $O(\Psi)$ est le temps d'exécution dans le pire des cas de $\Psi(s),\ s \in \mathbb{S}$.

Il est évident que la complexité totale de l'algorithme d'évaluation du noyau rationnel est $O(|T_x|\,|T|\,|T_y| + \Psi)$. Si nous supposons que la fonction Ψ peut être évaluée dans un temps constant (comme c'est le cas dans la pratique), la complexité de $K(T_x, T_y)$ devient $O(|T_x|\,|T|\,|T_y|)$.

Noyaux rationnels définis positifs symétriques

La définition d'un noyau rationnel fait appel aux transducteurs pondérés, mais ce n'est pas n'importe quel transducteur qui donne naissance à un noyau. le théorème 3.2 présente une méthode générale pour construire un noyau rationnel à partir d'un transducteur pondéré arbitraire. En effet, le Théorème fait appel à la propriété définie positive symétrique évoquée dans la Section (2.3.3) qui est une caractérisation des fonctions noyaux.

Théorème 3.2. *Pour tout transducteur pondéré $T = (\Sigma, \Delta, Q, E, I, F, \lambda, \rho)$, la fonction $K = T \circ T^{-1}$ est un noyau rationnel défini positif symétrique.*

Le transducteur pondéré T^{-1} désigne l'inverse de T, c'est bien le transducteur obtenu à partir de T en transposant les étiquettes d'entrée et de sortie

de chaque transition. Pour la preuve, les lecteurs peuvent trouver le détail dans Cortes *et al.* 2004; Mohri *et al.* 2012.

Du transducteur compteur au noyau rationnel

Nous avons déjà introduit lors de la définition des noyaux de séquences (Section 3.4) leur principe qui repose sur le comptage des motifs communs apparaissant dans les séquences. Il existe une méthode simple pour construire un transducteur qui compte le nombre d'occurrences d'un motif dans une séquence.

Soit X un automate représentant l'ensemble des motifs à compter. Le transducteur pondéré T_{count} de la Figure 3.4 (Théorème 3.3) peut être utilisé pour compter exactement le nombre d'occurrences du motif accepté par X pour un alphabet $\Sigma = \{a, b\}$. La transition $X : X/1$ représente le transducteur pondéré crée à partir de l'automate X en ajoutant à chaque transition un libellé de sortie identique au libellé existant et un poids égale à 1. Également, on associe un poids égale à 1 aux état finaux.

Théorème 3.3. *Pour tout $x \in \Sigma^*$ et pour tout z accepté par l'automate X, $T_{count}(x, z)$ représente le nombre d'occurrences du motif accepté z dans x.*

Nous laissons le soin au lecteur d'examiner la preuve dans Mohri *et al.* 2012.

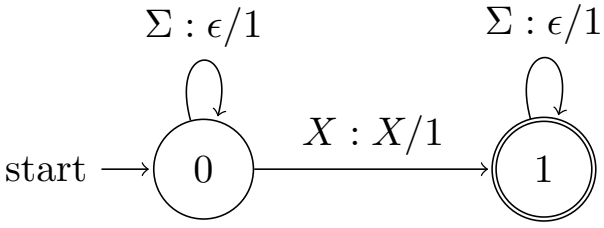


FIGURE 3.4 – Transducteur compteur T_{count} pour $\Sigma = \{a, b\}$. Extrait de Mohri *et al.* 2012.

Le transducteur pondéré T_{count} peut être utilisé pour créer un noyau rationnel positif défini symétrique $T_{count} \circ T_{count}^{-1}$ en utilisant le résultat du Théorème 3.2. Par la suite, à titre d'illustration nous présentons quelques transducteurs compteurs ainsi que les noyaux rationnels correspondants.

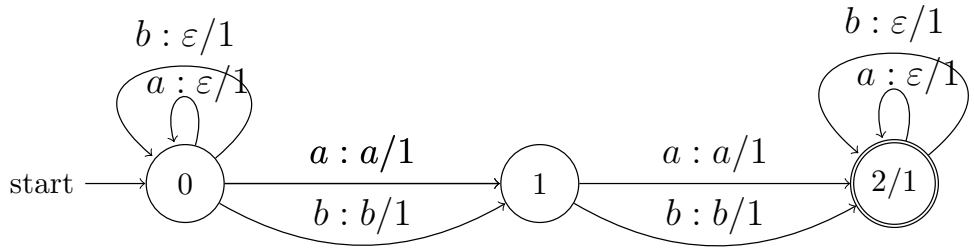


FIGURE 3.5 – Transducteur compteur bigram T_{bigram} pour $\Sigma = \{a, b\}$. Extrait de Cortes *et al.* 2004; Mohri *et al.* 2012; Cortes *et al.* 2002.

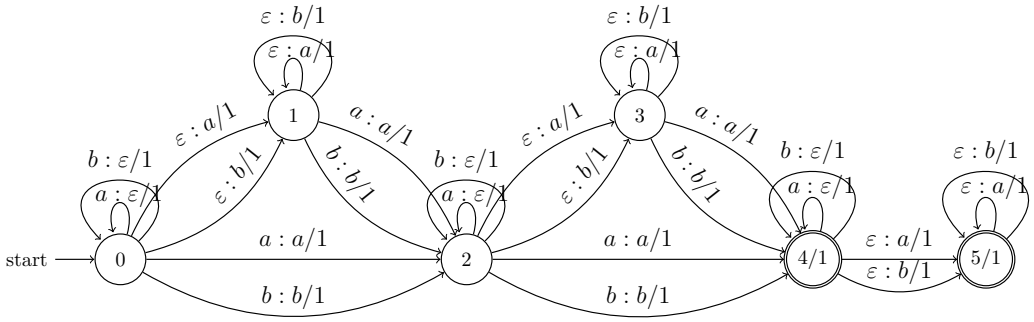


FIGURE 3.6 – Noyau rationnel bigram pour $\Sigma = \{a, b\}$. Extrait de Cortes *et al.* 2004.

Noyaux rationnels de séquences bigram et bigram avec « trous »

Nous commençons par la création d'un transducteur compteur T_{bigram} (Figure 3.5) sur un alphabet réduit $\Sigma = \{a, b\}$. Pour tout $x \in \Sigma^*$ et pour tout bigram $z \in \{aa,\ ab,\ ba,\ bb\}$ $T_{bigram}(x, z)$ est exactement le nombre d'occurrences du bigram z dans x.

La création du noyau rationnel bigram correspondant au transducteur compteur bigram est illustré par la Figure 3.6. Elle s'effectue par la composition de T_{bigram} avec son transducteur inverse T_{bigram}^{-1} ($T_{bigram} \circ T_{bigram}^{-1}$).

De la même manière, on peut créer un transducteur pour compter les bigrams avec « trous » (Figure 3.7) est le noyau correspondant (Figure 3.8).

Il est à noter que l'opération fondamentale pour les noyaux rationnels est l'opération de composition des transducteurs pondérés. Une opération un peu coûteuse, notamment pour les noyaux, qui nécessite d'être efficacement implémentés. Dans cette perspective, Allauzen & Mohri 2009 ont proposé une nouvelle méthode de composition des transducteurs appelée « N-Way Composition »

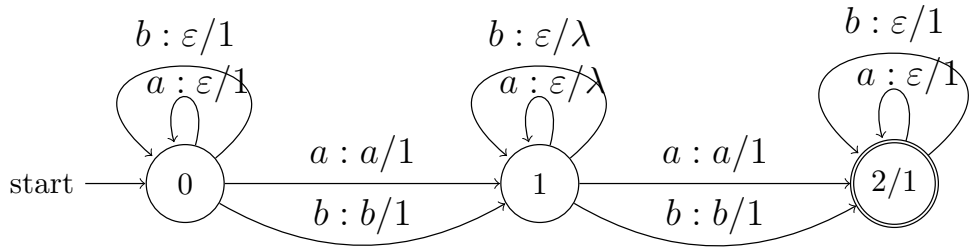


FIGURE 3.7 – Transducteur bigram avec « trous » T_{gappy_bigram} pour $\Sigma = \{a, b\}$ avec un facteur de pénalisation λ. Extrait de Mohri *et al.* 2012; Cortes *et al.* 2002.

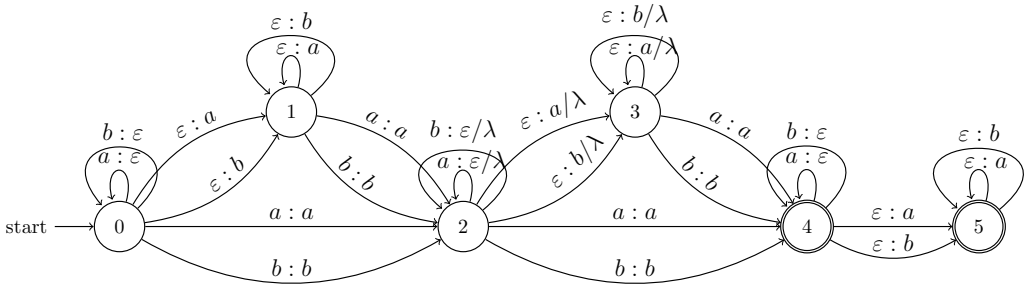


FIGURE 3.8 – Noyau rationnel bigram avec « trous » pour $\Sigma = \{a, b\}$ avec un facteur de pénalisation λ. Les pondérations des transitions non marquées sont égales à 1. Extrait de Cortes *et al.* 2002.

qui s'applique sur trois transducteurs ou plus. Cette nouvelle méthode de composition peut accélérer le processus d'évaluation des noyaux rationnels d'une manière remarquable.

3.5 Conclusion

Dans ce chapitre, nous avons adopté une démarche qui consiste à introduire les noyaux pour le texte selon la représentation des données. Nous avons considéré quatre représentations essentielles, à savoir, sacs de mots, vecteurs de concepts, séquences de symboles et sous forme de transducteurs pour lesquelles nous avons associé respectivement les noyaux des espaces vectoriels, sémantiques, de séquences et rationnels.

Un grand intérêt est accordé aux noyaux de séquences ou de mots, en particulier, le noyau sous-séquences de mots (String Subsequence Kernel, SSK) car il fait l'objet de notre contribution (Approche géométrique) traitée dans le Chapitre 5.

Chapitre 4

Noyaux d'arbres

LES arbres sont des structures de données qui surgissent naturellement pour représenter divers types de données dans le monde réel. De nombreux domaines d'applications impliquent une telle structure de données. À titre d'exemple, les arbres syntaxiques dans le traitement automatique de la langue, les documents HTML dans la recherche d'information, les structures de molécules dans la chimie numérique, les protocoles réseau dans la sécurité informatique et quelques représentations des images dans la vision par ordinateur.

Par ailleurs, la structure hiérarchique des arbres reflète la dépendance sous-jacente des informations du domaine qu'elle représente. En effet, une telle dépendance est indispensable lors du processus d'apprentissage ; elle embarque des informations pertinentes. En général, les approches de représentation plates des arbres, sous forme de vecteurs, ne parviennent pas à capturer ces dépendances. Cependant les méthodes à noyaux évitent d'accéder directement à l'espace de redescription, car il est possible de remplacer le produit scalaire par une fonction noyau qui calcule la similarité entre deux arbres directement dans leur espace d'entrée. Ainsi, les noyaux d'arbres constituent l'outil adéquat pour évaluer la similarité entre deux arbres.

Nous commençons par l'introduction du concept arbre ainsi que le paradigme de conception des noyaux d'arbres. Ensuite, nous entamons l'étude des noyaux d'arbres selon que les arbres soient ordonnés ou non.

4.1 Définitions

Avant d'entamer la description des noyaux d'arbres en détail, selon que les arbres soient ordonnés ou non, il est commode d'introduire quelques définitions et notations qui seront utilisées par la suite.

4.1.1 Graphes

Un *graphe* $G = (V, E)$ est un couple formé de deux ensembles, un ensemble V de sommets (vertices or nodes) et un sous ensemble E de $V \times V$ de *paires* (non ordonnées) de sommets distincts appelées arêtes (edges). Un chemin $P(v_i, v_j) = v_i \dots v_j$ est une séquence de sommets connectés par des arêtes. Il est simple s'il ne comporte pas de sommets répétés. Un cycle est un chemin avec au moins une arête dont le premier et le dernier sommet sont les mêmes. Par exemple $P = v_i \dots v_j$ avec $v_i = v_j$. La longueur d'un chemin ou d'un cycle est le nombre de ses arêtes.

Un graphe est sans cycle ou *acyclique*, s'il ne possède pas de cycles. Il est dit *orienté* (digraph) si l'ensemble E des *arêtes*, en général appelées *arcs*, est constitué de *couples* de sommets. Un couple étant ordonné contrairement à une paire. Dans le cas des graphes orientés, un circuit est un chemin avec au moins un arc dont le premier et le dernier sommet sont les mêmes. Un graphe est connexe s'il existe un chemin à partir de chaque sommet à n'importe quel autre sommet. Si une étiquette est attachée à chaque sommet, le graphe est dit *étiqueté*.

Le *degré* $d(v)$ d'un sommet v est le nombre d'arêtes incidentes à ce sommet. Dans un graphe orienté, $d^-(v)$ est le nombre d'arcs entrants à v, tandis que $d^+(v)$ est le nombre d'arcs sortants de v.

4.1.2 Arbres

Un *arbre* T est un graphe orienté acyclique dans lequel le degré entrant de chaque sommet (généralement appelé nœud pour les arbres) est au plus un. Un arbre *enraciné* est un arbre dans lequel il existe un nœud particulier avec le degré entrant zéro ($d^-(v) = 0$). Un tel nœud est dit racine de l'arbre T, noté $r(T)$.

Une *feuille* est un nœud v avec un degré sortant zéro ($d^+(v) = 0$). Tandis qu'un nœud avec $d^+(v) \neq 0$ est un nœud *interne*. S'il existe un arc e_{ij}, alors v_i est un nœud *parent* et v_j est un nœud *fils* ou *enfant*. Si v_j et v_k sont des fils d'un même parent v_i alors ils sont dits *frères*. Un nœud v_j est *descendant* d'un noeud v_i s'il existe un chemin de v_i vers v_j, dans ce cas v_i est *ascendant* de v_j.

Un *arbre ordonné* est un arbre dans lequel l'ensemble des fils de chaque nœud est ordonné selon une certaine relation. Un *arbre propre* est un arbre composé d'au moins deux nœuds. La taille $|T|$ d'un arbre est le nombre de ses nœuds. Deux arbres sont identiques s'il existe une correspondance un-un entre les nœuds, ceci en respectant la relation parent-fils ainsi que la relation d'ordre entre les fils de chaque nœud. Deux arbres étiquetés sont identiques s'ils sont identiques au sens des arbres et les nœuds qui correspondent portent les mêmes étiquettes.

Nous pouvons distinguer plusieurs types de sous arbres (Shawe-Taylor & Cristianini 2004) :

- Un *sous arbre complet* d'un arbre T à un nœud v est l'arbre obtenu en prenant tous les nœuds et arcs accessibles à partir de v. La Figure 4.1 montre un arbre avec ses sous-arbres complets (nous ne considérons que les arbres propres).
- Un *sous arbre co-enraciné* d'un arbre T est obtenu en retranchant une séquence de sous arbres complets et en les remplaçant par leurs racines. Par conséquent, si un nœud v est inclus dans un arbre co-enraciné, alors ces frères le sont.
- Un *sous arbre général* d'un arbre T est n'importe quel sous arbre co-enraciné d'un sous arbre complet. La Figure 4.2 montre un arbre avec certains de ses sous-arbres généraux.

4.2 Paradigme de conception des noyaux d'arbres

Nous avons déjà introduit dans la Section 3.4.1 le paradigme des noyaux de convolution (Haussler 1999) qui constitue une plate-forme générale pour construire des noyaux semi-définis positifs sur des structures discrètes (séquences, arbres, graphes, ...).

Plus tard, Shin & Kuboyama 2008 ont proposé une généralisation des noyaux de convolution, à savoir les *noyaux d'application* (mapping kernels). Le principe

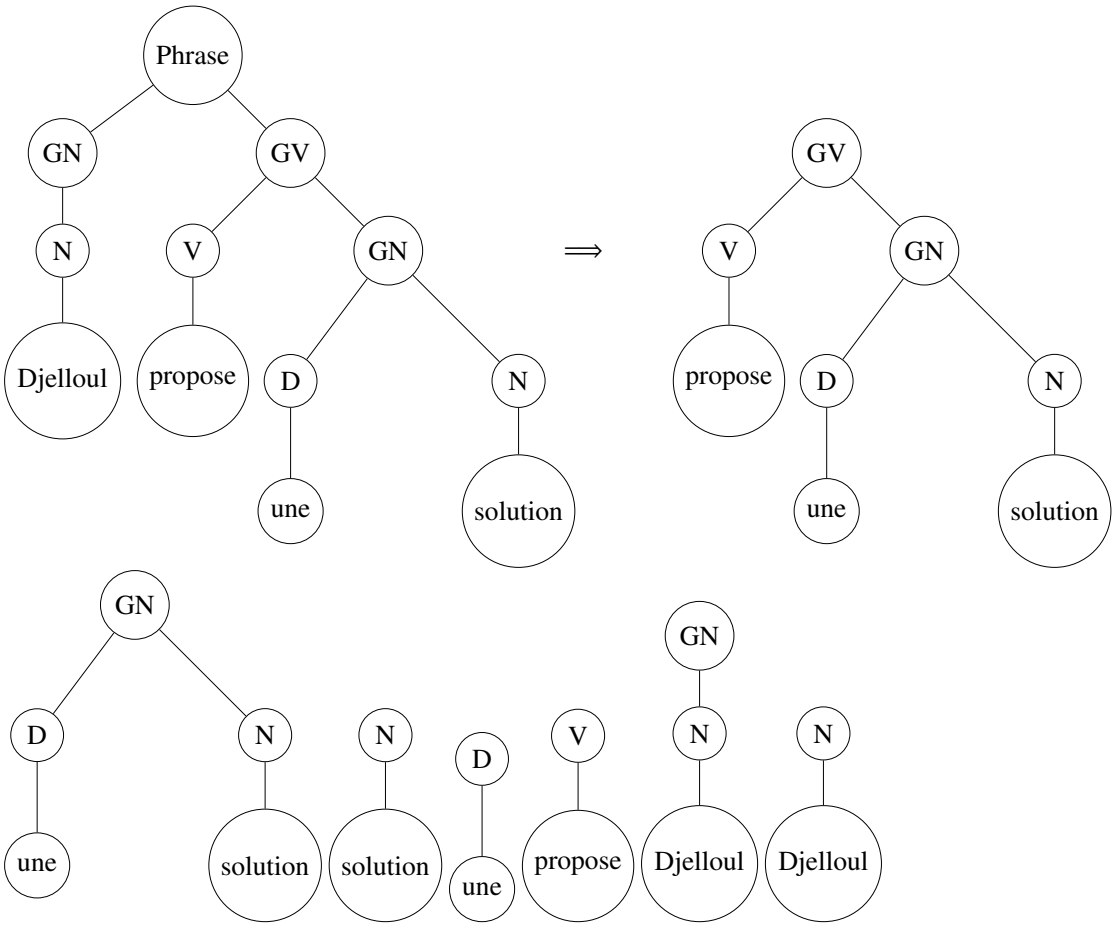


FIGURE 4.1 – Un arbre avec ses sous-arbres complets.

des noyaux d'application est utilisé pour concevoir de nouveaux noyaux en appliquant certaines contraintes sur l'espace des sous structures hérité du noyau de convolution correspondant par un système d'application. Le noyau d'application permet d'écarter les sous structures ineffectives pour une certaine tâche afin de permettre à d'autres structures effectives d'être plus pertinentes.

$$K(\mathbf{x}, \mathbf{y}) = \sum_{(u,v) \in \mathcal{M}_{\mathbf{x},\mathbf{y}}} k(u, v). \tag{4.1}$$

Formellement, soient $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ les données d'entrée et $k : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ un noyau valide. Alors, le noyau de l'Équation (4.1) est semi-défini positif quand les conditions suivantes sont vérifiées :

1. $\mathcal{M}$ est un ensemble fini et symétrique, défini par le système d'applications $\mathbb{M}$:

$$\mathbb{M} = (\mathbf{X}, \{\mathbf{U}_\mathbf{x} | \mathbf{x} \in \mathbf{X}\}, \{\mathcal{M}_{\mathbf{x},\mathbf{y}} \subseteq \mathbf{U}_\mathbf{x} \times \mathbf{U}_\mathbf{y} | (\mathbf{x}, \mathbf{y}) \in \mathbf{X} \times \mathbf{X}\}). \tag{4.2}$$

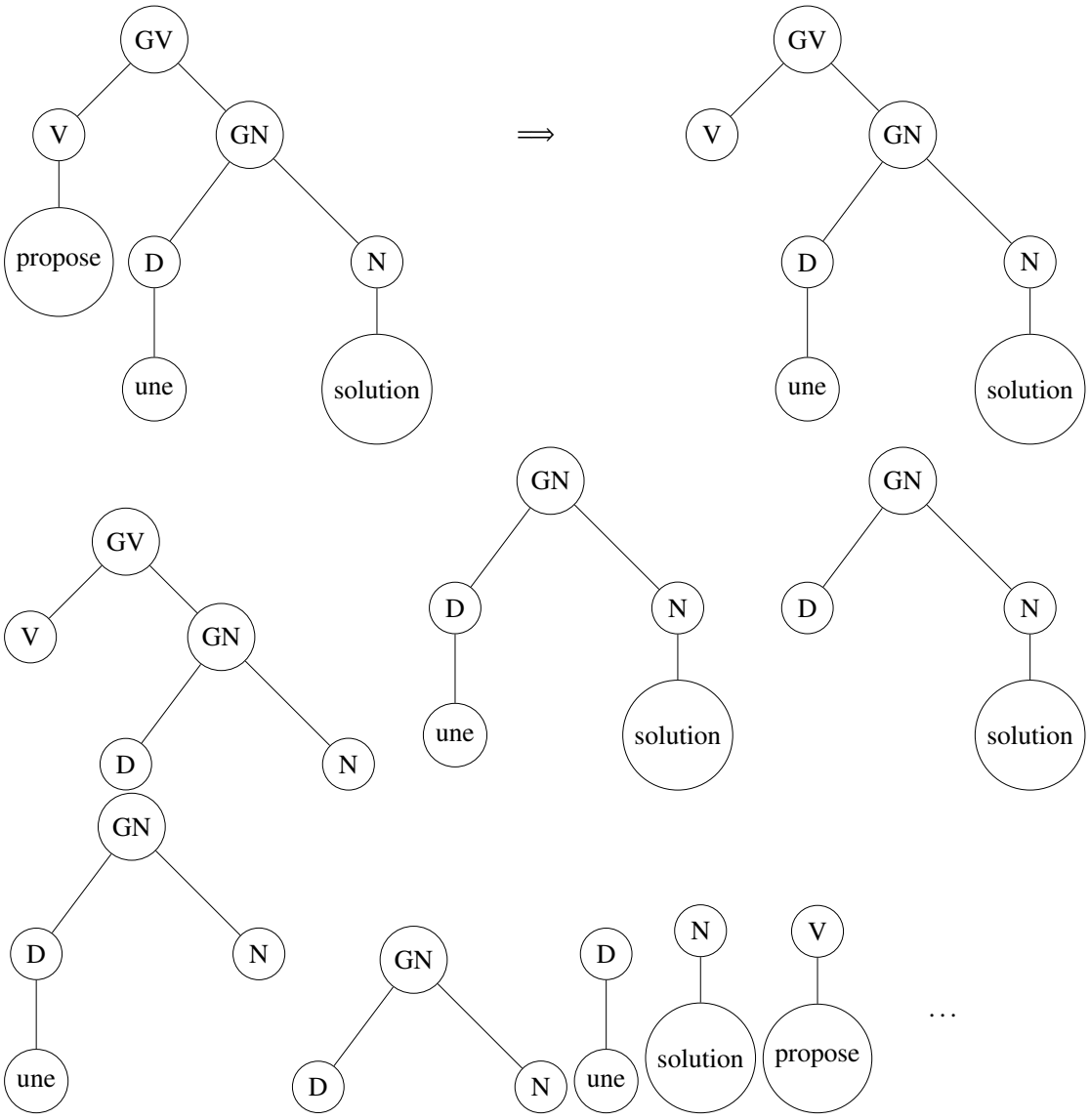


FIGURE 4.2 – Un arbre avec certains de ses sous-arbres généraux.

2. $\mathbb{M}$ est transitive, tel que pour tous $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{X}$

$$(u, v) \in \mathcal{M}_{\mathbf{x},\mathbf{y}} \wedge (v, w) \in \mathcal{M}_{\mathbf{y},\mathbf{z}} \Rightarrow (u, w) \in \mathcal{M}_{\mathbf{x},\mathbf{z}} \tag{4.3}$$

Le triplet $\mathbb{M}$ est composé par le domaine des exemples $\mathbf{X}$, l'espace des sous structures $U_{\mathbf{x}}$ et l'espace de redescription spécifié par $\mathcal{M}$, qui est un sous ensemble du produit cartésien de l'espace des sous structures.

Étant donné le système d'application de l'Équation (4.2), la propriété de transitivité de $\mathbb{M}$ introduite par l'Équation (4.3) est une condition nécessaire et suffisante pour que l'Équation (4.1) soit un noyau valide. Pour clarifier l'idée des noyaux d'application, nous reprenons un petit exemple d'un noyau linéaire

entre deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ (Da San Martino 2009), $K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} \mathbf{x}_i \mathbf{y}_i$. Un tel noyau peut être vu comme un cas particulier d'un noyau d'application où :

- Les vecteurs m-dimensionnel sont décrits par un ensemble de paires $(\mathbf{x}_i, i) \in \mathbf{X} \times \mathbb{N}$.
- $\mathbf{U}_\mathbf{x}$ est l'ensemble des paires constitué des composantes du vecteur $\mathbf{x}$ et leurs positions dans le vecteur, où $((\mathbf{x}', i), (\mathbf{y}', j)) \in \mathcal{M}_{\mathbf{x},\mathbf{y}}$ si $i = j$.

Par exemple le vecteur 3-dimensionnel $\mathbf{x} = (1, 4, 5)$ est décrit par l'ensemble $\{(1, 1), (4, 2), (5, 3)\}$. Le noyau est défini comme suit :

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \sum_j [i = j] \, \mathbf{x}_i \mathbf{y}_j = \sum_i \mathbf{x}_i \mathbf{y}_i.$$

L'un des principaux avantages de l'utilisation du noyau d'application consiste à donner la condition nécessaire et suffisante pour construire des noyaux sur des objets complexes qui est typiquement plus facile à prouver que la propriété d'être semi-défini positif.

Par ailleurs, le noyau d'application sur des données structurées possède un large éventail d'applications. À titre d'exemple, Shin & Kuboyama 2010 ont effectué une étude exhaustive et ont reporté que 18 parmi 19 noyaux d'arbres de différents types de la littérature peuvent être définis par le noyau d'application d'une manière simple.

4.3 Noyaux d'arbres ordonnés

La plupart des applications des noyaux pour les arbres impliquent des arbres ordonnés, dont la majorité fait partie du paradigme des noyaux d'applications introduit par la Section 4.2. Dans les sections qui suivent, nous en examinons quelques-uns, en considérant la variété des sous-structures.

4.3.1 Noyau sous-arbres

Vishwanathan & Smola 2002 ont proposé un algorithme permettant l'appariement des objets discrets tels que les mots et les arbres. Lorsqu'il appliqué aux

arbres, le noyau sous-arbres (subtree kernel, STK) utilise un espace de redescription indexé par des sous-arbres complets. La composante $\Phi_s(T)$ d'un sous-arbre complet t_s représente le nombre d'occurrences de t_s dans T.
Le noyau correspondant est exprimé sous forme d'une somme pondérée sur tous les sous-arbres complets partagés par deux arbres T_1 et T_2.

$$K(T_1, T_2) = \sum_{s \in \Sigma^*} \Phi_s(T_1)\Phi_s(T_2)w_s,$$

où Σ^* est l'ensemble de tous les sous-arbres complets et w_s est une pondération associée à l'arbre t_s.

En effet, le calcul du noyau STK est réduit au calcul du noyau de mots, en commençant par l'encodage des sous-arbres complets sous forme de mots. Pour se faire, on doit définir un ordre lexicographique entre les étiquettes de l'arbre, si elles existent. De plus, on rajoute deux symboles '[' et ']' avec '[' $<$ ']' et '[',']' $<$ $label(v)$ pour toutes les étiquettes de l'arbre. L'encodage d'un arbre complet de racine v est réalisé par la fonction $tag(v)$ décrite comme suit :

- Si v est une feuille non étiquetée alors $tag(v) = [\,]$;
- si v est une feuille étiquetée alors $tag(v) = [label(v)]$;
- si v est un nœud non étiqueté avec des fils $v_1, \ldots, v_c$ alors définir une permutation triée π des nœuds fils tel que $tag(v_{\pi_{(i)}}) \leq tag(v_{\pi_{(j)}})$ si $\pi_{(i)} \leq \pi_{(j)}$. Ainsi, définir $tag(v) = \left[tag(v_{\pi_{(1)}})tag(v_{\pi_{(2)}}) \ldots tag(v_{\pi_{(c)}})\right]$;
- si v est un nœud étiqueté avec des fils $v_1, \ldots, v_c$ alors effectuer les mêmes opérations que l'étape précédente et mettre $tag(v) = \left[label(v)tag(v_{\pi_{(1)}})tag(v_{\pi_{(2)}}) \ldots tag(v_{\pi_{(c)}})\right]$.

À titre d'exemple, l'arbre de la Figure 4.3 peut être représenté par le mot "$[phrase\,[GV\,[GN\,[N\,[solution]]\,[D\,[une]]]\,[V\,[propose]]]\,[GN\,[N\,[Djelloul]]]]$".

Le théorème 1.1 de (Vishwanathan & Smola 2002) résume les résultats obtenus en matière d'arbre avec l nœuds et λ la largeur maximale d'une étiquette :

1. $tag(v)$ peut être calculé dans un temps $(\lambda + 2)(l \log_2 l)$ et requiert un espace de stockage linéaire en fonction du nombre de nœuds dans l'arbre complet de racine v.
2. Chaque sous-mot s de $tag(v)$, commençant par '[' et se terminant par ']' balancé, dispose d'un sous-arbre complet correspondant.

3. Si les arbres T et $\tilde{T}$ sont équivalents (T peut être obtenu à partir de $\tilde{T}$ en permutant les nœuds fils) alors leurs $tag(v)$ est le même. Par ailleurs, $tag(v)$ permet une reconstruction d'un élément unique de la classe d'équivalence.

4.3.2 Noyau d'arbre syntaxique

Le noyau d'arbre syntaxique (Parse Tree Kernel, PTK) (Collins & Duffy 2002) (appelé aussi subset tree kernel) se base sur le comptage des sous-arbres généraux en commun des arbres syntaxiques . Les arbres syntaxiques sont obtenus à partir de la représentation des relations grammaticales entre les mots d'une phrase. Ils sont considérés comme une structure typique dans le traitement automatique de la langue. La Figure 4.3 présente un arbre syntaxique pour la phrase « Djelloul propose une solution ».

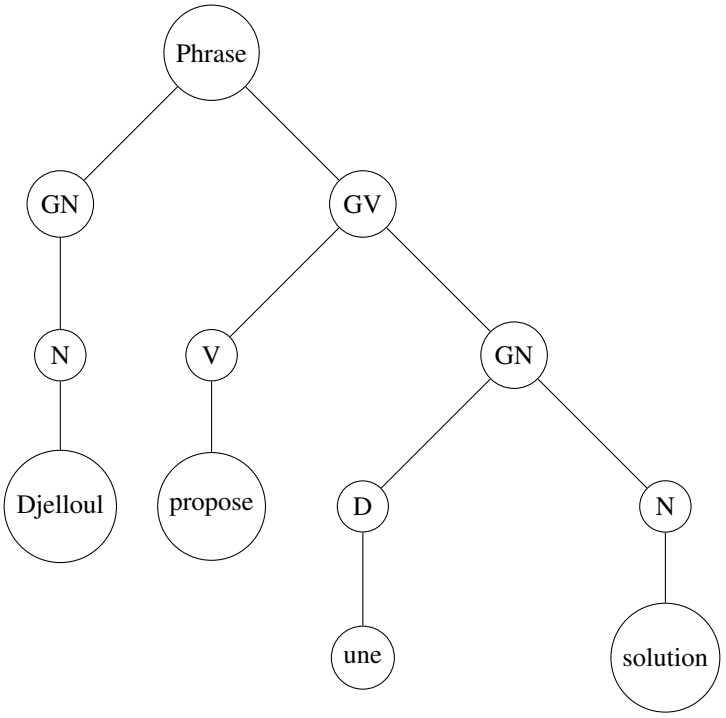


FIGURE 4.3 – Un arbre syntaxique.

Le noyau PTK utilise un espace de redescription indexé par tous les sous-arbres généraux d'un arbre syntaxique T. La composante $\Phi_s(T)$ d'un sous-arbre général t_s représente le nombre d'occurrences de t_s dans T. L'arbre T est représenté par le vecteur

$$\Phi(T) = \left[\Phi_1(T), \Phi_2(T), \ldots, \Phi_{|\mathcal{T}|}(T)\right], \tag{4.4}$$

où $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ est l'espace des sous-structures.

Soient T_1 et T_2 deux arbres, le noyau correspondant est défini comme suit :

$$\begin{aligned} K(T_1, T_2) &= \langle \Phi(T_1), \Phi(T_2) \rangle \\ &= \sum_{s=1}^{|\mathcal{T}|} \Phi_s(T_1)\Phi_s(T_2). \end{aligned} \tag{4.5}$$

Cependant, le calcul explicite de ce noyau est impossible, étant donné que le nombre des sous-arbres est exponentiel en fonction de la taille de l'arbre. La technique du noyau de convolution, permet de calculer le produit scalaire dans l'espace de redescription de haute dimension sans pour autant énumérer explicitement tous les descripteurs.

Dans l'optique de cette dernière considération, le noyau PTK peut être évalué récursivement. Soit $I_s(v)$ une fonction indicatrice égale à 1 si t_s est un sous-arbre co-enraciné à l'arbre T dans v, 0 sinon. Alors, $\Phi_s(T_1) = \sum_{v_1 \in V_1} I_s(v_1)$ et $\Phi_s(T_2) = \sum_{v_2 \in V_2} I_s(v_2)$, où V_1 et V_2 représentent respectivement l'ensemble des nœuds des arbres T_1 et T_2.

Par conséquent, le noyau PTK peut être exprimé ainsi :

$$\begin{aligned} K(T_1, T_2) &= \sum_{s=1}^{|\mathcal{T}|} \sum_{v_1 \in V_1} I_s(v_1) \sum_{v_2 \in V_2} I_s(v_2) \\ &= \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} \sum_{s=1}^{|\mathcal{T}|} I_s(v_1) I_s(v_2) \\ &= \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} C(v_1, v_2), \end{aligned} \tag{4.6}$$

où $C(v_1, v_2) = \sum_{s=1}^{|\mathcal{T}|} I_s(v_1) I_s(v_2)$, elle peut être évaluée récursivement par le bias de la définition suivante :

- Si les productions à v_1 et à v_2 sont différentes, alors $C(v_1, v_2) = 0$,
- si les productions à v_1 et à v_2 sont identiques et v_1 et v_2 disposent seulement des fils feuilles (des symboles pré-terminaux), alors $C(v_1, v_2) = 1$,

- si les productions à v_1 et à v_2 sont identiques et v_1 et v_2 ne sont pas pré-terminaux, alors :

$$C(v_1, v_2) = \prod_{j=1}^{nc(v_1)} (1 + C(ch(v_1, j), ch(v_2, j))), \tag{4.7}$$

où $nc(v)$ est le nombre de fils de v et $ch(v, j)$ renvoie le j^e fils du nœud v. Pour se préoccuper de l'influence de la taille des fragments des sous-arbres sur la valeur du noyau, il est possible, soit de limiter la profondeur des sous-arbres considérés, soit les pénaliser en fonction de leur tailles. Ceci peut être obtenu en introduisant un facteur de pénalisation $\lambda \in]0, 1]$ et en modifiant le cas de base et la définition récursive de C respectivement comme suit :

$$C(v_1, v_2) = \lambda \; et \; C(v_1, v_2) = \lambda \prod_{j=1}^{nc(v_1)} (1 + C(ch(v_1, j), ch(v_2, j))).$$

Ceci correspond à un noyau modifié : $K(T_1, T_2) = \sum_{s=1}^{|\mathcal{T}|} \lambda^{size_s} \Phi_s(T_1)\Phi_s(T_2)$, où $size_s$ est le nombre de nœuds du sous-arbre t_s.

L'utilisation de la technique de la programmation dynamique conduit à une complexité de calcul du noyau PTK dans le pire des cas $O(n^2)$, où n est le nombre de nœuds de l'arbre d'entrée de plus grande taille.

4.3.3 Noyau sous-arbre élastique

Le noyau sous-arbre élastique est proposé dans Kashima & Koyanagi 2002 en tant qu'extension du noyau d'arbre syntaxique (Collins & Duffy 2002). Il est défini sur les arbres ordonnés étiquetés. Le mot élastique implique des interprétations flexibles des motifs en commun. La représentation vectorielle et la définition du noyau sont les mêmes que celles indiquées pour le noyau d'arbre syntaxique par les Équations (4.4) et (4.6).

Dans le cas du noyau d'arbre syntaxique, l'Équation récursive (4.7) n'est invoquée que dans le cas où les productions du nœud v_1 sont identiques à celles de v_2. Cependant, dans le cas des noyaux élastiques, les descripteurs incluent, en plus des sous-arbres généraux, des nœuds avec différentes étiquettes et des sous-structures combinant des sous-arbres avec leurs descendants. Par conséquent, il faut considérer d'autres correspondances :

Les nœuds v_1 et v_2 peuvent correspondre même s'ils n'ont pas le le même nombre de fils, mais ceci avec une seule contrainte stipulant que l'ordre de fils soit préservé. Autrement dit, si les fils i_1 et i_2 de v_1 correspondent aux fils j_1 et j_2 de v_2 et si $i_1 < i_2$ alors $j_1 < j_2$ doit être vérifiée.

Pour une certaine paire de nœuds v_1 et v_2, soit $S_{v_1,v_2}(i,j)$ la somme de produit de nombre de fois où chaque sous-arbre apparaît dans v_1 et v_2 en considérant uniquement les nœuds jusqu'à l'i^e fils de v_1 et les nœuds jusqu'au j^e fils de v_2. Ainsi, l'Équation (4.7) de C devient :

$$C(v_1, v_2) = S_{v_1,v_2}(nc(v_1), nc(v_2)).$$

Étant donné que toutes les correspondances préservent l'ordre de gauche à droite, la somme $S_{v_1,v_2}(i,j)$ peut être définie récursivement comme suit :

$$\begin{aligned} S_{v_1,v_2}(i,j) =& S_{v_1,v_2}(i-1,j) + S_{v_1,v_2}(i,j-1) - S_{v_1,v_2}(i-1,j-1) \\ &+ S_{v_1,v_2}(i-1,j-1) \cdot C(ch(v_1,i), ch(v_2,j)), \end{aligned} \tag{4.8}$$

avec $S_{v_1,v_2}(i,0) = S_{v_1,v_2}(0,j) = 1$. Par conséquent, il est possible de calculer efficacement $C(v_1, v_2)$ en utilisant la technique de la programmation dynamique.

Par ailleurs, les auteurs ont procédé aussi à une extension pour autoriser les mutations des étiquettes. Ils ont relâché la condition d'apparition des sous-arbres. Lors du comptage du nombre d'occurrences d'un sous-arbre t_i dans un arbre T, on doit pénaliser les mutations des étiquettes. Ceci est obtenu à l'aide d'une fonction de mutation $f : \Sigma \times \Sigma \to [0,1]$, où $f(l_1|l_2)$ est le coût de transformation de l'étiquette l_1 à l_2. Notons que $f(l_1|l_2) = 1$ dans le cas où $l_1 = l_2$. La Figure 4.4 montre un exemple de mutation des étiquettes, les nœuds ombragés sont interprétés comme des mutations.

On peut définir, maintenant, une fonction de similarité entre les étiquettes de deux nœuds comme suit :

$$Sim(l(v_1), l(v_2)) = \sum_{a \in \Sigma} f(l(v_1)|a) f(l(v_2)|a).$$

La contribution d'un tel coût de mutation dans le calcul de C se manifeste comme suit :

$$C(v_1, v_2) = Sim(l(v_1), l(v_2)) \cdot S_{v_1,v_2}(nc(v_1), nc(v_2)).$$

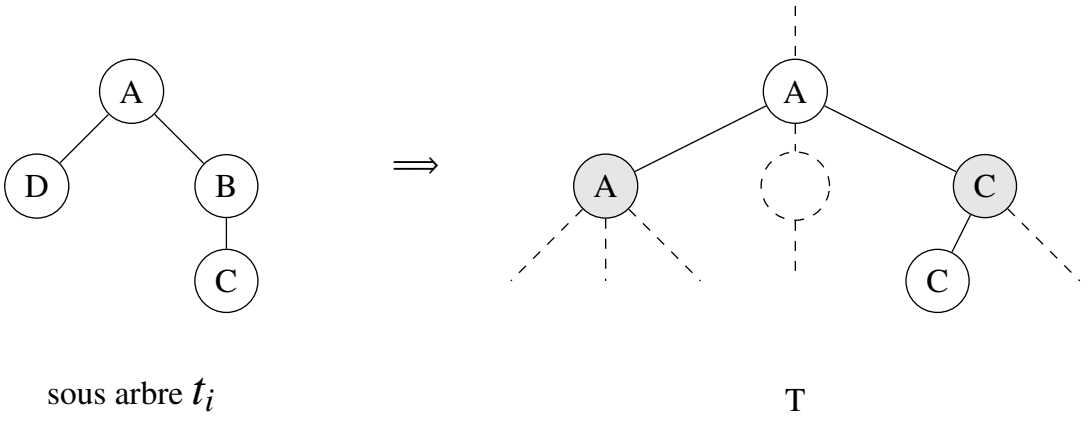


FIGURE 4.4 – Un exemple de mutation des étiquettes. Extrait de Kashima & Koyanagi 2002.

Une dernière extension est réalisée par Kashima & Koyanagi 2002. Elle consiste à autoriser des correspondances des structures élastiques. Dans ce contexte, on dit qu'un sous-arbre apparaît dans un arbre si le sous-arbre est incorporé dans l'arbre en préservant les positions relatives des nœuds du sous-arbre. La Figure 4.5 présente un exemple d'incorporation d'un sous-arbre t_i dans T.

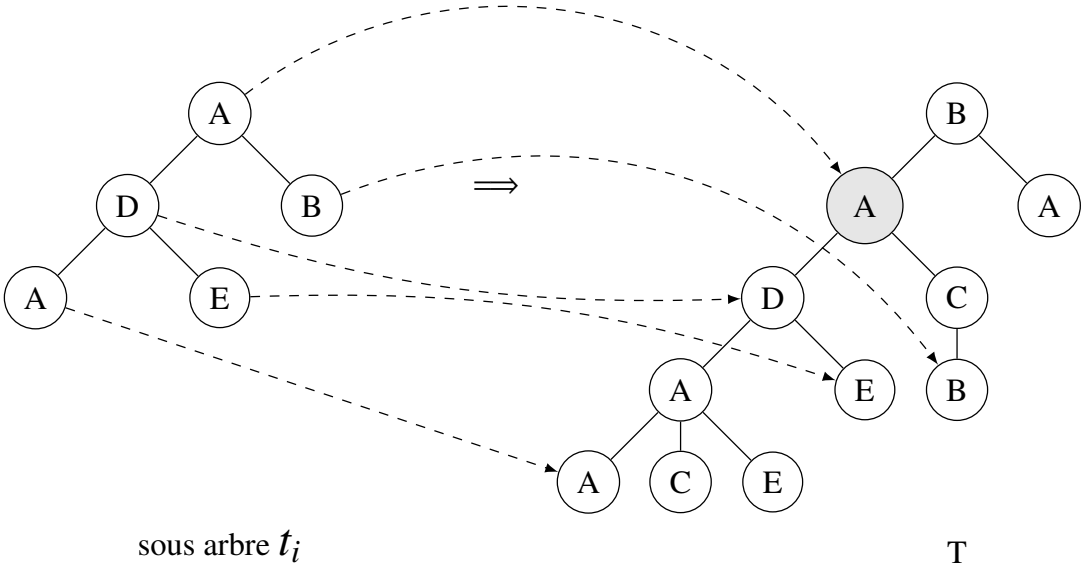


FIGURE 4.5 – Un exemple d'incorporation d'un sous-arbre t_i dans un arbre T. Extrait de Kashima & Koyanagi 2002.

Si un nœud est un descendant (fils gauche) d'un autre nœud dans le sous-arbre, cette relation doit être maintenue dans l'incorporation d'un tel sous-arbre dans l'arbre.

Étant donné que tous les descendants peuvent faire partie d'un sous-arbre élastique, chacun d'entre eux doit être considéré dans l'appariement. Pour cette raison, on doit introduire de nouvelles variables $C_a(v_1, v_2)$ définies comme suit :

$$C_a(v_1, v_2) = \sum_{v_a \in D_{v_1}} \sum_{v_b \in D_{v_2}} C(v_1, v_2),$$

où D_{v_i} est l'ensemble de nœuds descendants de v_i y compris v_i. En utilisant $C_a(v_1, v_2)$, la relation récursive (4.8) est modifiée comme suit :

$$\begin{aligned} S_{v_1,v_2}(i,j) =& S_{v_1,v_2}(i-1,j) + S_{v_1,v_2}(i,j-1) - S_{v_1,v_2}(i-1,j-1) \\ &+ S_{v_1,v_2}(i-1,j-1) \cdot C_a(ch(v_1,i), ch(v_2,j)). \end{aligned}$$

Et par conséquent, $C_a(v_1, v_2)$ peut être définie récursivement comme suit :

$$\begin{aligned} C_a(v_1, v_2) = & \sum_{j=1}^{nc(v_2)} C_a(v_1, ch(v_2, j)) + \sum_{i=1}^{nc(v_1)} C_a(ch(v_1, i), v_2) \\ & - \sum_{j=1}^{nc(v_2)} \sum_{i=1}^{nc(v_1)} C_a(ch(v_1, i), ch(v_2, j)) + C(v_1, v_2). \end{aligned}$$

Malgré que l'espace de redescription associé au noyau sous-arbre élastique est beaucoup plus grand que celui associé au noyau d'arbre syntaxique, les auteurs ont montré que la complexité des deux algorithmes est la même.

4.3.4 Noyau d'arbre partiel

Le noyau d'arbre partiel (Partial Tree kernel, PT) est proposé par Moschitti 2006 pour effectuer une correspondance partielle entre les sous-arbres. Il peut être considéré comme une extension du noyau PTK introduit dans la Section 4.3.2. Un arbre partiel est obtenu en définissant une relaxation concernant les contraintes sur les sous-arbres généraux. En effet, un arbre partiel est généré en appliquant les règles de production partielles de la grammaire. A titre d'exemple, si on fait référence à l'arbre de la Figure 4.2, les productions [GV [V]] et [GV [GN]] sont des arbres partiels valides. La Figure 4.6 montre un arbre syntaxique avec certains de ses arbres partiels.

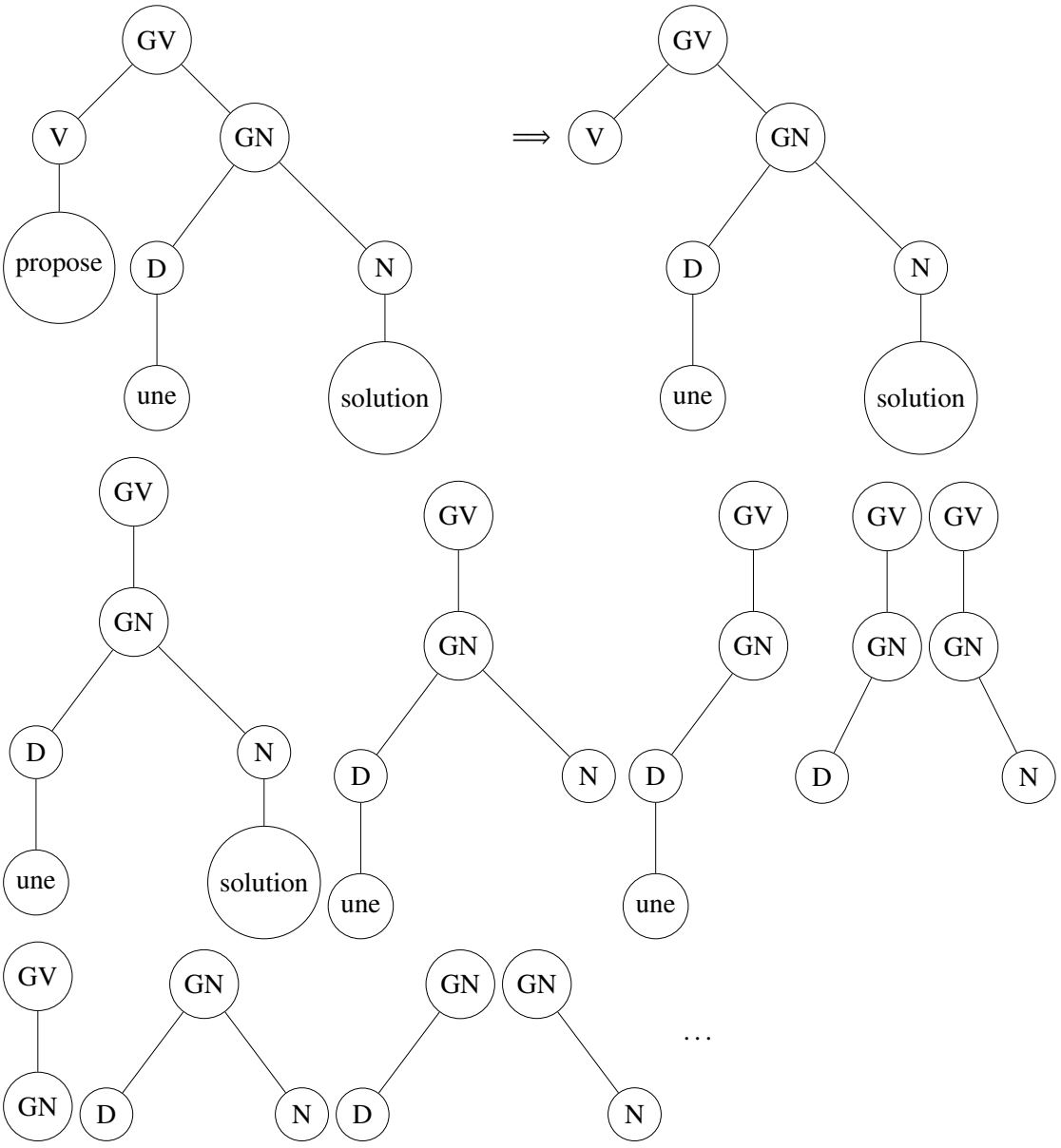


FIGURE 4.6 – Un arbre avec ses sous-arbres partiels.

Notez que l'espace de redescription du noyau PT est beaucoup plus grand que celui du noyau PTK.

La définition du noyau PT est assez similaire à celle du noyau PTK :

$$K(T_1, T_2) = \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} C(v_1, v_2),$$

avec une modification de la définition de C comme suit :

- Si les productions à v_1 et à v_2 sont différentes, alors $C(v_1, v_2) = 0$.

- Sinon

$$C(v_1, v_2) = 1 + \sum_{J_1, J_2, |J_1|=|J_2|} \prod_{i=1}^{|J_1|} C(ch(v_1, J_{1i}), ch(v_2, J_{2i})), \tag{4.9}$$

où $J_1 = (J_{11}, J_{12}, J_{13}, \ldots)$ et $J_2 = (J_{21}, J_{22}, J_{23}, \ldots)$ sont des indices des séquences associées, respectivement, aux séquences de fils ordonnés des nœuds v_1 et v_2. J_{1i} et J_{2i} pointent le i^e fils des deux séquences et $|J_1|$ renvoie la longueur de la séquence J_1.

En outre, deux facteurs de pénalisation ont été rajoutés : μ pour la hauteur de l'arbre et λ pour la longueur de la séquence des fils. Il en découle que :

$$C(v_1, v_2) = \mu(\lambda^2 + \sum_{J_1, J_2, |J_1|=|J_2|} \lambda^{d(J_1)+d(J_2)} \prod_{i=1}^{|J_1|} C(ch(v_1, J_{1i}), ch(v_2, J_{2i}))), \tag{4.10}$$

où $d(J_1) = J_{1|J_1|} - J_{11}$. De cette manière, les grands arbres et les sous-arbres construits sur des séquences de fils contenant des trous sont pénalisés.

De toute évidence, l'implémentation naïve pour évaluer l'Équation (4.10) requiert un temps exponentiel. Cependant, le recours à la programmation dynamique ramène cette complexité à $O(\rho^3|V_1||V_2|)$, où ρ est le degré sortant maximum des deux arbres. Notez que le ρ moyen dans le traitement automatique de la langue est très petit et la complexité globale peut être réduite en évitant le calcul des paires de nœuds avec des étiquettes différentes.

Les expérimentations avec la méthode de Séparateur à Vaste Marge (SVM) sur la tâche de l'étiquetage de rôles sémantiques (Semantic Role Labeling, SRL) ainsi qu'à la classification des questions montrent que le temps d'exécution du noyau est linéaire et que le noyau PT surpasse les autres noyaux d'arbres quand il est appliqué à des paradigmes d'analyses appropriés.

4.3.5 Noyau d'arbre guidé par la grammaire

Les noyaux d'arbres de convolution effectuent un appariement exact entre les sous-structures et ne considèrent pas les connaissances linguistiques lors de la conception des noyaux.

Pour surmonter ces problèmes, Zhang *et al.* 2007; Zhang *et al.* 2008 ont proposé un noyau d'arbre guidé par la grammaire pour l'étiquetage des rôles sémantiques (semantic role labeling, SRL). En effet, ils ont introduit deux améliorations. La première est un appariement approximatif guidé par la grammaire entre les sous-structures. Cette amélioration se manifeste par la faculté d'ignorer un fils d'un nœud. Pour ce faire, ils ont défini deux contraintes :

- les noeuds restants forment une règle grammaticale valide.
- la règle réduite doit contenir au moins deux fils et le premier fils correspondant à la règle originale doit être gardé.

À titre d'illustration, deux arbres syntaxiques associés aux deux phrases « il propose une solution efficace » et « il propose une solution » ne correspondent pas pour le noyau PTK. Par contre, le noyau d'arbre guidé par la grammaire leurs permet d'être appariées en réduisant la phrase « il propose une solution efficace » $\rightarrow$ « il propose une solution ».

Compte tenu de l'ensemble des règles de réduction, le mécanisme de correspondance approximative des sous-structures peut être formulé ainsi :

$$M(r_1, r_2) = \sum_{i,j} \left[T^i_{r_1}, T^j_{r_2}\right] \times \lambda_1^{a_i+b_j}, \tag{4.11}$$

où r_1 est une règle de production représentant un sous-arbre de profondeur un. $T^i_{r_1}$ est la i^e variation de r_1 en supprimant un ou plusieurs nœuds optionnels. De même pour r_2 et $T^j_{r_2}$. $\lambda_1 \in [0, 1]$ est un paramètre pour pénaliser les nœuds optionnels. Les paramètres a_i et b_j représentent respectivement le nombre d'occurrences des nœuds optionnels supprimés dans les sous-arbres $T^i_{r_1}$ et $T^j_{r_2}$. La mesure $M(r_1, r_2)$ renvoie la similarité entre les deux sous-arbres r_1 et r_2 en cumulant les similarités de toutes les variations possibles de r_1 et r_2.

La seconde amélioration permet un niveau restreint (soft) d'appariement entre les étiquettes. Deux étiquettes de nœuds différentes correspondent si elles appartiennent au même ensemble de nœuds d'équivalence. Ce dernier peut comprendre des descripteurs de nœuds qui jouent le même rôle, tels que N (solution), NS (solutions) et GN (une solution).Cette correspondance approximative des nœuds peut être formulée comme suit :

$$M(f_1, f_2) = \sum_{i,j} \left[f^i_1, f^j_2\right] \times \lambda_2^{a_i+b_j}, \tag{4.12}$$

où f_1 est un descripteur de nœud, f_1^i est la i^e mutation de f_1, a_i est égale à 0 si f_1^i et f_1 sont identiques et 1 sinon. De même pour f_2.

Étant donné les deux améliorations présentées ci-dessus, il est possible de définir un nouveau noyau dont l'espace de redescription est indexé par des sous-arbres en autorisant les règles de production originales et réduites et les mutations des descripteurs des nœuds. La composante de chaque sous-arbre représente le nombre d'occurrences de cette entrée dans l'arbre syntaxique. Le nouveau noyau est défini comme suit :

$$K(T_1, T_2) = \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} C(v_1, v_2),$$

avec une modification de la définition de C :

- Si v_1 et v_2 sont des pré-terminaux, alors

$$C(v_1, v_2) = \lambda M(f_1, f_2), \tag{4.13}$$

  où f_1 et f_2 dénotent respectivement des descripteurs des nœuds v_1 et v_2 et $M(f_1, f_2)$ est définie par l'Équation (4.12),
- sinon, si v_1 et v_2 sont des non terminaux identiques, alors générer toutes les variations des sous-arbres de profondeur un, T_{v_1} et T_{v_2}, enracinés respectivement à v_1 et v_2 en supprimant les différents nœuds optionnels. Par conséquent :

$$C(v_1, v_2) = \lambda \sum_{i,j} \left[T_{v_1}^i, T_{v_2}^j\right] \lambda_1^{a_i + b_j} \times \prod_{k=1}^{nc(v_1,i)} (1 + C(ch(v_1, i, k), ch(v_2, j, k))), \tag{4.14}$$

  où $nc(v_1, i)$ renvoie le nombre de fils de v_1 dans son i^e sous-arbre $T_{v_1}^i$, $ch(v_1, i, k)$ est le k^e fils du nœud v_1 dans le i^e sous-arbre $T_{v_1}^i$ et $\lambda \in [0, 1]$ est un facteur de pénalisation.
- sinon, $C(v_1, v_2) = 0$.

La Figure 4.7 montre la différence dans les sous-structures entre le noyau PTK (Collins & Duffy 2002) et le noyau d'arbre guidé par la grammaire.

Un problème majeur du noyau d'arbre guidé par la grammaire est sa complexité de calcul exponentielle. Zhang *et al.* 2008 ont présenté une solution efficace comparable à celle proposée pour le noyau PT (Moschitti 2006). Elle repose sur la programmation dynamique. La complexité obtenue pour l'évaluation du

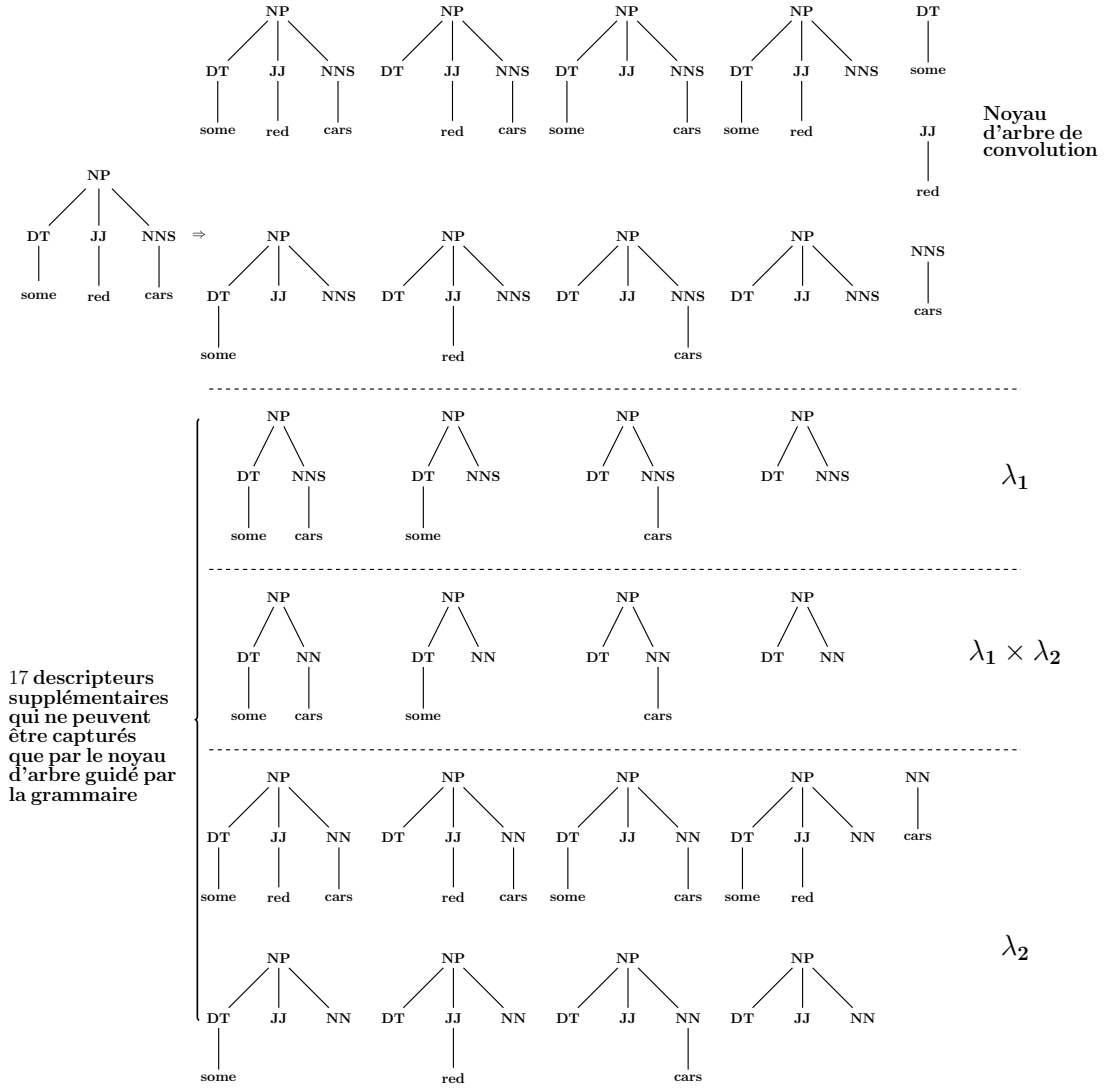


FIGURE 4.7 – Un arbre syntaxique avec ses sous-arbres du noyau d'arbre guidé par la grammaire comparés avec ceux du noyau PTK. Extrait de Zhang *et al.* 2008.

noyau entre deux arbres est $O(\rho^3|V_1||V_2|)$, où ρ est le degré sortant maximum des deux arbres.

Le noyau d'arbre guidé par la grammaire peut être considéré comme une spécialisation du noyau PT. Cependant, le noyau PT dispose d'un espace de redescription plus grand et les sous-structures sont arbitraires. Elles n'ont aucune motivation linguistique. Ainsi le bruit causé par ces sous-structures peut affecter considérablement les performances du noyau PT.

4.3.6 Noyau d'arbre syntaxique sémantique

Dans le cas des noyaux d'arbres syntaxiques, si deux arbres disposent des sous-structures représentant le même phénomène syntaxique, mais emploient une terminologie (au niveau des feuilles) différentes mais reliées, ces structures ne fournissent aucune contribution. A titre d'exemple, les deux arbres syntaxiques associés aux deux phrases "indique la pneumonie bactérienne" et "indique une infection virale" ne correspondent pas pour le noyau PTK (Collins & Duffy 2002).

Comme solution à un tel problème, Les noyaux d'arbres syntaxiques sémantiques (Semantic Syntactic Tree Kernels, SSTK) ont été proposés (Bloehdorn & Moschitti 2007; Bloehdorn 2008) pour qu'ils soient utilisés dans les tâches de classification de texte. Ces types de noyaux reposent sur le principe d'exploiter conjointement les informations syntaxiques et sémantiques. Le principe de la similarité sémantique est matérialisé par la mise en place d'une correspondance partielle entre les sous-structures. Une correspondance partielle apparaît quand deux règles de productions diffèrent seulement par leurs symboles terminaux. Par exemple, [N[pneumonie]] et [N[infection]] correspondent partiellement. Dans cette optique, un *noyau de similarité de fragments d'arbres* est défini selon un noyau k_s prédéfini :

$$K_f(f_1, f_2) = comp(f_1, f_2) \prod_{i=1}^{nt(f_1)} k_s(f_1(i), f_2(i)), \tag{4.15}$$

où $comp(f_1, f_2)$ est la fonction de compatibilité entre les deux fragments d'arbres (sous-arbres) f_1 et f_2. Elle est égale à 1 si f_1 et f_2 ne diffèrent que dans les nœuds terminaux, 0 sinon. La fonction $nt(f_1)$ représente le nombre de nœuds terminaux de f_1 et $f_1(i)$ est le i^{eme} symbole terminal de f_1 (numérotés de gauche à droite).

Le noyau SSTK peut être défini comme étant la somme de toutes les évaluations K_f sur toutes les paires des fragments d'arbres dans les arbres d'entrée :

$$\begin{aligned} K(T_1, T_2) &= \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} C(v_1, v_2) \\ avec\, C(v_1, v_2) &= \sum_{i=1}^{|F|} \sum_{j=1}^{|F|} I_i(v_1) I_j(v_2) K_f(f_i, f_j), \end{aligned} \tag{4.16}$$

où $I_i(v)$ est une fonction indicatrice qui détermine si le fragment f_i est enraciné au noeud v et $F = \{f_1, f_2, \ldots\}$ est l'ensemble de toutes les sous-structures.

Afin d'évaluer, efficacement, le noyau SSTK, il est possible d'utiliser une récursion semblable à celle de l'évaluation de C dans le cas du noyau PTK :

- Si v_1 et v_2 sont des pré-terminaux et l'étiquette de v_1 est égale à l'étiquette de v_2 alors $C(v_1, v_2) = \lambda k_s(ch(v_1, 1), ch(v_2, 1))$,
- si v_1 et v_2 ne sont pas des pré-terminaux et les règles de production de v_1 diffèrent de celles de v_2, Alors $C(v_1, v_2) = 0$,
- si v_1 et v_2 ne sont pas des pré-terminaux et les règles de production de v_1 et v_2 sont identiques, alors $C(v_1, v_2) = \prod_{j=1}^{nc(v_1)}(1 + C(ch(v_1, j), ch(v_2, j)))$.

Notez que dans le cas où v_1 et v_2 sont des pré-terminaux, ils ne peuvent avoir qu'un seul fils ($ch(v_1, 1)$ et $ch(v_2, 1)$).

En utilisant la technique de la programmation dynamique, la complexité de l'évaluation du noyau SSTK, dans le pire des cas, est $O(|V_1|, |V_2|)$, où $|V_1|$ et $|V_2|$ désignent respectivement le nombre de nœuds des arbres T_1 et T_2. Jusqu'à présent, k_s est défini comme étant une « boite noire ». Il est mis en œuvre par le bias de deux techniques. La première consiste à utiliser une taxonomie (e.g. WorldNet) pour calculer la similarité entre les termes. Tandis que la seconde utilise la technique d'indexation sémantique latente (latent semantic indexing, LSI) qui calcule la similarité en considérant l'analyse des co-occurrences des termes dans les documents et vice-versa.

4.3.7 Noyaux d'arbres approximatifs

La complexité des noyaux de convolution pour les arbres est fondamentalement quadratique en fonction du nombre de nœuds des arbres d'entrée. Ces noyaux de convolution font recours à la programmation dynamique. Une telle technique est applicable pour des arbres de petites tailles.

Dans le cas des arbres de grandes tailles, les ressources en matière de mémoire et temps d'exécution peuvent être épuisées. Pour cette raison, il est nécessaire de concevoir des noyaux efficaces avec un degré d'expressivité suffisant. Dans ce contexte, les noyaux d'arbres approximatifs ont été proposés (Rieck *et al.* 2008; Rieck *et al.* 2010), dans une perspective d'approximer le calcul

des noyaux d'arbres et permettre, ainsi, un apprentissage efficace avec des arbres de tailles arbitraires.

L'approximation est mise en place par un mécanisme de sélection d'un ensemble restreint de sous-arbres enracinés à des symboles particuliers d'une grammaire. La sélection est réalisée en utilisant une fonction de sélection $\gamma : \mathbf{S} \to \{0, 1\}$ qui contrôle si les sous-arbres enracinés à $s \in \mathbf{S}$ doivent être pris en compte dans la convolution $(\gamma(s) = 1)$ ou non $(\gamma(s) = 0)$. Ainsi, les noyaux d'arbres approximatifs sont définis :

$$\tilde{k}(T_1, T_2) = \sum_{s \in \mathbf{S}} \gamma(s) \sum_{\substack{v_1 \in V_1 \\ label(v_1)=s}} \sum_{\substack{v_2 \in V_2 \\ label(v_2)=s}} C(v_1, v_2), \tag{4.17}$$

où $\mathbf{S}$ est un ensemble de symboles terminaux et non terminaux, V_1 et V_2 désignent respectivement le nombre de nœuds des arbres T_1 et T_2.

La fonction C est définie comme suit :

- $C(v_1, v_2) = 0$, si v_1 et v_2 ne dérivent pas de la même production,
- $C(v_1, v_2) = 0$, si v_1 et v_2 ne sont pas sélectionnés, c'est à dire $\gamma(label(v_1)) = 0$ ou $\gamma(label(v_2)) = 0$,
- $C(v_1, v_2) = \lambda$, si v_1 et v_2 sont des feuilles de la même production,
- Sinon, $C(v_1, v_2) = \lambda \prod_{j=1}^{nc(v_1)} (1 + C(ch(v_1, j), ch(v_2, j)))$.

Notez que le noyau PTK (Collins & Duffy 2002) de l'Équation (4.5) peut être considéré comme un cas particulier du noyau d'arbre approximatif de l'Équation (4.17) si $\gamma(s) = 1$ pour tous les symboles $s \in \mathbf{S}$.

Pour déterminer l'intérêt du noyau d'arbre approximatif, il est possible de calculer l'alignement du noyau (Shawe-Taylor & Cristianini 2004) qui est une mesure de similarité entre une matrice noyau à évaluer et une matrice cible. La matrice yy^T est une matrice cible idéale pour les ensembles d'apprentissage équilibrés. Le vecteur y recueille toutes les classes cibles de la classification. Cet alignement de noyau est exprimé ainsi :

$$\left\langle yy^T, \tilde{K} \right\rangle_{\mathbf{F}} = \sum_{y_i = y_j} \tilde{K}_{ij} - \sum_{y_i \neq y_j} \tilde{K}_{ij},$$

où $\langle \cdot, \cdot \rangle_{\mathbf{F}}$ désigne le produit scalaire de Fiobenius et $\tilde{K}$ est la matrice noyau approximatif avec des éléments $\tilde{K}_{ij} = \tilde{k}(v_i, v_j)$.

Dans un but d'aligner le noyau $\tilde{K}$ à y, on doit maximiser le programme linéaire suivant en fonction de γ :

$$\max_{\gamma \in \{0,1\}^{|\mathbf{S}|}} \sum_{i,j=1}^{n} y_i y_j \tilde{k}(T_i, T_j) \tag{4.18}$$

où |S| est la taille de l'espace de redescription et n est la taille de l'ensemble d'apprentissage.

Pour résoudre l'Équation (4.18) efficacement, on doit borner le nombre de symboles sélectionnés par γ à N (N est le nombre des différentes sous-structures pour le calcul du noyau). Ainsi nous obtenons le programme linéaire suivant :

$$\max_{\gamma \in \{0,1\}^{|\mathbf{S}|}} \sum_{i,j=1}^{n} \mathbf{y}_i \mathbf{y}_j \sum_{s \in \mathbf{S}} \gamma(s) \sum_{\substack{v_1 \in V_1 \\ label(v_1)=s}} \sum_{\substack{v_2 \in V_2 \\ label(v_2)=s}} C(v_1, v_2),$$
$$\text{sous la contrainte } \sum_{s \in \mathbf{S}} \gamma(s) = N.$$

Par ailleurs, les auteurs ont évalué les noyaux d'arbres approximatifs en utilisant la méthode SVM comme algorithme d'apprentissage pour la classification des questions, la détection des spams et la détection des intrusions. Les expérimentations ont révélé des améliorations significatives en terme de temps d'exécution et usage de l'espace mémoire par rapport aux noyaux de convolution réguliers.

4.4 Noyaux d'arbres non ordonnés

La définition des noyaux sur les arbres non ordonnés est plus complexe par rapport au cas des arbres ordonnés. En outre, dans la littérature, il existe peu de méthodes d'application de tels noyaux. Néanmoins, nous en exhibons deux pour dévoiler leurs secrets.

4.4.1 Noyau d'arbre q-gram bi-foliaire

Pour contourner les problèmes rencontrés lors de la conception des noyaux pour les arbres non ordonnés, Kuboyama *et al.* 2008 ont développé un noyau

efficace et expressif pour les arbres non ordonnés étiquetés et enracinés.

Le noyau considère des sous-structures *q-gram bi-foliaires* comme des descripteurs. Un q-gram bi-foliaire est un arbre avec au plus deux feuilles et exactement q nœuds. Par exemple, la Figure 4.8 montre tous les 5-gram bi-foliaires possibles.

Par la suite, les auteurs ont introduit le concept de *profile de q-gram bi-foliaire*, $\Phi(T)_q$ en tant que séquences de fréquences de tous les q-gram bi-foliaires incorporés dans l'arbre T. Par conséquent, le noyau d'arbre q-gram bi-foliaire peut être évalué comme suit :

$$K_q(T_1, T_2) = \langle \Phi_q(T_1), \Phi_q(T_2) \rangle \,, \tag{4.19}$$

où T_1 et T_2 sont des arbres non ordonnés étiquetés et enracinés et q est un entier supérieur ou égale à 2. Dans le cas où $q = 1$, le noyau $K_q(T_1, T_2)$ représente le produit scalaire des fréquences des étiquettes de T_1 et T_2.

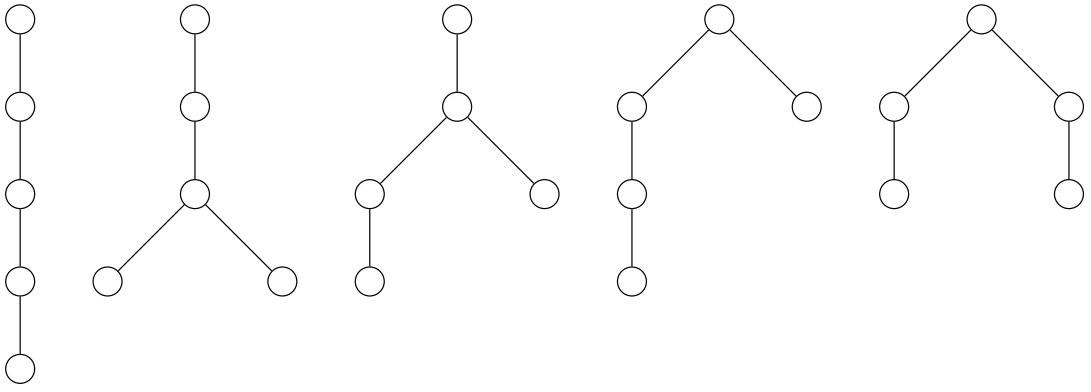


FIGURE 4.8 – Tous les 5-grams bi-foliaires. Extrait de Kuboyama *et al.* 2008.

Pour calculer efficacement le noyau q-gram bi-foliaire, les auteurs ont conçu un algorithme pour calculer le profile q-gram bi-foliaire dans un temps $O(q\, d \min(q, d)\, l\, n)$, où n est le nombre de noeuds, d est la profondeur et l est le nombre de feuilles du q-gram bi-foliaire.

Il est à noter que le noyau q-gram bi-foliaire est appliqué pour la classification des structures Glucanes en bio-informatique.

Par ailleurs, les comparaisons empiriques montrent que le noyau q-gram bi-foliaire outrepasse les noyaux d'arbres non-ordonnés existants en terme de performances prédictives. Néanmoins, les expérimentations suggèrent, aussi,

que les performances dépendent du nombre q et la valeur optimale dépend de l'ensemble de données.

4.4.2 Noyau des sous-chemins

Le noyau des sous-chemins (subpath kernel) (Kimura *et al.* 2011) est proposé pour la classification des arbres non ordonnés enracinés. Le noyau proposé utilise des sous-structures verticales appelées sous-chemins en tant que descripteurs de l'espace de redescription. En effet, les sous-chemins sont considérés comme des sous-mots de la racine aux feuilles. A titre d'illustration, la Figure 4.9 montre un arbre avec tous ses sous-chemins.

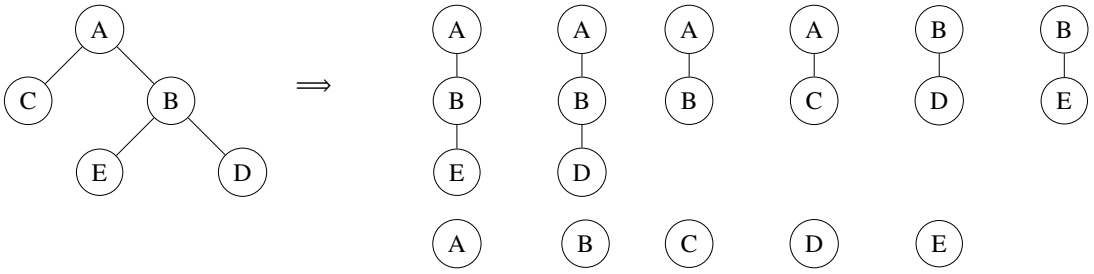


FIGURE 4.9 – Un arbre avec tous ses sous-chemins. Extrait de Kimura *et al.* 2011.

Chaque dimension du vecteur des descripteurs correspond au nombre du sous-chemin apparaissant dans l'arbre. Ainsi, le noyau des sous-chemins peut être exprimé comme suit :

$$\begin{aligned} K(T_1, T_2) &= \langle \Phi(T_1), \Phi(T_2) \rangle \\ &= \sum_{s \in S} \Phi_s(T_1) \Phi_s(T_2), \end{aligned} \tag{4.20}$$

où s est un sous-chemin, S est l'ensemble des sous-chemins de T_1 et T_2 et $\Phi_s(T_i)$ est le nombre du sous-chemin s apparaissant dans T_i.

Il est clair que l'implémentation naïve de l'Equation (4.20) nécessite d'énumérer tous les sous-chemins et construire explicitement les vecteurs des descripteurs. Ceci conduit à une complexité temporelle, dans le pire des cas, en $O((|T_1| + |T_2|)^2)$, où $|T_i|$ désigne le nombre de noeuds dans l'arbre T_i.

Pour surmonter le problème sus-cité, les auteurs ont proposé d'abord une représentation préfixielle des sous-chemins, ensuite, ils ont fait appel au tri

rapide multi-clés (Bentley & Sedgewick 1997) pour énumérer efficacement les sous-chemins. La complexité spatiale achevée est en $O((|T_1| + |T_2|))$, tandis que celle temporelle est en $O((|T_1| + |T_2|) \log(|T_1| + |T_2|))$.

Les expérimentations conduites sur des ensembles de données XML et Glucanes ont révélé que le noyau proposé est compétitif aux noyaux d'arbres existants et qu'il est plus rapide, en pratique, que les noyaux d'arbres avec un temps linéaire (e.g. le noyau sous-arbres (STK) décrit dans la Section 4.3.1).

4.5 Conclusion

Nous avons présenté, dans ce chapitre, les noyaux d'arbres. Nous avons commencé par l'introduction du paradigme de leur conception (noyaux d'application).

Ensuite, nous avons abordé notre étude selon la taxonomie arbre ordonné ou non ainsi que la variété des sous-structures (sous arbre complet, sous arbre co-enraciné, sous arbre général).

En ce qui concerne les noyaux d'arbres ordonnés impliqués par la plupart des applications des noyaux d'arbres, et au delà des noyaux sous arbres et noyaux d'arbres syntaxiques qui effectuent un appariement exact entre les sous-structures, nous avons considéré d'autres noyaux qui impliquent des interprétation flexibles des motifs commun (noyaux sous arbres élastiques, noyaux d'arbres partiels).

De plus, nous avons considéré des noyaux d'arbres qui s'intéressent aux connaissances linguistiques (noyaux d'arbres guidés par la grammaire) et aux informations sémantiques (noyaux d'arbres syntaxiques sémantiques). En fin pour des raisons d'efficacité de calcul, nous avons présenté les noyaux d'arbres approximatifs.

Pour les noyaux d'arbres non ordonnés, nous avons exhibé deux : le noyau d'arbre q-gram bi-foliaire et le noyau des sous-chemins

Deuxième partie

Contributions principales

"Perhaps the most important principle for the good algorithm designer is to refuse to be content."

Aho, Hopcroft, and Ullman, The Design and Analysis of Computer Algorithms, 1974

« *Cependant, si nous découvrons une théorie complète, elle devrait un jour être compréhensible dans ses grandes lignes par tout le monde, et non par une poignées de scientifiques.* »

Stephane Hawking, physicien théoricien et cosmologiste britannique

Chapitre 5

Approches géométriques efficaces pour le calcul du noyau SSK

NOUS avons vu, dans les chapitres précédents, que les méthodes à noyaux utilisent des espaces de redescription de hautes dimensions, voire infinies. C'est pour cette raison que l'une des clés principales des méthodes à noyaux est l'efficacité d'évaluation.

Le présent chapitre tente de répondre à cette exigence pour le noyau SSK (String Subsequence Kernel) qui est largement utilisé dans plusieurs tâches de l'apprentissage automatique.

Le chapitre est organisé comme suit : Après une brève présentation des travaux connexes, la première partie se focalise sur notre contribution à base de liste linéaire chaînée conjointement utilisée avec une structure de données géométrique (Bellaouar *et al.* 2014). La deuxième partie expose une amélioration substantielle de la première proposition soutenue par une étude expérimentale (Bellaouar *et al.* 2018).

5.1 Travaux connexes

Dans la Section 3.4.6, nous avons présenté en détail les noyaux sous-séquences de mots (SSK, String Subsequence Kernel) ainsi que trois implémentations efficaces, à savoir, les approches de la programmation dynamique, la programmation dynamique éparse et celle à base de trie. Dans la présente section, nous rappelons brièvement les propriétés de ces implémentations efficaces en se basant sur les structures de données utilisées et la complexité atteinte.

L'approche de la programmation dynamique (Lodhi *et al.* 2002), propose un algorithme (Algorithm 5) qui utilise une table de programmation dynamique dont l'analyse conduit à une complexité $O(p\,|s||t|)$. Où, p désigne la longueur des sous-séquences et $|s|$ et $|t|$ les longueurs des mots à comparer.

L'approche de la programmation dynamique éparse (Rousu & Shawe-Taylor 2005) utilise un ensemble de listes avec un arbre de somme d'intervalles (Range sum tree) pour éviter les calculs inutiles. Le temps de calcul du noyau est $O(p|L_1|\log n)$, où $|L_1|$ est la taille de la liste la plus longue et n est la longueur minimale des deux mots s et t.

En ce qui concerne l'approche à base de trie, il existe plusieurs variantes. Nous avons décrit celle présentée dans
citealpRousu2005ECG. Pour des raisons d'efficacité, les auteurs restreignent le nombre de « trous » à un entier donné g_{max}, alors le calcul est approximatif. La complexité achevée est $O(\binom{p+g_{max}}{g_{max}}(|s|+|t|))$.

5.2 Implémentation efficace à base d'une approche géométrique

Comme pour l'approche de la programmation dynamique éparse (Rousu & Shawe-Taylor 2005), notre approche (Bellaouar *et al.* 2014) essaye de contourner les inconvénients de la méthode de la programmation dynamique (Lodhi *et al.* 2002) et d'en proposer des solutions visant à améliorer la complexité du noyau SSK.

(a)

K_1^S	g	a	t	t	a
c	0	0	0	0	0
a	0	λ^2	0	0	λ^2
t	0	0	λ^2	λ^2	0
a	0	λ^2	0	0	λ^2

(b)

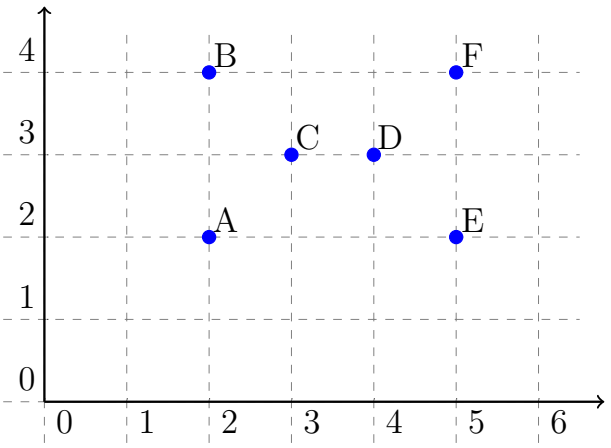


FIGURE 5.1 – Représentation de la table des suffixes (a) comme un espace 2D (b) pour les mots $s = gatta$ et $t = cata$.

Pour décrire notre approche, nous maintenons le même exemple de déroulement qui utilise deux mots s = gatta et t = cata avec le paramètre p qui désigne la longueur des sous-séquences et λ un facteur de pénalisation.

L'inconvénient majeur de l'approche de la programmation dynamique est que dans la pratique, la matrice du noyau suffixe $K_p^S(s,t)$ est éparse. Autrement dit, la plupart de ses entrées ne contribuent pas dans le résultat final du noyau. En effet, le calcul de $K_p^S(s,t)$ est conditionné par la correspondance de symboles finaux de s et de t ($s_{|s|} = t_{|t|}$). Il est judicieux donc de ne garder que la liste des paires d'indices satisfaisant cette condition plutôt que de garder la table entière :

$$L(s,t) = \{(i,j) : s_i = t_j\}.$$

Cependant, la complexité de l'implémentation naïve de la version liste est $O(p|L|^2)$, et il ne semble pas évident de calculer efficacement le noyau SSK sur une structure de données liste. Afin de remédier à ce problème, nous avons observé que la table des suffixes peut être représentée par un espace géométrique de deux dimensions où les entrées $s_i = t_j$ jouent le rôle de points dans cet espace. La Figure 5.1 illustre cette idée sur l'exemple de déroulement. Dans ce cas la liste de correspondance générée est :

$$L(s,t) = \{A, B, C, D, E, F\} = \{(2,2), (2,4), (3,3), (4,3), (5,2), (5,4)\}.$$

Techniquement parlant, le problème de calcul du noyau SSK peut être réduit en un problème de géométrie algorithmique. Dans l'optique de cette observation, le calcul de $K_p^S(s,t)$ peut être interprété comme une requête d'intervalle orthogonale. Dans la littérature, il existe plusieurs structures de données qui sont utilisées dans la géométrie algorithmique. Nous avons examiné une structure de donnée spatiale connue sous le nom de kd-tree (Bentley 1975; Berg *et al.* 2008; Samet 1990). Elle marque un coût de $O(p(|L|\sqrt{|L|}+K))$ pour calculer le noyau SSK, où K désigne le total des points rapportés. Il est claire que cette amélioration relative n'est pas assez satisfaisante. Alors, nous avons adopté une autre structure de données spatiale appelée arbre d'intervalle (Range tree) (Berg *et al.* 2008; Samet 1990; Bentley 1979; Bentley & Maurer 1980), ayant un temps de calcul pour les requêtes d'intervalles rectangulaires beaucoup mieux par rapport aux kd-trees.

Nous décrivons une telle structure et sa relation avec le calcul du noyau SSK dans ce qui suit.

5.2.1 Représentation de la table des suffixes

Les entrées (k,l) dans la liste $L(s,t)$ correspondent à un ensemble de points S dans le plan où les paires d'indice (k,l) jouent le rôle des coordonnées des points. L'ensemble S est représenté par un arbre d'intervalles 2D (2-dimensionnels), où les nœuds représentent les points dans l'espace. Ainsi, la représentation de la table des suffixes revient à construire un arbre d'intervalles 2D. Un arbre d'intervalles (Range tree), dénoté par $\mathcal{RT}$ est principalement un arbre binaire de recherche balancé (ABRB) augmenté avec une structure de données associée.

Afin de construire une telle structure, d'abord, nous considérons un ensemble S_x constitué des valeurs de la première coordonnée (coordonnée en x) de tous les points de l'ensemble S. Par la suite, un ABRB appelé x-$\mathcal{RT}$ est construit. Les points de l'ensemble S_x constituent les feuilles de l'arbre x-$\mathcal{RT}$. Chaque nœud v de x-$\mathcal{RT}$, qu'il soit interne ou feuille est augmenté par un arbre d'intervalles 1D y-$\mathcal{RT}$, il peut être un ABRB ou un tableau trié. La structure y-$\mathcal{RT}$ est alimentée par un sous ensemble canonique $P(v)$ sur les coordonnées y. Un sous ensemble canonique d'un nœud v est l'ensemble des points emmagasinés dans les feuilles d'un sous arbre enraciné au nœud v. La Figure 5.2 illustre le

processus de construction d'un arbre d'intervalles 2D pour un ensemble de points $S = \{(2,2),(5,2),(3,3),(4,3),(2,4),(5,4)\}$.

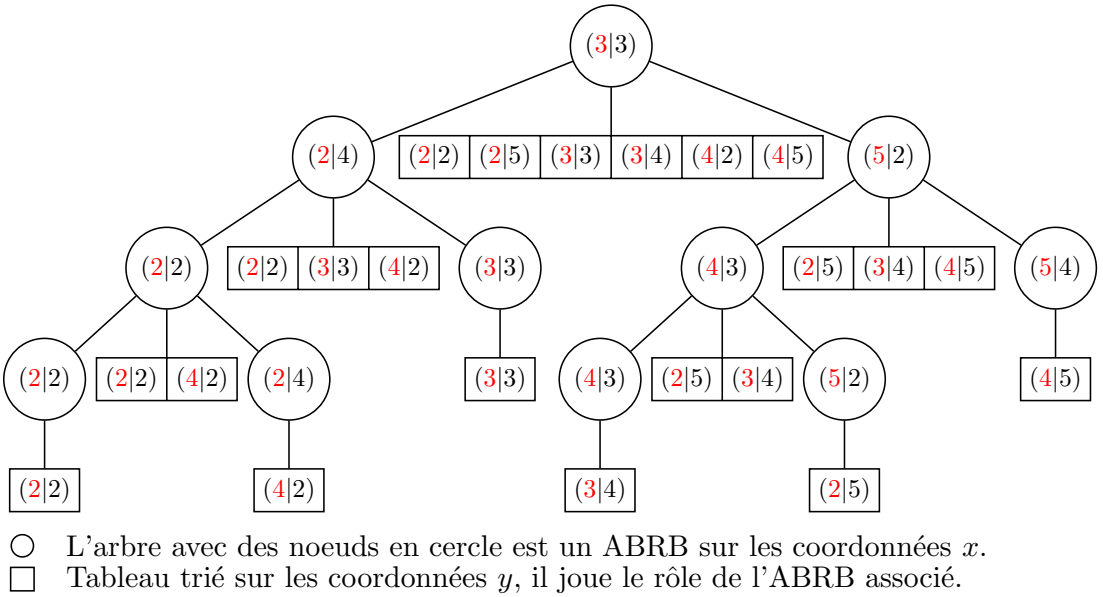


FIGURE 5.2 – Un arbre d'intervalles 2D. L'arbre principal est un arbre balancé 1D sur les coordonnées x où chaque nœud est augmenté par un arbre d'intervalles 1D sur les coordonnées y.

Dans le cas où deux points possèdent les mêmes coordonnées x ou y, nous devons définir un ordre total, tel que l'ordre lexicographique. Il consiste à remplacer les nombres réels par un espace de nombres-composés (Berg *et al.* 2008). Un nombre composé de deux réels x et y est désigné par $x|y$, de telle sorte que pour deux points, nous avons :

$$(x|y) < (x'|y') \Leftrightarrow x < x' \vee (x = x' \wedge y < y').$$

Dans une telle situation, nous devons transformer la requête d'intervalles $[x_1 : x_2] \times [y_1 : y_2]$ liée à un ensemble de points dans un plan à une requête d'intervalles $[(x_1| - \infty) : (x_2| + \infty)] \times [(y_1| - \infty) : (y_2| + \infty)]$ liée à un espace composé.

En se basant sur l'analyse de la géométrie algorithmique (Berg *et al.* 2008), notre arbre d'intervalles 2D a besoin de $O(|L| \log |L|)$ espace et peut être construit dans un temps $O(|L| \log |L|)$. Ceci conduit au lemme suivant :

Lemme 5.1. *Soient s et t deux mots et $L(s,t) = \{(i,j) : s_i = t_j\}$ la liste de correspondance associée à la version suffixe du noyau SSK. Un arbre d'intervalles pour $L(s,t)$ nécessite un espace de $O(|L| \log |L|)$ et prend un temps de construction de $O(|L| \log |L|)$.*

5.2.2 Localisation des points dans un intervalle

Nous rappelons que le calcul de la récursion de l'Équation (3.13) peut être interprété comme une évaluation d'une requête d'intervalles 2D appliquée sur un arbre d'intervalles 2D. Une telle évaluation localise tous les points qui se trouvent dans un intervalle donné.

Une idée utile en terme d'efficacité, consiste à traiter une requête d'intervalles 2D comme deux requêtes unidimensionnelles imbriquées. Autrement dit, soit $[x_1 : x_2] \times [y_1 : y_2]$ une requête d'intervalles 2D, nous commençons d'abord par la recherche des points avec des coordonnées x concernés par la requête d'intervalles $[x_1 : x_2]$. Par conséquent, nous sélectionnons $O(\log |L|)$ sous arbres. Nous considérons seulement les ensembles canoniques pour les sous arbres résultats, qui contiennent exactement les points qui se trouvent dans l'intervalle $[x_1 : x_2]$. Dans l'étape suivante, nous tenons compte seulement des points qui se trouvent dans l'intervalle$[y_1 : y_2]$.

Suite à cette description, nous pouvons conclure que la tâche complète d'une requête d'intervalles 2D peut être effectuée dans un temps $O(\log^2 |L|+k)$, où k est le nombre de points dans l'intervalle. Ce temps d'exécution peut être amélioré en renforçant l'arbre d'intervalles 2D avec la technique de la cascade fractionnaire.

5.2.3 Cascade fractionnaire

L'observation clé que nous pouvons faire lors de l'invocation de la requête d'intervalles rectangulaire est que nous devions chercher le même intervalle $[y_1 : y_2]$ dans la structure associée y-$\mathcal{RT}$ de $O(\log |L|)$ nœuds trouvés lors de l'interrogation de x-$\mathcal{RT}$ par la requête d'intervalle $[x_1 : x_2]$.

En outre, il existe une relation d'inclusion entre ces structures associées. Le but de la technique de la cascade fractionnaire consiste à exécuter une recherche binaire, uniquement, une fois et utiliser le résultat pour accélérer d'autres recherches sans pour autant accroître l'usage de l'espace mémoire de plus d'un facteur constant. L'application de la technique de la cascade fractionnaire introduite par
citealpChazelleG86 sur un arbre d'intervalles crée une nouvelle structure de données nommée arbre d'intervalles en couches (layered range tree, LRT). La

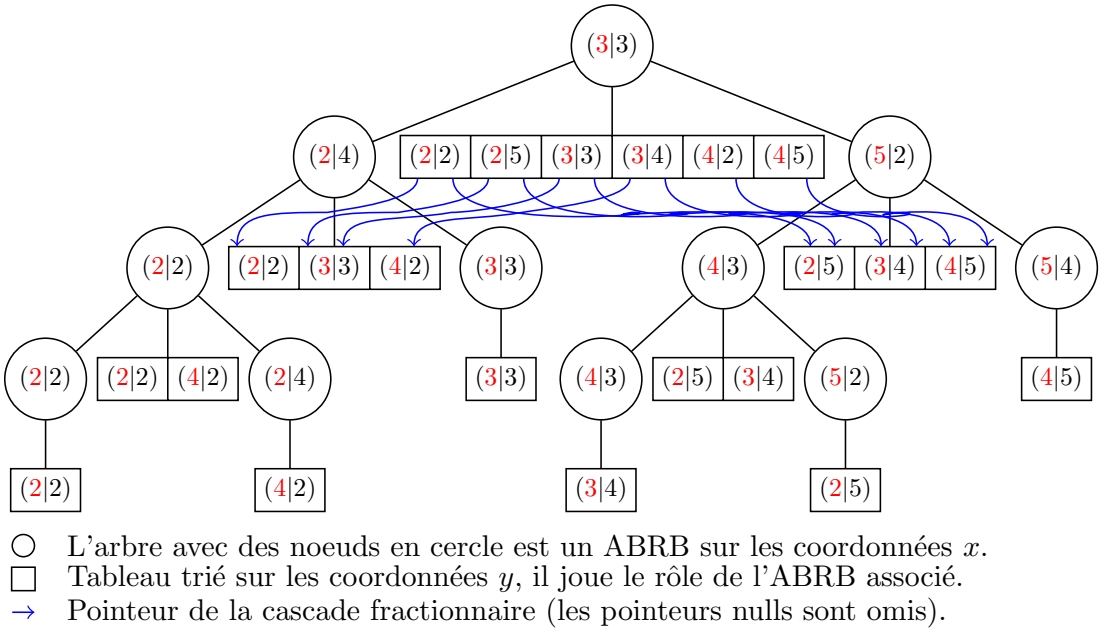


FIGURE 5.3 – Un arbre d'intervalles en couches, une illustration de la technique de cascade fractionnaire (Seulement entre deux niveaux).

Figure 5.3 illustre une telle technique à travers un exemple de calcul du noyau SSK.

En utilisant cette technique, le temps de la requête d'intervalles rectangulaire devient $O(\log |L| + k)$, où k désigne le nombre de points rapportés. Pour le calcul de $K_p^S(s,t)$, nous devons considérer $|L|$ entrées de la liste de correspondance. Pour calculer le noyau SSK, le processus s'exécute itérativement p fois. Par conséquent, nous obtenons une complexité en temps de $O(p|L| \log |L| + K)$, où K est le nombre total des points rapportés pour toutes les entrées de la liste $L(s,t)$. Ce résultat combiné avec celui du lemme 5.1 conduit au lemme suivant :

Lemme 5.2. *Soient s et t deux mots et $L(s,t) = \{(i,j) : s_i = t_j\}$ la liste de correspondance associée à la version suffixe du noyau SSK. Un arbre d'intervalles en couches (LRT, layered range tree) pour la liste $L(s,t)$ utilise un espace mémoire de $O(|L| \log |L|)$ et peut être construit dans un temps $O(|L| \log |L|)$. Avec cet arbre d'intervalles en couches, le noyau SSK de longueur p peut être évalué en $O(p(|L| \log |L| + K))$, où K est le nombre total des points rapportés pour toutes les entrées de $L(s,t)$.*

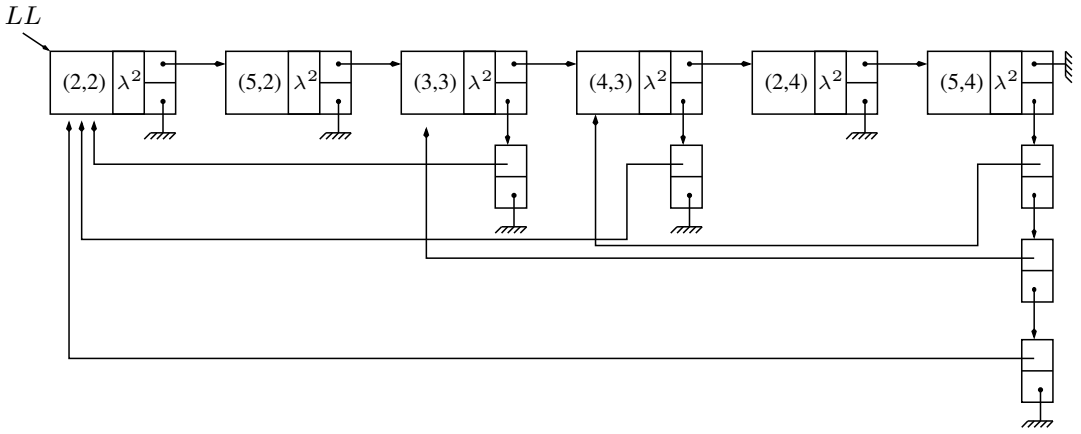


FIGURE 5.4 – Liste de listes inhérente à K_1^S(gatta,cata).

5.2.4 Construction de la liste des listes et calcul du noyau SSK

Une autre observation nous conduit à poursuivre notre chemin d'amélioration de la complexité de calcul du noyau SSK. Il est évident de constater que les coordonnées des points, dans notre cas, demeurent inchangées pendant tout le processus de calcul. Alors, au lieu d'invoquer la requête d'intervalles 2D plusieurs fois selon l'évolution du paramètre p, il est plus avantageux de ne faire le calcul qu'une seule fois pour l'utiliser ultérieurement. Par conséquent, dans cette phase, nous avons étendu notre liste de correspondance pour être une liste de listes (Figure 5.4). L'Algorithme 7 construit cette liste de listes.

Algorithm 7: Création de la liste de listes

Input: Liste de correspondance $L(s,t)$ et arbre d'intervalles en couches $\mathcal{RT}$
Output: Liste de listes $LL(s,t)$: la liste de correspondance enrichie par des listes contenant les points rapportés

1 **foreach** *entry* $(k,l) \in L(s,t)$ **do**
  `/* Préparation de la requête d'intervalles */`
2 $x_1 \leftarrow 0$
3 $y_1 \leftarrow 0$
4 $x_2 \leftarrow k-1$
5 $y_2 \leftarrow l-1$
6 relatedpoints ← 2D-RANGE-QUERY($\mathcal{RT}$, $[(x_1|-\infty) : (x_2|+\infty)] \times [(y_1|-\infty) : (y_2|+\infty)]$)
7 **while** *There exists* $(i,j) \in$ *relatedpoints* **do**
    `/* (k,l)-list est une liste dans LL(s,t) */`
8 add (i,j) to (k,l)-list

La complexité de la construction de la liste des listes est la complexité d'invocation de la requête d'intervalles 2D pour toutes les entrées de la liste de correspondance. Ceci conduit à une complexité temporelle de $O(|L| \log |L| + K)$.

Après la construction de la liste de listes, le calcul du noyau SSK doit sommer les valeurs de tous les points rapportés et enregistrés dans la liste de listes. Le processus est décrit par l'Algorithme 8. Le coût de calcul est $O(K)$, car nous devons parcourir les K points. Puisque nous allons évaluer le noyau SSK pour $p \in [1..\min(|s|, |t|)]$, ceci conduit à une complexité $O(pK)$. Alors la complexité totale du processus de calcul du noyau SSK est $O(|L| \log |L| + pK)$ qui comprend la construction de la liste des listes et le calcul proprement dit du noyau SSK. Ceci étant dit, le théorème suivant synthétise le résultat pour le calcul du noyau SSK.

Théorème 5.3. *Soient s et t deux mots et $L(s,t) = \{(i,j) : s_i = t_j\}$ la liste de correspondance associée à la version suffixe du noyau SSK. Un arbre d'intervalles en couches et une liste des listes pour $L(s,t)$ nécessitent un espace mémoire de $O(|L| \log |L| + K)$ et ils peuvent être construits dans un temps $O(|L| \log |L| + K)$. Avec ces structures de données, le noyau SSK d'une longueur p peut être évalué dans un temps $O(|L| \log |L| + pK)$, où K est le nombre total de points rapportés pour toutes les entrées de $L(s,t)$.*

5.2.5 Discussion et critique

Nous avons présenté un algorithme qui calcule efficacement le noyau SSK. Notre approche est raffinée à travers trois phases. Nous avons commencé par la construction d'une liste de correspondance $L(s,t)$ qui contient, uniquement, l'information qui contribue au résultat. Pour localiser efficacement les points liés pour chaque entrée de la liste, nous avons construit un arbre d'intervalles en couches. Finalement, nous avons construit une liste de listes pour évaluer efficacement le noyau SSK. La tâche entière nécessite un temps de calcul de $O(|L| \log |L| + pK)$ et un espace mémoire de $O(|L| \log |L| + K)$, où $|L|$ désigne la taille de la liste de correspondance, p la longueur des sous-séquences et K le nombre total des points rapportés.

D'une part, le résultat obtenu témoigne d'une amélioration de la complexité asymptotique comparée à l'implémentation naïve de la version liste, $O(p\,|L|^2)$. D'autre part, notre algorithme est sensible à la sortie. Une telle propriété nous

Algorithm 8: Calcul du noyau SSK

Input: Liste de listes $LL(s,t)$, longueur des sous-séquences p et facteur de pénalisation λ
Output: valeurs du noyau $K_q(s,t) = K(q) : q = 1, \ldots, p$

for *q=1 :p* **do**
  /* Initialisation */
  $K(q) \leftarrow 0$
  /* Initialisation */
  $KPS(1 : |max|) \leftarrow 0$
  foreach *entry* $(k,l) \in LL(s,t)$ **do**
    foreach *entry* $r \in (k,l)$-$list$ **do**
      /* (k,l)–$list$ est une liste associée à l'entrée (k,l) */
      $(i,j) \leftarrow r.Key$
      $KPS(i,j) \leftarrow r.Value$
      $KPS(k,l) \leftarrow KPS(k,l) + \lambda^{k-i+l-j}\, KPS(i,j)$
    $K(q) \leftarrow K(q) + KPS(k,l)$)
  /* Préparation de $LL(s,t)$ pour le prochain calcul */
  foreach *entry* $(k,l) \in KPS$ **do**
    Update $LL(k,l)$ with $KPS(k,l)$

dicte de procéder à une analyse empirique. C'est ce que nous avons fait, mais après certaines améliorations de la présente contribution (voir Section 5.3).

Néanmoins, sur la base des complexités asymptotiques des différentes approches et les expérimentations conduites dans citealpRousu2005ECG, nous pouvons mentionner quelques discussions. L'approche de la programmation dynamique est rapide quand la table de la programmation dynamique DP_p est dense. Ce cas est atteint pour des documents longs sur un alphabet de petite taille ou même sur des documents courts. L'approche à base de trie est rapide pour des alphabets de taille moyenne, mais la restriction des « trous » reste un petit handicap. Dans une telle situation, d'une part, nous pensons que cette restriction peut influencer la précision du noyau SSK, d'autre part, il semble que l'algorithme à base de trie avec des restrictions fortes devient plus proche au noyau p-spectre qu'au noyau SSK.

Par ailleurs, nous rappelons que notre approche et celle de la programmation dynamique éparse sont proposées dans le contexte où la plupart des entrées de la table DP_p sont nulles. Ce cas aura lieu pour des alphabets de grandes taille. En se basant sur la complexité de l'approche éparse, $O(p|L| \log \min(|s|, |t|))$, il est clair

que son efficacité dépend de la taille des mots à comparer. Pour notre approche, elle dépend seulement de la taille de la liste de correspondance, autrement dit, du nombre de sous-séquences en commun des deux mots à comparer. Sous ces conditions, notre approche surpasse pour les mots de taille importante.

Un avantage remarquable pour notre approche est qu'elle sépare le processus de localisation des données de celui de calcul proprement dit. Cette séparation limite l'influence de la longueur des sous-séquences sur l'évaluation du noyau. L'impact demeure, uniquement, sur le processus de calcul.

En outre une telle séparation peut être favorable si l'on suppose que le problème est multidimensionnel, par exemple, le problème de comparaison de mots multiples (*the multiple string comparison*), l'un des domaines de recherche les plus actifs dans l'analyse des séquences biologiques. Notre approche ouvre une nouvelle vision pour aborder ce type de problèmes. En terme de complexité, ceci peut influencer, seulement, le processus de localisation par un facteur logarithmique. En effet, un arbre d'intervalles en couches peut rapporter des points se situant dans un intervalle rectangulaire en un temps $O(\log^{d-1}|L| + k)$, dans un espace d-dimensionnel.

Ceci étant dit, la complexité de notre approche dépend du paramètre K désignant le nombre de points rapportés. C'est un paramètre à la sortie, il n'est connu, pour un cas donné, qu'après l'exécution du programme de calcul du noyau SSK. Notre objectif vise à outrepasser ce paramètre. C'est bien l'objet de la contribution décrite dans la section qui suit.

5.3 Amélioration de l'approche géométrique

Dans une perspective de proposer une méthode efficace de calcul du noyau SSK, nous avons proposé dans
citealpBellaouarlata2014 une approche que nous avons qualifié de géométrique (Section 5.2). Dans cette approche, nous avons utilisé un arbre d'intervalles en couches (Layered Range Tree, LRT) conjointement avec une liste de listes. Mais la structure de données LRT rapporte tous les points qui se trouvent dans un intervalle donné qui seront stockés, par la suite, dans la liste de listes (Le problème du paramètre K que nous avons évoqué à la fin de la Section 5.2.5).

Par conséquence, le noyau SSK de longueur p peut être évalué en un temps $O(|L|\log|L| + pK)$ et nécessite un espace de stockage $O(|L|\log|L| + K)$, où K désigne le total des points rapportés sur toutes les entrées de la liste de correspondance $L(s,t)$.

Cependant, pour le calcul du noyau SSK nous aurons besoin, seulement, de la somme des valeurs des points rapportés. De plus, dans le pire des cas, le paramètre K est $O(|L|^2)$ et $|L|$ est $O(|s||t|)$. Il est claire qu'il serait très utile, pour le calcul du SSK, de pouvoir calculer la somme des valeurs de points qui se trouvent dans une plage spécifiée sans parcourir ni stocker cette collection de points. Pour atteindre cet objectif, nous proposons une nouvelle approche géométrique qui s'appuie sur notre travail précédent (Bellaouar *et al.* 2014) en étendant la structure LRT avec les opérations d'agrégation, en particulier l'opération somme.

5.3.1 Extension de l'arbre d'intervalles en couches

La présente section fait état de notre démarche d'extension de la structure de données LRT avec les opérations d'agrégation. Ainsi, une nouvelle structure de données est crée, baptisée arbre de somme d'intervalles en couches (Layered Range Sum Tree, LRST).

Un LRST est un LRT avec deux extensions substantielles, à savoir, la *somme partielle* et la *cascade fractionnaire double* pour réduire le temps d'exécution d'une requête somme d'intervalles de $O(\log|L|+k)$ à $O(\log|L|)$, où k désigne le nombre de points rapportés dans un intervalle.

Le but de l'extension somme partielle est de pouvoir calculer la somme des valeurs dans une plage dans un temps constant. Pour cette raison, nous substituons les structures de données associées $\mathcal{T}_{assoc}$ (où chaque entrée i stocke une paire clé-valeur (y_i, v_i)) dans la structure LRT par de nouvelles structures de données associées $\mathcal{T}'_{assoc}$, où chaque entrée i contient une paire clé-valeur (y_i, ps_i) où $ps_i = \sum_{k=1}^{i} v_k$ est la somme partielle sur les valeurs de $\mathcal{T}_{assoc}$ à la position i. Cette extension est faite pour calculer la somme d'intervalle, dans le cas général, de $\mathcal{T}_{assoc}$ dans $[i,j]$ dans un temps $O(1)$. Ceci se justifie par une

	1	2	3	4	5	6
$\mathcal{T}_{assoc}$	$(2\|2, \lambda^{-2})$	$(2\|5, \lambda^{-5})$	$(3\|3, \lambda^{-4})$	$(3\|4, \lambda^{-5})$	$(4\|2, \lambda^{-4})$	$(4\|5, \lambda^{-7})$

	1	2	3	4	5	6
$\mathcal{T}'_{assoc}$	$(2\|2, \lambda^{-2})$	$(2\|5, \lambda^{-2} + \lambda^{-5})$	$(3\|3, \lambda^{-2} + \lambda^{-4} + \lambda^{-5})$	$(3\|4, \lambda^{-2} + \lambda^{-4} + 2\lambda^{-5})$	$(4\|2, \lambda^{-2} + 2\lambda^{-4} + 2\lambda^{-5})$	$(4\|5, \lambda^{-2} + 2\lambda^{-4} + 2\lambda^{-5} + \lambda^{-7})$

FIGURE 5.5 – Extension d'une structure de donnée associée $\mathcal{T}_{assoc}$ dans LRT avec les sommes partielles.

simple opération de soustraction dans la structure $\mathcal{T}'_{assoc}$ comme suit :

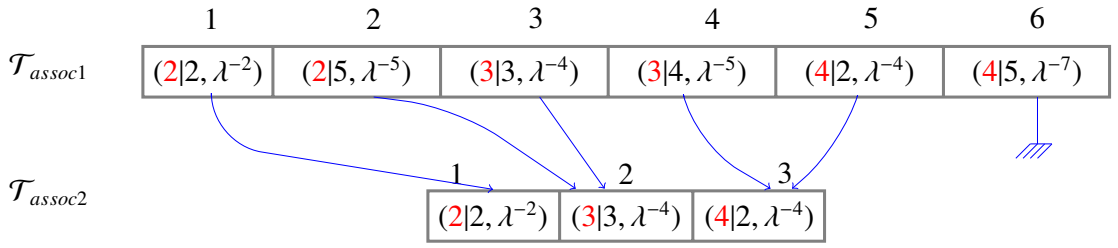
$$\text{Rangesum}([\text{i}, \text{j}]) = ps_j - ps_{i-1}.$$

En se basant sur l'exemple de déroulement, la Figure 5.5 illustre le mécanisme de cette extension. Pour cet exemple, $\text{Rangesum}([2, 5]) = ps_5 - ps_1 = \lambda^{-2} + 2\lambda^{-4} + 2\lambda^{-5} - \lambda^{-2} = 2\lambda^{-4} + 2\lambda^{-5}$ peut être calculée dans un temps constant.

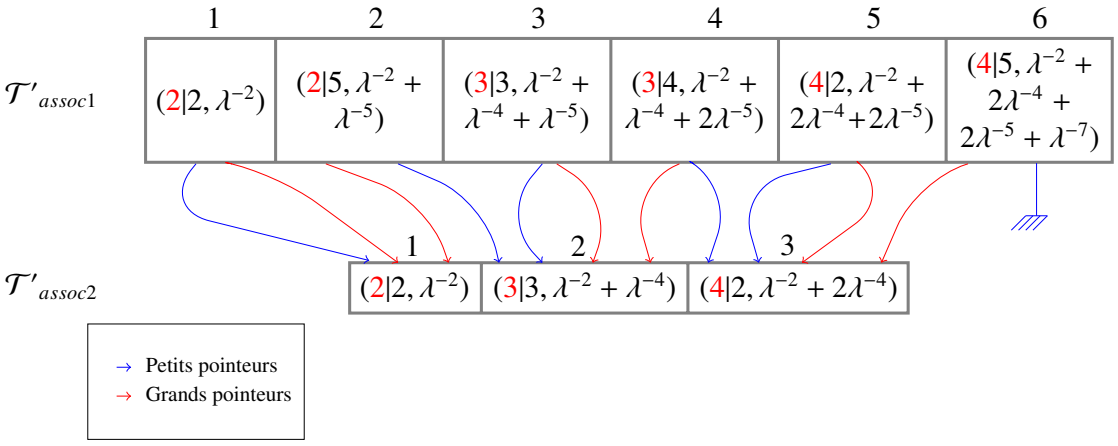
La deuxième extension implique la technique de la cascade fractionnaire. Ce réarrangement consiste à utiliser deux pointeurs pour effectuer deux recherches binaires et une opération d'agrégation au lieu d'un pointeur avec une recherche binaire et un parcours des objets à rapporter dans la technique de la cascade fractionnaire classique.

Pour illustrer un tel mécanisme, soient $\mathcal{T}'_{assoc1}$ et $\mathcal{T}'_{assoc2}$ deux tableaux triés en cascade qui sauvegardent les somme partielles de $\mathcal{T}_{assoc1}$ et $\mathcal{T}_{assoc2}$ respectivement. Le tableau $\mathcal{T}_{assoc1}$ est une structure associée au nœud v de LRT et $\mathcal{T}_{assoc2}$ est associée au fils gauche (ou fils droit) du nœud v. En outre, les éléments de $\mathcal{T}_{assoc2}$ constituent un sous-ensemble de ceux de $\mathcal{T}_{assoc1}$.

Supposons que nous voulons calculer la somme d'intervalle avec une requête $q = [y_1, y_2]$ dans $\mathcal{T}_{assoc1}$ et $\mathcal{T}_{assoc2}$. Nous commençons par une recherche binaire avec y_1 dans $\mathcal{T}'_{assoc1}$ pour trouver la plus petite clé supérieure ou égale à y_1. Nous effectuons, encore, une autre recherche binaire avec y_2 dans $\mathcal{T}'_{assoc1}$ pour trouver la plus grande clé inférieure ou égale à y_2. Si une entrée $\mathcal{T}'_{assoc1}[i]$ emmagasine une clé y_i, alors nous sauvegardons un pointeur vers l'entrée dans $\mathcal{T}'_{assoc2}$ avec la plus petite clé supérieure ou égale à y_i, dit *petit pointeur*, et un second pointeur à l'entrée dans $\mathcal{T}'_{assoc2}$ avec la plus grande clè inférieure ou égale à y_i, dit *grand pointeur*. S'il n'existe pas de telle(s) clé(s), alors le(s) pointeur(s) est (sont) mis à



(a) Cascade Fractionnaire avant extension.



(b) Cascade Fractionnaire après extension.

FIGURE 5.6 – La technique de la cascade fractionnaire double (b) : une extension de la cascade fractionnaire classique (a) avec grands pointeurs.

nil. La technique de la cascade fractionnaire étendue est illustrée par le biais de la Figure 5.6.

Afin de montrer l'impact de ces deux extensions sur le temps et l'espace de la requête somme d'intervalle, nous présentons un exemple qui implique l'ensemble de points utilisé dans la Figure 5.6 et une requête d'intervalle $q = [2|5, 4|5]$.

Occupons nous d'abord par le calcul de la requête somme d'intervalle dans $\mathcal{T}_{assoc1}$ et $\mathcal{T}_{assoc2}$ (avant extensions). Nous commençons par une recherche binaire avec $2|5$ dans $\mathcal{T}_{assoc1}$ et parcourons $\mathcal{T}_{assoc1}$ vers la droite jusqu'à ce qu'une clé plus grande que $4|5$ soit rencontrée. Ainsi, nous rapportons et stockons l'ensemble de points suivant : $\{(2|5, \lambda^{-5}), (3|3, \lambda^{-4}), (3|4, \lambda^{-5}), (4|2, \lambda^{-4}), (4|5, \lambda^{-7})\}$. Pour rapporter les points de $\mathcal{T}_{assoc2}$, nous devons suivre le pointeur de $\mathcal{T}_{assoc1}[2]$ qui pointe vers $\mathcal{T}_{assoc2}[2]$. *À* partir de cet emplacement nous commençons à parcourir $\mathcal{T}_{assoc2}$ vers la droite. L'ensemble des points $\{(3|3, \lambda^{-4}), (4|2, \lambda^{-4})\}$ constitue les points rapportés et stockés à partir de $\mathcal{T}_{assoc2}$. Par la suite, nous devons parcourir tous

les points rapportés pour calculer la somme d'intervalle inhérente à la requête d'intervalle q. Il est clair que pour le calcul de la requête somme d'intervalle, nous aurons besoins d'un temps $O(\log|\mathcal{T}_{assoc1}|+k)$ et d'un espace mémoire (au delà de l'espace de LRT) de $O(k)$, où k est le nombre de points rapportés.

Passons maintenant au calcul de la requête somme d'intervalle en considérant nos extensions de LRT. Nous commençons par deux recherches binaires avec $2|5$ et $4|5$ dans $\mathcal{T}'_{assoc1}$. Les recherches binaires conduisent directement au calcul de $\mathcal{T}'_{assoc1}.\text{Rangesum}([2,6]) = ps_6 - ps_1 = 2\lambda^{-4}+2\lambda^{-5}+\lambda^{-7}$ dans un temps $O(\log|\mathcal{T}_{assoc1}|)$. Pour calculer la requête d'intervalle dans $\mathcal{T}'_{assoc2}$, nous évitons les recherches binaires. Nous suivons le petit pointeur de $\mathcal{T}'_{assoc1}[2]$ dans $\mathcal{T}'_{assoc2}[2]$ et le grand pointeur de $\mathcal{T}'_{assoc1}[6]$ dans $\mathcal{T}'_{assoc2}[3]$. Aussi, le calcul de $\mathcal{T}'_{assoc2}.\text{Rangesum}([1,3]) = ps_3 - ps_1 = 2\lambda^{-4}$ prend un temps constant. Il est clair que le temps d'exécution de la requête somme d'intervalle dans $\mathcal{T}'_{assoc1}$ et $\mathcal{T}'_{assoc2}$ est $O(\log|\mathcal{T}_{assoc1}|)$. En conséquence, nous pouvons imaginer l'impact de ces deux extensions (somme partielle et technique de la cascade fractionnaire double) sur le temps et l'espace de la requête somme d'intervalle, en particulier pour les problèmes à grande échelle.

Il n'est pas difficile de voir que nos extensions n'affectent ni la complexité spatiale ni temporelle de la construction de la structure LRT. Ainsi, en se basant sur l'analyse de la géométrie algorithmique, notre structure LRST nécessite $O(|L|\log|L|)$ espace mémoire et peut être construite dans un temps $O(|L|\log|L|)$. Ceci conduit au lemme suivant :

Lemme 5.4. *Soient s et t deux mots et $L(s,t)=\{(i,j) : s_i = t_j\}$ la liste de correspondance associée à la version suffixe du noyau SSK. Un arbre d'intervalles de somme en couches (layered range sum tree, LRST) pour $L(s,t)$ nécessite $O(|L|\log|L|)$ espace mémoire et prend un temps de construction de $O(|L|\log|L|)$.*

5.3.2 Calcul du noyau sous-séquences de mots (SSK)

Dans la Section 5.3.1, nous avons illustré et montré l'effet de nos extensions de LRT dans une somme d'intervalles appliquée à deux structures associées en cascade. Exploitons maintenant le LRST (LRT étendu) pour calculer efficacement le SSK. Ceci est concrétisé par (Algorithm 9).

Tout d'abord, nous construisons la liste de correspondances $L(s,t) = \{((i,j), \widetilde{K_p^S}(s(1:i), t(1:j))) : s_i = t_j\}$ où $\widetilde{K_p^S}(s(1:i), t(1:j)) = \lambda^{-i-j} K_p^S(s(1:i), t(1:j))$. Ainsi la récursion de l'Équation (3.13) devient :

$$\widetilde{K_p^S}(sa, tb) \quad = \quad [a=b] \sum_{i \leq |s|} \sum_{j \leq |t|} \lambda^{i+j} \widetilde{K_{p-1}^S}(s(1:i), t(1:j)). \tag{5.1}$$

Algorithm 9: Évaluation du noyau SSK par l'approche géométrique.

Input: Les mots s et t, la longueur des sous-séquences p et le facteur de pénalisation λ.
Output: Valeurs du noyau $K_q(s,t) = K(q) : q = 1, \ldots, p$.

```
1  m ← length(s)
2  n ← length(t)
3  Création de la liste de correspondance initiale L
   /* Évaluation de K_1(s,t)                                   */
4  foreach ((i,j), v) ∈ L do
5      K[1] ← K[1] + v · λ^{i+j}
   /* Calcul de K_q(s,t) : q = 2, ..., p                       */
6  for q = 2 :p do
7      Construction de la structure LRST associée à la liste de correspondance L
8      foreach ((i,j), v) ∈ L do
           /* Préparation de la requête d'intervalles pour
              l'entrée (i,j)                                   */
9          rq ← [(0| − ∞) : (i − 1| + ∞)] × [(0| − ∞) : (j − 1| + ∞)]
10         result ← Rangesum(rq)
11         if result > 0 then
12             K[q] = K[q] + result · λ^{i+j}
13             appendlist(newL, ((i,j), result))
14     L ← newL
```

Afin de construire, efficacement, la liste de correspondance, nous avons exploité l'idée de
citealpRousu2005ECG. Elle consiste à créer pour chaque symbole $c \in \Sigma$ une liste $I(c)$ des positions des occurrences $(c = s_i)$ dans le mot s. Par la suite, pour chaque symbole $t_j \in t$, nous insérons une paire clé-valeur $((i,j), \widetilde{K_p^S}(s(1:i), t(1:j)))$ dans la liste $L(s,t)$ correspondant à $I(t_j)$. Ce processus prend $O(|s|+|t|+|\Sigma|+|L|)$ espace mémoire et un temps d'exécution de $O(|s|+|t|+|L|)$.

À titre d'exemple, la liste de correspondance pour notre exemple de déroulement se présente comme suit :

$$L(s,t) = \{((2,2),\lambda^{-2}),((2,4),\lambda^{-4}),((3,3),\lambda^{-4}),((4,3),\lambda^{-5}),((5,2),\lambda^{-5}),((5,4),\lambda^{-7})\}. \tag{5.2}$$

Après la création de la liste de correspondance initiale, nous commençons par le calcul du noyau SSK pour les sous-séquences de longueur $p = 1$. Ce calcul n'a pas besoins de la structure LRST. Il suffit de parcourir la liste de correspondance et effectuer la somme de ses valeurs.

Pour les sous-séquences de longueur $p > 1$, nous devons, d'abord, créer la structure LRST associée à la liste de correspondance. Par la suite, pour chaque élément $((k,l),\widetilde{K_p^S}(s(1:k),t(1:l)))$, nous invoquons la structure LRST par la requête $rq = [0,k-1]\times[0,l-1]$. En réponse à cette requête, la structure LRST renvoie la somme d'intervalles :

$$\text{Rangesum(rq)} = \sum_{i<k}\sum_{j<l}(\widetilde{K_p^S}(s(1:i),t(1:j))).$$

Si Rangesum(rq) est positive, alors la paire clé-valeur est insérée dans une nouvelle liste de correspondance pour le niveau $q+1$ et cumuler le résultat Rangesum(rq) pour calculer le noyau SSK au niveau q (lignes 12-13 de l'Algorithme 9). À chaque itération, nous devons créer une nouvelle structure LRST associée à la nouvelle liste de correspondance jusqu'à ce qu'on atteint la longueur des sous-séquences p demandée.

Nous rappelons que dans notre cas, nous avons utilisé des nombres composés au lieu des nombres réels, voir Section 5.2.1. Dans une telle situation, nous devons transformer la requête d'intervalles $[x_1 : x_2]\times[y_1 : y_2]$ liée à l'ensemble des points dans l'espace à une requête d'intervalles $[(x_1|-\infty) : (x_2|+\infty)]\times[(y_1|-\infty) : (y_2|+\infty)]$ liée à l'espace composé.

En utilisant notre approche géométrique, le temps d'exécution de la requête des sommes d'intervalles devient $O(\log|L|)$. Pour l'évaluation de $K_p^S(s,t)$, il est nécessaire de considérer $|L|$ entrées de la liste de correspondance. Ce processus est itéré p fois, ainsi, nous obtenons une complexité temporelle de $O(p|L|\log|L|)$ pour le calcul du noyau SSK. Ce résultat combiné à celui du lemme 5.4 conduit au théorème 5.5 qui synthétise le résultat de notre approche proposée pour calculer le noyau SSK :

Théorème 5.5. *Soient s et t deux mots et $L(s,t) = \{(i,j) : s_i = t_j\}$ la liste de correspondance associée à la version suffixe du noyau SSK. En utilisant l'arbre d'intervalles de somme en couches (layered range sum tree, LRST), le noyau SSK de longueur p peut être évalué dans un temps $O(p|L|\log|L|)$.*

À titre d'illustration, calculons $K_2(s,t)$, pour l'exemple de déroulement. Nous devons invoquer la requête de somme d'intervalles sur la structure LRST représentée par la Figure 5.7 (pour p=2). La Table 5.1 montre les requêtes d'intervalles correspondantes aux entrées de la liste de correspondance de l'Équation 5.2.

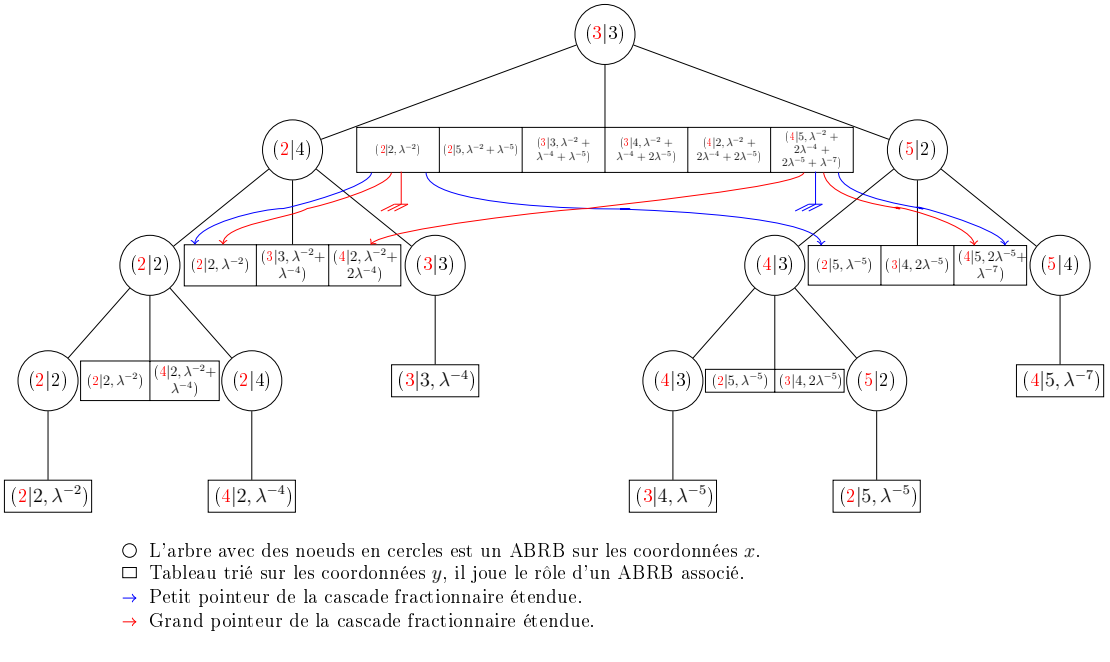


FIGURE 5.7 – État de LRST pour l'exemple de déroulement à l'étape $p = 2$ avec une illustration de la cascade fractionnaire étendue (seulement entre deux niveaux, une partie des pointeurs est montrée).

L'évaluation du noyau SSK est réalisé par le cumul de toutes les sommes d'intervalles associées aux entrées de la liste de correspondance comme suit :

$$K_2(s,t) = \sum_{i=1}^{6} \text{Rangesum}(\text{rq}_\text{i}).$$

Pour décrire la façon dont cela peut être traité, nous nous occupons, par exemple, de la somme d'intervalles de la requête rq_6. Dans la structure de données associée au nœud de division $(3|3)$ de la Figure 5.7, nous trouvons les entrées $(2|2)$ et $(3|4)$ dont les coordonnées y sont respectivement la plus petite supérieure ou égale à $(0|-\infty)$ et la plus grande inférieure ou égale à $(3|+\infty)$. Ceci peut être

TABLE 5.1 – Requêtes d'intervalles, dans l'espace composé, associées à l'exemple de déroulement.

Entrée liste correspondance	**Nom requête d'intervalle**	**Requête d'intervalle dans l'espace composé**
$(2,2)$	rq_1	$[(0\|-\infty) : (1\|+\infty)] \times [(0\|-\infty) : (1\|+\infty)]$
$(2,4)$	rq_2	$[(0\|-\infty) : (1\|+\infty)] \times [(0\|-\infty) : (3\|+\infty)]$
$(3,3)$	rq_3	$[(0\|-\infty) : (2\|+\infty)] \times [(0\|-\infty) : (2\|+\infty)]$
$(4,3)$	rq_4	$[(0\|-\infty) : (3\|+\infty)] \times [(0\|-\infty) : (2\|+\infty)]$
$(5,2)$	rq_5	$[(0\|-\infty) : (4\|+\infty)] \times [(0\|-\infty) : (1\|+\infty)]$
$(5,4)$	rq_6	$[(0\|-\infty) : (4\|+\infty)] \times [(0\|-\infty) : (3\|+\infty)]$

réalisé par une recherche binaire. Ensuite, nous cherchons les nœuds qui sont au-dessous du nœud de division $(3|3)$ et qui sont le fils droit d'un nœud dans le chemin de recherche vers $(0|-\infty)$ où le chemin va à gauche, ou bien le fils gauche d'un nœud dans le chemin de recherche vers $(4|+\infty)$ où le chemin va à droite. Les nœuds collectés sont $(3|3), (2|2), (4|3)$ et le résultat renvoyé à partir de la structure associée est $\lambda^{-5}+\lambda^{-4}+\lambda^{-2}$. Ceci est effectué dans un temps constant en suivant le petit et grand pointeurs à partir de la structure associée du nœud de division. Avec le même mécanisme, nous obtenons les résultats des requêtes de sommes d'intervalles suivants :

$$\begin{aligned}
\mathrm{Rangesum}(\mathrm{rq}_1) &= 0 \\
\mathrm{Rangesum}(\mathrm{rq}_2) &= 0 \\
\mathrm{Rangesum}(\mathrm{rq}_3) &= \lambda^{-2} \\
\mathrm{Rangesum}(\mathrm{rq}_4) &= \lambda^{-2} \\
\mathrm{Rangesum}(\mathrm{rq}_5) &= 0
\end{aligned}$$

Après la remise en échelle des valeurs retournées par un facteur λ^{i+j}, nous obtenons la valeur de $K_2(s,t) = \lambda^{-2}\cdot\lambda^{3+3} + \lambda^{-2}\cdot\lambda^{4+3} + (\lambda^{-5}+\lambda^{-4}+\lambda^{-2})\cdot\lambda^{5+4} = 2\lambda^4 + 2\lambda^5 + \lambda^7$. Tout en invoquant les requêtes de sommes d'intervalles, nous préparons la nouvelle liste de correspondance pour la prochaine étape. Dans notre cas, la nouvelle liste contient les correspondances suivantes :

$$L(s,t) = \{((3,3),\lambda^{-2}), ((4,3),\lambda^{-2}), ((5,2),\lambda^{-5}+\lambda^{-4}+\lambda^{-2})\}.$$

5.3.3 Expérimentations

Dans cette section, nous décrivons les expérimentations qui se focalisent sur l'évaluation de notre approche géométrique (à base de LRST) contre les approches de la programmation dynamique et celle de la programmation dynamique éparse. La métrique utilisée pour évaluer l'efficacité des différentes approches est le temps d'exécution. Par la suite, ces approches sont référencées respectivement par les termes Géométrique, Dynamique et Éparse.

Nous avons écarté l'approche à base de Trie de cette comparaison car c'est une approche d'approximation. En effet, Nous avons opté pour le choix qui donne des opportunités égales aux différents algorithmes à comparer. Nous n'avons donc gardé que les algorithmes exacts (Dynamique, Éparse et Géométrique).

Pour faire des comparaisons similaires à celles menées par citealpRousu2005ECG, nous essayons de garder les mêmes conditions de leurs expérimentations. Pour cette raison, nous conduisons une série d'expérimentations sur des données générées synthétiquement et sur les articles de presse de Reuter.

Les tests ont été menés sur un processeur Intel Core i7 à 2.40 GHZ avec 16 Go de RAM sous Windows 8.1, 64 bit. Nous avons implémenté tous les algorithmes testés en Java. Pour l'implémentation de la structure de données LRST, nous avons étendu une implémentation de la structure LRT disponible sur https://github.com/epsilony/.

Expérimentations sur des données synthétiques

Ces expérimentations portent sur les effets de la longueur des mots et la taille de l'alphabet sur l'efficacité des différents approches et pour déterminer sous quelles conditions notre approche est plus performante.

Nous avons généré aléatoirement des paires de mots de longueurs différentes $(2, 4, \ldots, 8192)$ sur des alphabets de différentes tailles $(2, 4, \ldots, 8192)$. Pour simplifier la générations de mots, nous considérons les symboles de mots comme des entiers de $[1, \text{alphabet size}]$. Pour la commodité de la visualisation des données, nous avons utilisé l'échelle logarithmique sur tous les axes.

Afin de réaliser des expérimentations suffisamment précises, nous avons généré plusieurs paires pour la même longueur de mot et pour la même taille de l'alphabet. Pour chaque paire, nous avons pris plusieurs mesures du temps d'exécution avec une longueur des sous-séquences $p = 10$ et un facteur de pénalisation $\lambda = 0.5$.

Dans l'intention d'avoir une analyse expérimentale précise, nous avons distingué trois segments selon la longueur des mots : $(2, \ldots, 16)$, $(32, \ldots, 512)$ et $(1024, \ldots, 8192)$ nommés court, moyen et long. Ainsi, pour chaque taille de l'alphabet, nous calculons le temps d'exécution moyen sur toutes les longueurs de mots de chaque segment.

La Figure 5.8 révèle, pour notre approche géométrique, une dépendance inverse du temps d'exécution du calcul du noyau SSK avec la taille de l'alphabet. Cependant, pour une taille d'alphabet le temps d'exécution est proportionnel à la longueur du mot.

La Figure 5.9 montre les performances de l'approche proposée contre celles de Dynamique et Éparse. Tout d'abord, il est facile de remarquer que le temps d'exécution de Dynamique ne dépend pas de la taille de l'alphabet, il dépend uniquement de la longueur des mots pour une longueur p donnée des sous-séquences. En revanche, la dépendance de Éparse et Géométrique à la taille de l'alphabet est nettement visible.

Nous pouvons affirmer que Dynamique excelle pour les mots courts (tel que représenté dans la Figure 5.9 (a)), et aussi pour les mots longs et moyens sur un alphabet de petite taille (Figures 5.9 (b) et (c)). Cependant, notre approche (Géométrique) surpasse dans le cas des mots sur un alphabet de taille moyenne ou grande (taille de l'alphabet supérieure ou égale à 256) à l'exception de ceux ayant une longueur courte.

Il reste à présenter les résultats des essais comparatifs avec Éparse qui partage les mêmes motivations avec notre approche. Les auteurs de citealpRousu2005ECG affirment qu'avec des mots longs sur des alphabets de grandes tailles leur approche Éparse est plus rapide. La Figure 5.9 (c) montre que, dans ces conditions, notre approche domine. De plus, notre approche est plus rapide que Éparse pour les mots longs et absolument pour des mots sur des alphabets de grandes tailles. En revanche, elle devient plus lente que Éparse pour les mots courts basés sur des alphabets de petites tailles.

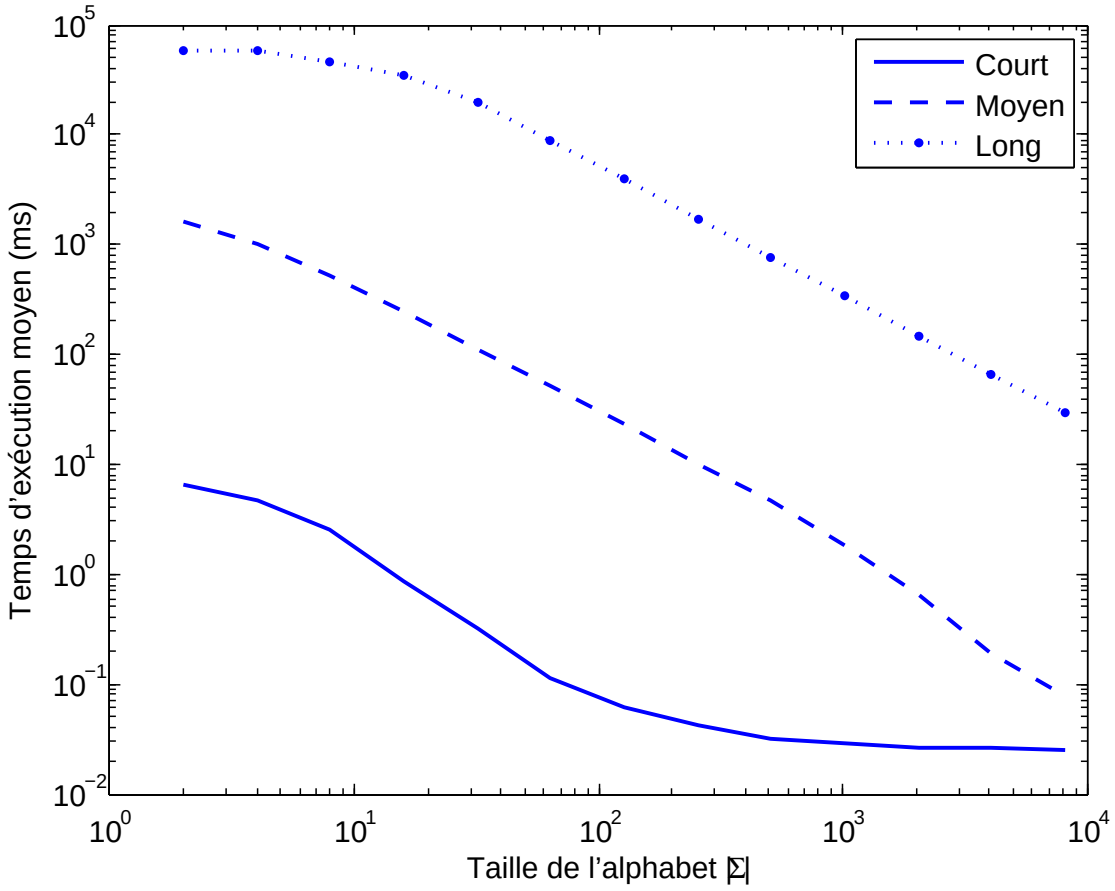


FIGURE 5.8 – Temps d'exécution moyen de l'algorithme Géométrique pour des données synthétiques sur trois segments de longueur de mots (Court, Moyen et Long). La longueur des sous-séquences $p = 10$ et le facteur de pénalisation $\lambda = 0.5$ sont utilisés.

Expérimentations sur des données réelles

Notre deuxième série d'expérimentations utilise la collection Reuters-21578 d'articles de presse en anglais, en tant qu'ensemble de données réel, pour évaluer l'efficacité de l'approche Géométrique contre Dynamique et Éparse. Nous avons créé un ensemble de données représenté sous forme de séquences de syllabes en transformant tous les articles XML en documents texte. D'une part, nous avons choisi la représentation syllabique des documents textuels pour préserver les mêmes conditions des expérimentations menées par citealpRousu2005ECG ; d'autre part, les syllabes peuvent être efficacement utilisés conjointement avec les noyaux de mots (Saunders *et al.* 2002).

Les documents texte sont pré-traités en ôtant les mots vides, les signes de ponctuation, les symboles spéciaux et finalement nous avons procédé au découpage des mots en syllabes. Au total, nous avons généré 22260 syllabes distinctes.

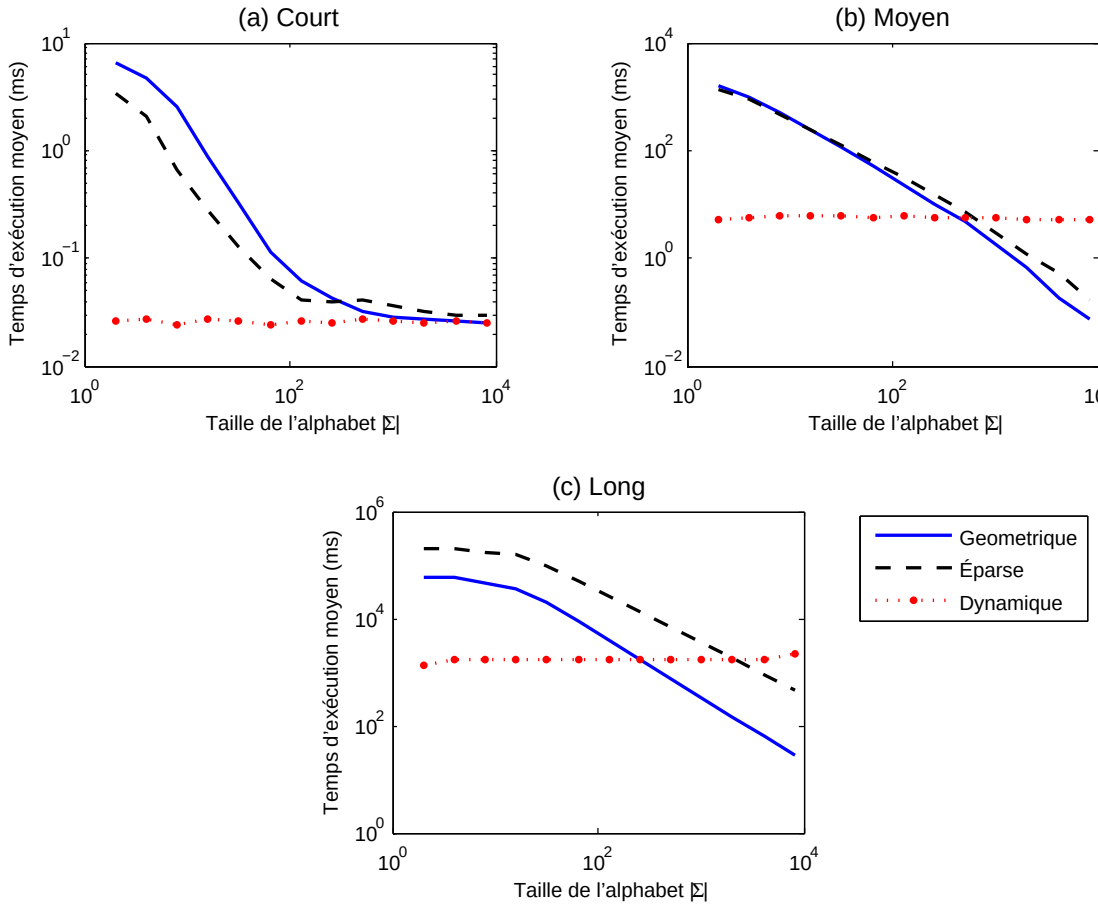


FIGURE 5.9 – Comparaison du temps d'exécution moyen des approches Géométrique, Dynamique et Éparse pour des données synthétiques sur trois segments de longueur de mots (Court, Moyen et Long). La longueur des sous-séquences $p = 10$ et le facteur de pénalisation $\lambda = 0.5$ sont utilisés.

Comme dans la première série des expérimentations, nous avons codé chaque alphabet syllabe par un entier. Dans le dessein de traiter aléatoirement les documents, nous avons remanié (shuffle) cet ensemble de données préliminaires.

Pour des commodités de la visualisation, lors de la création des paires de documents, nous avons veillé à ce que leurs longueurs soient proches. Sous cette condition, nous avons recueilli 916 paires de documents en tant qu'ensemble de données final.

Afin de comparer les algorithmes candidats, nous avons calculé le noyau SSK pour chaque paire de documents de l'ensemble de données en faisant varier la longueur des sous-séquences p de 2 à 20.

La Figure 5.10 et la Figure 5.11 représentent respectivement les clusters de documents où Géométrique est plus rapide que Dynamique et Éparse. Une paire

de document (s, t) est représentée graphiquement en fonction de la fréquence de correspondance inverse (axe des X) et la taille du document (axe des Y).

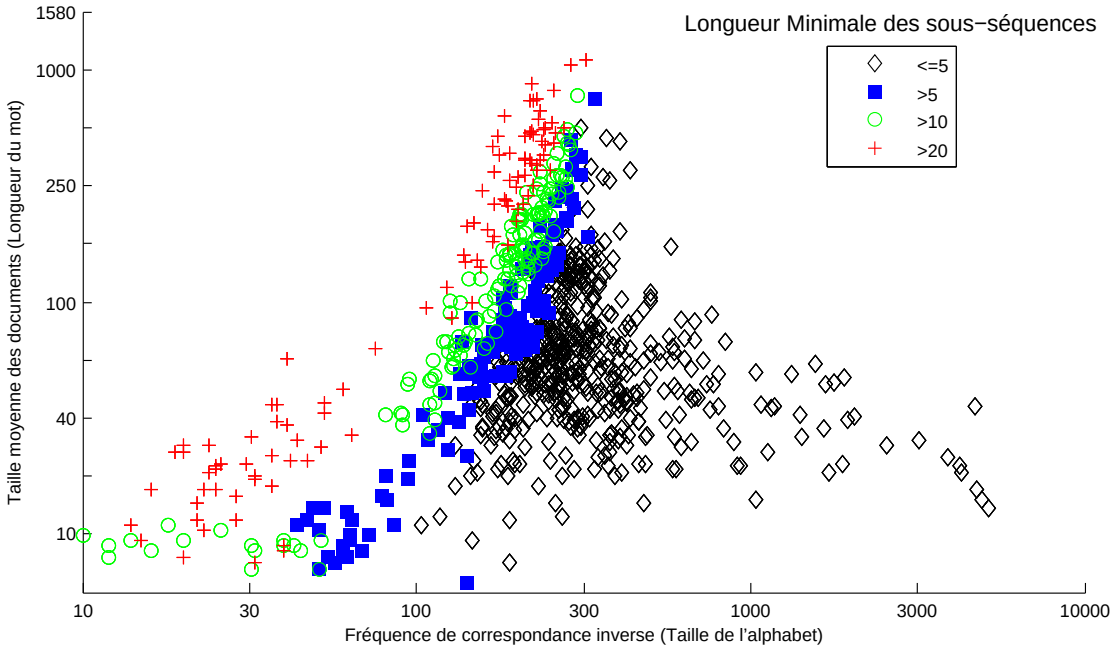


FIGURE 5.10 – Les clusters des paires de documents où Géométrique est plus rapide que Dynamique en fonction de la longueur des sous-séquences p.

La fréquence de correspondance inverse (inverse match frequency) est donnée par : $|s||t|/|L|$, elle joue le rôle de la taille de l'alphabet inhérente aux documents s et t. La taille du document est calculée comme la moyenne arithmétique des tailles des paires de documents, elle joue le rôle de la longueur du mot. Chaque cluster se distingue par un marqueur spécial qui correspond à la longueur minimale de la sous-séquence nécessaire pour rendre Géométrique plus rapide que Dynamique et/ou Éparse. Pour le cluster marqué par les diamants noirs, $p \leq 5$ est suffisante. La longueur $5 < p \leq 10$ est requise pour le cluster marqué par des carrés bleus remplis. Pour le cluster marqué par des cercles verts, $10 < p \leq 20$ est exigée. Quant au dernier groupe marqué par des signes plus, $p \geq 20$ est nécessaire.

En examinant la Figure 5.10, il convient de distinguer trois cas de figures. Le premier cas surgit lorsque la fréquence de correspondance inverse est faible (alphabet de petite taille), c'est-à-dire pour une table de programmation dynamique dense. Dans ce cas, nous aurons besoin de valeurs importantes de la longueur des sous-séquences ($p > 10$ pour les petits documents et $p > 20$ pour les plus longs) pour rendre Géométrique plus rapide que Dynamique. Le deuxième cas concerne les bonnes fréquences de correspondance inverses (alphabet de grande

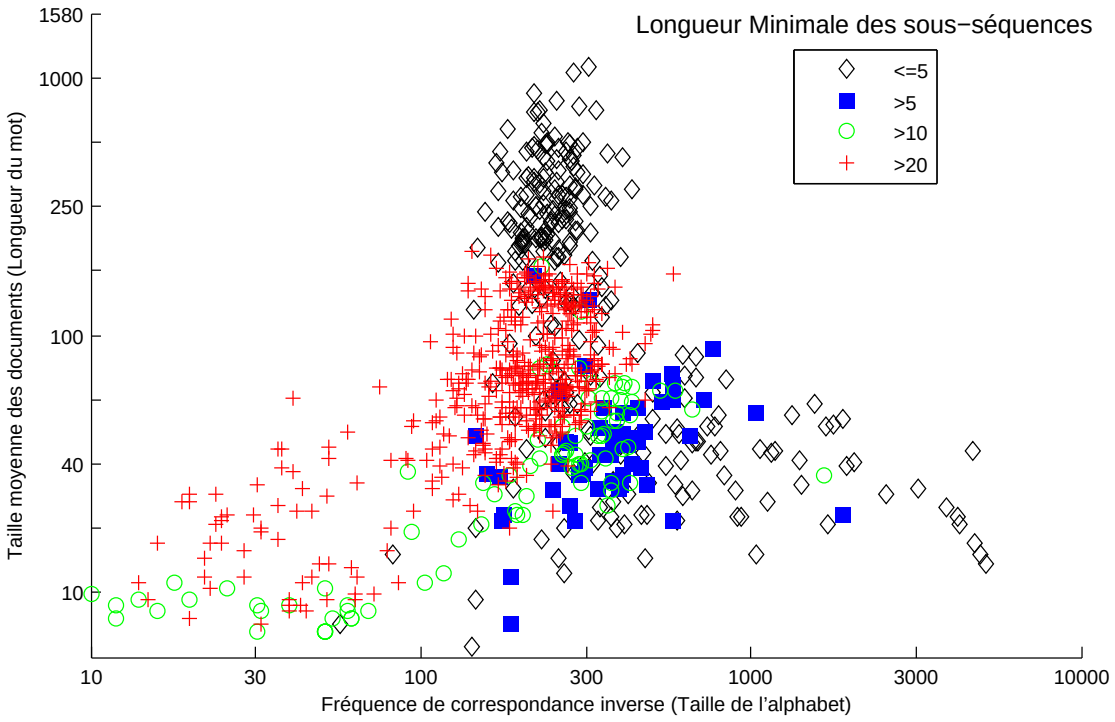


FIGURE 5.11 – Les clusters des paires de documents où Géométrique est plus rapide que Éparse en fonction de la longueur des sous-séquences p.

taille) afférentes à des tables de programmation dynamique creuses. Dans ce cas, les petites valeurs de la longueur des sous-séquences ($p \leq 5$) suffisent pour que Géométrique soit plus rapide que Dynamique. Le troisième cas apparaît lorsque nous traitons des fréquences de correspondance inverses modérées (alphabets de tailles moyennes), les valeurs de p qui rendent l'approche Géométrique plus rapide que Dynamique dépend de la taille des documents. Pour les documents de grandes tailles, les grandes valeurs de p sont requises.

Les résultats de la comparaison entre Géométrique et Éparse sur les données des articles de presse sont dépeints par la Figure 5.11. Un éventail de trois cas possibles peut être discuté. Le premier cas émerge lorsque la taille du document devient grande et aussi pour de bonnes fréquences de correspondances inverses. Dans ce cas, les petites valeurs de la longueur des sous-séquences ($p \leq 5$) suffisent pour faire Géométrique plus rapide que Éparse. Le second cas apparaît pour les petits documents et pour de mauvaises fréquences de correspondances inverses. La longueur des sous-séquences requises doit être importante ($p > 10$ pour les très petits documents et $p > 20$ pour les petits). Le troisième cas concerne les fréquences de correspondances inverses modérées. Dans ce cas, la valeur de la longueur des sous-séquences qui fait Géométrique plus rapide que Éparse

dépend de la taille du document, sauf pour les documents de grandes tailles qui tombent dans le premier cas.

5.3.4 Discussion des résultats de l'expérimentation

Les résultats des deux séries d'expérimentation révèlent que les algorithmes comparés se comportent essentiellement de la même manière, à la fois sur des données générées synthétiquement et sur les données réelles. Ces résultats témoignent que notre approche Géométrique surpasse les autres approches pour l'alphabet de grande taille, sauf pour les mots les plus courts. En outre, en ce qui concerne l'approche Éparse, Géométrique est compétitive pour les mots longs.

Nous pouvons étayer ces résultats par l'argumentation suivante : d'abord, la taille de l'alphabet et la longueur des mots affectent sensiblement la forme de la matrice du noyau. Les alphabets de grandes tailles peuvent réduire potentiellement les sous-séquences de correspondance, en particulier sur des mots longs, donnant lieu à une forme de matrice creuse. Par conséquent, une grande quantité de données stockées dans la matrice du noyau ne contribuent pas au résultat. Dans les autres cas, pour une matrice dense, notre approche peut être moins efficace que Dynamique avec un facteur au plus $\log |L|$.

Par ailleurs, comme la taille de l'ensemble des listes de correspondance de l'approche Éparse est égale à la taille de la liste de correspondance de Géométrique, nous pouvons conclure que la complexité des approches Géométrique et Éparse ne diffèrent que par les facteurs $\log |L|$ et $\log \min(|s|, |t|)$. Alors, la dépendance inverse de $|L|$ et $|\Sigma|$ s'engage en faveur de notre approche. En outre, les comparaisons conduites sur notre ensemble de données des articles de presse témoignent que plus de 73% $(677/916)$ des paires de documents produisent des listes de correspondance dont la taille est inférieure au minimum des tailles des paires de documents. De plus, les comparaisons montrent que pour les mots longs, $|L| << \min(|s|, |t|)$, en rappelant que la taille de la liste de correspondance diminue pendant que l'exécution SSK progresse.

Dans la littérature, les alphabets de grandes tailles font référence, généralement, aux documents musicaux ou textuels (Kuksa *et al.* 2009). Par exemple, dans le but de la classification du texte,
citealpicpr2010spatial ont utilisé l'ensemble de données Reuters-21578 dans

lequel l'alphabet de mots comprend 29.224 symboles. De même, pour la classification de la musique par genre, ils ont utilisé un benchmark standard (10 genres, chacun comprend 100 séquences audio, quantifiés en mots sur un alphabet de taille 1024). Par conséquent, sur la base de la propriété de modularité des méthodes à noyaux (Section 2.3.2), notre approche peut être utilisée dans de nombreuses applications telles que la classification de texte, l'extraction de l'information et la classification de la musique par genre.

De plus, pour répondre aux requêtes d'intervalles orthogonales, l'approche Éparse invoque la requête d'intervalle unidimensionnelle plusieurs fois. Tandis que l'approche Géométrique fournit de bons scores en utilisant les requêtes d'intervalles orthogonales conjointement avec la mise en œuvre de la technique de la cascade fractionnaire tout en exploitant notre extension de la structure de données LRT pour obtenir directement la somme d'intervalles.

Cependant, notre approche proposée dépend de la taille de l'alphabet. Elle ne peut pas être appliquée efficacement pour des problèmes avec un alphabet de petite taille comme dans le cas des traitements de bio-séquences. Une alternative à cette limitation consiste à développer une plate-forme qui implémente les différentes approches qui calculent efficacement le SSK. La sélection automatique de l'approche appropriée, dans la pratique, sera faite par un modèle d'anticipation qui sera conçu ultérieurement.

De plus, nous pouvons utiliser notre approche conjointement avec la technique de la programmation dynamique. Le scénario commence en appelant Dynamique pour les premières étapes, mais une fois que la table de la programmation dynamique devient creuse, pendant la progression de la longueur des sous-séquences p, nous lançons Géométrique qui sera plus efficace pour la tâche restante.

Au niveau de l'implémentation, un grand effort de programmation est épargné par la présence de programmes de la géométrie algorithmique bien étudiés et prêts à utiliser. Par conséquent, l'accent sera mise sur des variantes de noyaux de séquences qui peuvent être facilement adaptées.

5.4 Conclusion

Ce chapitre avait l'ambition de répondre à la question de recherche « efficacité de calcul des fonctions noyaux » qui est une propriété clé des méthodes à noyaux. Pour ce faire, nous avons ciblé le noyau SSK (String Subsequence Kernel), car il est déployé avec succès dans plusieurs tâches d'apprentissage automatique.

Nous avons commencé par une brève présentation des travaux connexes (présentés en détail dans le Chapitre 3).

Ensuite, nous nous sommes focalisés sur notre première contribution à base de liste linéaire chaînée utilisée conjointement avec la structure spatiale « layered range tree » (LRT).

Dans une perspective d'améliorer notre première contribution, nous avons proposé une nouvelle approche géométrique qui se repose sur l'extension de la structure LRT avec les opérations d'agrégation, donnant naissance, ainsi, à une nouvelle structuré spatiale baptisée « lyered range sum tree » (LRST).

Pour valider notre approche géométrique (à base de LRST), nous avons conduit des expérimentations à la fois sur des données synthétiques et réelles. Les résultats des deux séries des expérimentations ont témoigné que notre approche est plus efficace pour l'alphabet de grande taille à l'exception des mots trop courts.

Chapitre 6

Noyaux de séquences : unification et généralisation

DEPUIS des millénaires, des chercheurs ont travaillé avec ténacité sur le principe de l'unification universelle. En effet, le but de cette tendance constante vers une théorie d'unification consiste à découvrir le secret de l'univers naturel (Hoare 1996). Cependant, si le but de l'unification des théories a été si réussi en sciences naturelles, en physique moderne et en mathématiques modernes, il semble que l'idée d'unifier les théories de l'informatique en général (Hoare 1996) et de la fouille de données en particulier (Yang & Wu 2006) est plus difficile.

Au cours des dernières années, un effort important a été consacré aux noyaux de séquences centré sur des problèmes individuels, ce qui a conduit à une variété d'approches. En tant que contribution au développement d'une théorie d'unification de l'apprentissage automatique, notre objectif pour le présent chapitre, consiste à proposer une plate-forme générale pour calculer les noyaux de séquences qui peut être aussi utile pour le traitement des noyaux d'*ensembles de séquences*. Ceci en tenant compte de l'efficacité de calcul qui est une propriété clé des méthodes à noyaux.

Le chapitre est organisé comme suit : nous présentons d'abord quelques informations préliminaires inhérentes aux automates pondérés. La suite sera réservée à notre contribution qui comporte la présentation de notre approche,

l'évaluation des noyaux de séquences à base d'automates pondérés ainsi que son amélioration. Ensuite, la caractéristique d'unification de notre modèle est mise en exergue par le biais d'une étude de la relation entre notre approche avec d'autres noyaux de séquences et enfin, la généralisation de notre modèle pour un nouveau noyau d'ensembles de séquences est détaillée.

6.1 Préliminaires

Cette section est un complément des informations préliminaires présentées dans la Section 3.1.4.

Un *automate fini* sur un alphabet Σ est un quintuplet $A = (\Sigma, Q, E, I, F)$, où Q est un ensemble fini d'états, $E \subseteq Q \times \Sigma \times Q$ est un ensemble de transitions, I (resp. F) $\subseteq Q$ est un ensemble d'états initiaux (resp. finaux). Un *chemin* de A est dit réussi si son état de départ appartient à I et celui d'arrivée appartient à F.

Un *automate pondéré* sur un semi-anneau $\mathbb{S}$ est un 7-uplet $A = (\Sigma, Q, E, I, F, \lambda, \rho)$, où Σ est un alphabet, Q est un ensemble fini d'états, $E \subseteq Q \times \Sigma \times S \times Q$ est un ensemble fini de transitions, I (resp. F) $\subseteq Q$ est un ensemble d'états initiaux (resp. finaux), $\lambda : I \to S$ (resp $\rho : F \to S$) est la fonction de pondération pour les états initiaux (resp. finaux).

Pour une transition $e = (p, a, s, q) \in E$, nous appelons $o(e) = p$ son état origine ou source, $l(e) = a$ son étiquette, $wt(e) = s$ son poids ou coefficient et $d(e) = q$ son état destination ou arrivé. Si $s = 0$, nous considérons qu'il n'existe pas une transition entre les états p et q. Un chemin $\pi = e_1 e_2 \dots e_k$ est une suite de transitions de telle sorte que l'origine de chacune est la destination de la précédente. Nous pouvons étendre les fonctions précédentes pour les chemins en définissant $o(\pi) = o(e_1)$, $d(\pi) = d(e_k)$, l'étiquette $l(\pi) = l(e_1)l(e_2)\dots l(e_k)$ est le produit des étiquettes des transitions qui le constituent. Le poids du chemin π est le produit des poids de ses transitions, $wt(\pi) = wt(e_1) \cdot wt(e_2) \dots \cdot wt(e_k)$. Un chemin de A est dit réussi (calcul) si son état de départ appartient à I et celui d'arrivée appartient à F. Le poids d'un calcul est le poids du chemin multiplié respectivement à gauche et à droite par le poids initial de l'état de départ et le poids final de l'état d'arrivée.

Soit Σ un alphabet et $\mathbb{S}$ un semi-anneau, une série formelle r est une application $\Sigma^* \to \mathbb{S}$. L'image par r d'un mot $w \in \Sigma^*$ est dénotée par (r, w) au lieu de $r(w)$ et il est appelé le coefficient de w dans r. La série r, elle même, est exprimée en tant que somme formelle :

$$r = \sum_{w \in \Sigma^*} (r, w)w. \tag{6.1}$$

Le support de r est le langage $supp(r) = \{w \in \Sigma^* | (r, w) \neq 0\}$. L'ensemble de séries formelles sur Σ à coefficients dans $\mathbb{S}$ est désigné par $\mathbb{S}\langle\langle\Sigma\rangle\rangle$. Une série avec un support fini est un polynôme. $\mathbb{S}\langle\Sigma\rangle$ désigne l'ensemble des polynômes.

Une structure d'un semi-anneau est définie sur $\mathbb{S}\langle\langle\Sigma\rangle\rangle$. Pour $r_1, r_2, r \in \mathbb{S}\langle\langle\Sigma\rangle\rangle$ et $s \in \mathbb{S}$, les opérations suivantes sont définies :

- La somme : $(r_1 + r_2, w) = (r_1, w) + (r_2, w)$,
- le produit de Cauchy : $(r_1 \cdot r_2, w) = \sum_{w_1 w_2 = w} (r_1, w_1)(r_2, w_2)$,
- le produit d'Hadamard : $(r_1 \odot r_2, w) = (r_1, w)(r_2, w)$,
- le produit scalaire : $(sr, w) = s(r, w)$ et $(rs, w) = (r, w)s$.

L'ensemble des polynômes $\mathbb{S}\langle\Sigma\rangle$ est un sous semi-anneau de $\mathbb{S}\langle\langle\Sigma\rangle\rangle$.

6.2 Modèle à base d'automates pondérés

Cette section se focalise sur la présentation de notre approche dont le but consiste à proposer une plate-forme générale pour le calcul des noyaux de séquences. Une telle plate-forme peut être aussi utilisée pour le traitement les noyaux d'ensembles de séquences.

Tout au long de notre étude, nous serons guidés par la citation suivante (Mohri *et al.* 2012) :

> « essential need for kernels is an efficient computation, and more complex definitions would lead to substantially more costly computational complexities for kernel computation ».

Ainsi, nous devons trouver un compromis entre l'efficacité de calcul du noyau et la robustesse du modèle de séquence.

Dans ce qui suit, nous commençons par puiser l'idée générale du modèle proposé qui sera suivi par la définition d'un automate pondéré représentant

toutes les sous-séquences d'un mot donné. Enfin, nous nous intéressons à la validité et à l'efficacité de notre approche.

6.2.1 Principe général

L'idée principale derrière notre approche est la relation entre le triplet « noyau », « séries formelles » et « automates pondérés ». Sans perte de généralité, nous considérons le noyau toutes sous-séquences avec trous. L'espace de redescription de ce noyau est indexé par tous les mots de Σ^*. Une composante d'une sous-séquence w dans un mot s est représentée par :

$$\Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)},\ w \in \Sigma^*. \tag{6.2}$$

L'Équation (6.2) est une application de Σ^* dans $\mathbb{S}$ (dans notre contexte $\mathbb{S} = \mathbb{R}$), elle ressemble à la définition des séries formelles dans (6.1), où $(r_s, w) = \sum_{I:w=s(I)} \lambda^{l(I)}$ est le coefficient de la série. Par conséquent, nous pouvons considérer la série formelle de l'Équation 6.3 comme une nouvelle représentation du mot s dans l'espace de redescription.

$$r_s = \sum_{w\in\Sigma^*} \sum_{I:w=s(I)} \lambda^{l(I)} w \tag{6.3}$$

Du point de vue théorie des automates, cela peut être vu comme le comportement $\|A\|$ d'un automate pondéré, c'est-à-dire le poids de toutes les sous-séquence $w \in \Sigma^*$ dans le mot s :

$$\|A\| : \Sigma^* \to \mathbb{S},$$

avec

$$(\|A\|, w) = \sum_{\pi:l(\pi)=w} wt(\pi) = \sum_{I:w=s(I)} \lambda^{l(I)} = \Phi_w(s). \tag{6.4}$$

Nous rappelons que nous avons utilisé, ici, la pondération proposée dans citealpLodhi2002TCU et nous supposons que les fonctions de pondérations initiale et finale λ et ρ sont des scalaires égaux à 1. Ceci est bien un cas de figure des relations entre la théorie des langages formelles et celle des séries formelles.

Nous pouvons, maintenant, conclure que la projection des mots dans un espace de redescription de haute dimension du noyau toutes sous-séquences avec trous est représentée par une série formelle qui peut être réalisée par un automate pondéré. Alors, les noyaux de séquences peuvent être représentés d'une façon efficace et compacte à l'aide des automates pondérés. Par ailleurs, il est bien connu que ce modèle correspond bien à la représentation d'un ensemble d'objets. Ces caractéristiques font des automates pondérés un bon candidat pour atteindre notre objectif de développement d'une plate-forme générale pour calculer les noyaux des séquence et les noyaux d'ensembles de séquences.

6.2.2 Automate pondéré représentant toutes les sous-séquences

Dans cette section, nous définissons une construction particulière d'un automate pondéré compact et non déterministe qui réalise la projection associée au noyau toutes sous-séquences avec trous donné par l'Équation 6.3. Une telle construction est baptisée *GASWA* pour *Gappy All Subsequence Weighted Automaton*.

Définition 6.1. (**GASWA**) Soit un mot $s = s_1 s_2 ... s_n$ sur Σ^*. L'automate pondéré de toutes les sous-séquences avec trous (GASWA) associé $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ est défini comme suit :

- $Q_s = \{q_0\} \cup \{q_i, q'_i, q''_i \ / \ 1 \leq i \leq n\}$.
- $E_s = \{(q_0, \lambda, s_1, q_1), (q_0, \lambda, s_1, q'_1), (q_0, 1, \epsilon, q''_1)\} \cup \{(q'_i, \lambda, s_{i+1}, q_{i+1}) \ / \ 1 \leq i \leq n-1\} \cup \{(q'_i, \lambda, s_{i+1}, q'_{i+1}) \ / \ 1 \leq i \leq n-1\} \cup \{(q'_i, \lambda, \epsilon, q'_{i+1}) \ / \ 1 \leq i \leq n-1\} \cup \{(q''_i, \lambda, s_{i+1}, q_{i+1}) \ / \ 1 \leq i \leq n-1\} \cup \{(q''_i, 1, \epsilon, q''_{i+1}) \ / \ 1 \leq i < n-1\}$.
- $I_s = \{q_0\}$.
- $F_s = \{q_i \ / \ 0 \leq i \leq n\}$.
- Les fonctions de pondération initiale et finale λ et ρ sont des scalaires égaux à 1.

La Figure 6.1 illustre la définition de GASWA. Pour la commodité des applications spécifiques du calcul du noyau, GASWA est défini sur le semi-anneau des probabilités $\langle \mathbb{R}_+, +, \times, 0, 1 \rangle$. Nous pouvons distinguer trois niveaux contenant respectivement les états $q_0, q'_1, \ldots, q'_n$; $q_0, \ldots, q_n$ et $q_0, q''_1, \ldots, q''_{n-1}$. Le premier niveau représente le mot s. Le poids de chaque transition est λ^1 pondérant la

distance entre deux symboles consécutifs dans le mot s. Pour autoriser les trous, le premier niveau est étendu par des ϵ-transitions $e_i = (q'_i, \epsilon, \lambda, q'_{i+1}),\ i = 1..n-1$, chaque trou est pénalisé par un facteur λ. Notez l'absence d'une ϵ-transition entre (q_0, q'_1) car l'effet de cette transition est identique à celui de commencer la sous-séquence à partir de l'état q''_1.

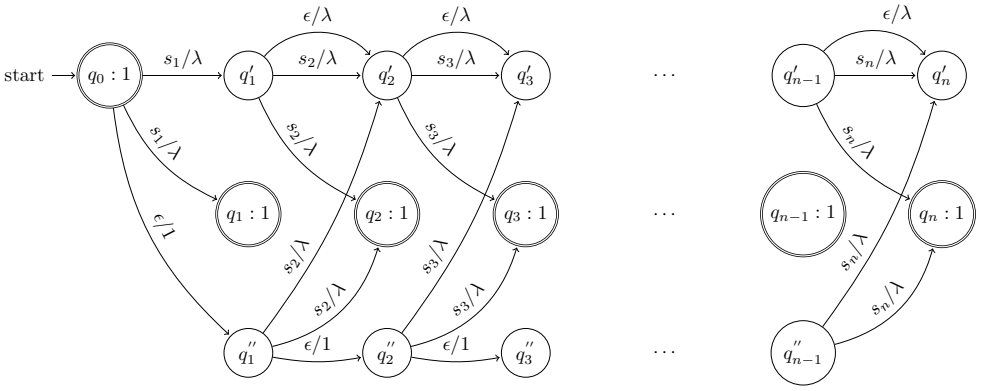


FIGURE 6.1 – Automate pondéré pour toutes les sous-séquences d'un mot $s = s_1 s_2 \dots s_n$.

Afin de mettre fin à une sous-séquence à un symbole donné, nous devons rajouter des transitions à partir du premier niveau au second : $e_i = (q'_i, s_{i+1}, \lambda, q_{i+1}),\ i = 0..n-1$, avec $q'_0 = q_0$. Par contre, pour commencer avec n'importe quel symbole de s, nous devons créer les transitions de troisième niveau : $e_i = (q''_i, \epsilon, \lambda^0, q''_{i+1}),\ i = 0..n-2$, avec $q''_0 = q_0$. Le poids $\lambda^0 = 1$ indique l'absence d'un trou ou d'une distance entre deux symboles. A partir de l'état $q''_i,\ i = 1..n-1$, nous pouvons soit terminer une sous-séquence avec la transition $e_i = (q''_i, s_{i+1}, \lambda, q_{i+1})$ ou bien continuer la sous-séquence à partir de l'état q'_{i+1} à travers la transition $e_i = (q''_i, s_{i+1}, \lambda, q'_{i+1})$.

Pour illustrer la Définition 6.1, nous construisons un GASWA pour le mot $s = cata$. L'espace de redescription associé à s peut être représenté par la Table 6.1. La projection du mot $s = cata$ peut être représentée par la série formelle de l'Équation 6.5.

TABLE 6.1 – Espace de redescription du mot $s = cata$ associé au noyau toutes sous-séquences avec trous.

Sous-séquence	ϵ	c	a	t	ca	ct	at	aa	cat	caa	ata	cata
coefficient	1	λ	2λ	λ	$\lambda^2 + \lambda^4$	λ^3	λ^2	λ^3	λ^3	λ^4	λ^3	λ^4

$$r_s = 1 + \lambda c + 2\lambda a + \lambda t + (\lambda^2 + \lambda^4)ca + \lambda^2 at + \lambda^3 aa + \lambda^3 cat + \lambda^4 caa + \lambda^3 ata + \lambda^4 cata. \tag{6.5}$$

La série formelle de l'Équation 6.5 peut être réalisée par l'automate GASWA de la Figure 6.2.

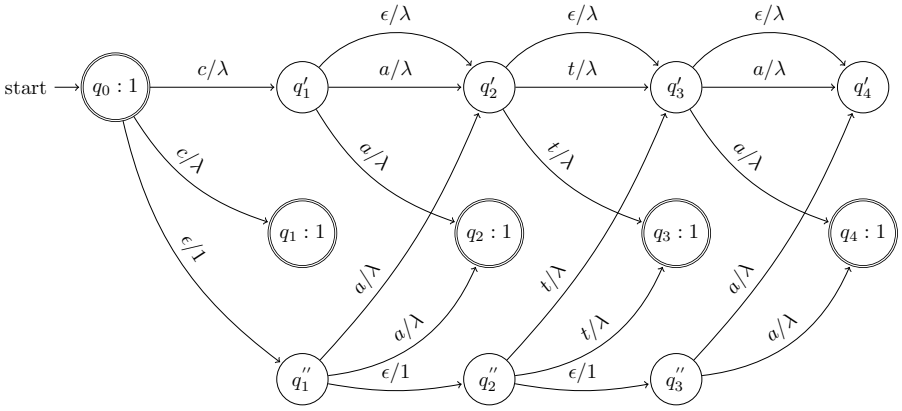


FIGURE 6.2 – Automate pondéré pour toutes les sous-séquences du mot $s = cata$.

6.2.3 GASWA : validité et efficacité

Dans cette section, nous nous concentrons sur la validité de la construction de GASWA et son efficacité. Tout d'abord, nous prouvons que la série formelle associée au noyau toutes sous-séquences avec trous est égale à celle réalisée par GASWA. Ensuite, nous démontrons que la construction GASWA est compacte.

Proposition 6.2. *Soit $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ un automate pondéré GASWA associé au mot $s = s_1 s_2 \dots s_n$. Nous dénotons par $\mathfrak{S}_k$, $k = 0..n$ la série formelle réalisée dans l'état q_k qui peut être formalisée comme suit :*

$$\mathfrak{S}_k = \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w, \tag{6.6}$$

I^k dénote les indices des tuples I où $i_k = s_k$.

Démonstration. La preuve est par induction. Le cas de base correspond à $k = 0$ et $k = 1$.

Commençons par $k = 0$, dans ce cas I^0 correspond au tuple vide, par convention $l(I^0) = 0$ (voir Section 3.4.2), donc $\mathfrak{S}_0 = 1\epsilon$. En examinant la Figure 6.1, il

est facile de voir que la série réalisée dans l'état q_0 est égale à 1ϵ, le coefficient 1 correspond à la pondération initiale de l'état q_0. Ainsi, l'affirmation de l'Équation 6.6 est vérifiée pour le cas de base $k = 0$.

Pour le cas où $k = 1$, $\mathfrak{S}_1 = \lambda s_1$. En examinant la Figure 6.1, il n'est pas difficile de voir que la série réalisée dans l'état q_1 est égale à λs_1. Donc l'affirmation de l'Équation 6.6 est vraie pour le cas de base $k = 1$.

Maintenant, supposons que l'Équation 6.6 est correcte dans l'état q_k, pour un $k > 1$. Nous devons la démontrer pour l'état q_{k+1}, soit :

$$\mathfrak{S}_{k+1} = \sum_{I^{k+1}:w=s(I^{k+1})} \lambda^{l(I^{k+1})} w. \tag{6.7}$$

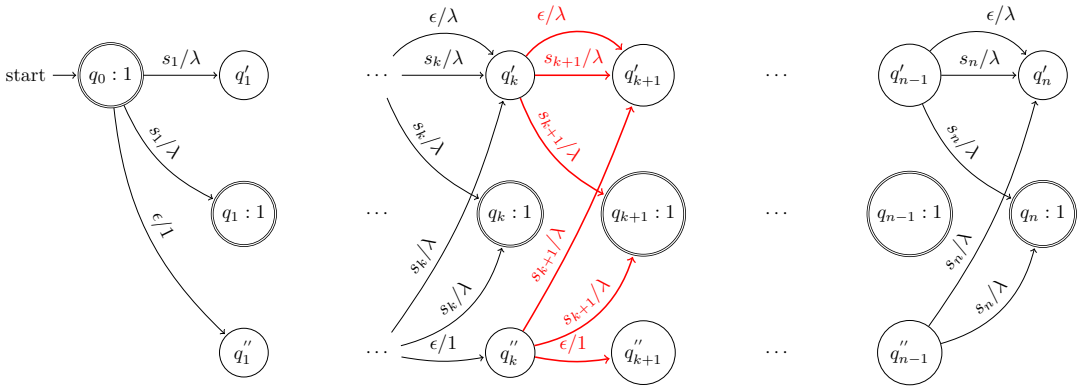


FIGURE 6.3 – Automate pondéré pour toutes les sous-séquences d'un mot $s = s_1 s_2 \dots s_n$ pour le but de la preuve par induction.

Par construction (Figure 6.3), La série formelle dans l'état q_{k+1} peut être exprimée comme suit :

$$\mathfrak{S}_{k+1} = \lambda s_{k+1} + \lambda s_{k+1} \mathfrak{S}'_k, \tag{6.8}$$

où $\mathfrak{S}'_k$ dénote la série réalisée dans l'état q'_k.

$$\begin{aligned} \mathfrak{S}'_k &= \mathfrak{S}_k + (\mathfrak{S}_k - \lambda s_k) s_k^{-1} \\ &= \mathfrak{S}_k + \mathfrak{S}_k s_k^{-1} - \lambda. \end{aligned} \tag{6.9}$$

Notez que s_k^{-1} est la dérivation du second terme de l'Équation 6.9 par s_k.
Par substitution de 6.9 dans 6.8 nous obtenons :

$$\begin{aligned}
\mathfrak{S}_{k+1} &= \lambda s_{k+1} + \lambda s_{k+1}(\mathfrak{S}_k + \mathfrak{S}_k s_k^{-1} - \lambda) \\
&= \lambda s_{k+1} + \lambda s_{k+1} \left[\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w + \Big(\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \Big) s_k^{-1} - \lambda \right] \\
&= \lambda s_{k+1} + \lambda \Big(\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \Big) s_{k+1} + \left[\lambda \Big(\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \Big) s_{k+1} \right] s_k^{-1} - \lambda^2 s_{k+1} \\
&= \lambda s_{k+1} + \lambda \Big(\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \Big) s_{k+1} \\
&\quad + \left[\lambda (\lambda s_k s_{k+1}) s_k^{-1} + \Big(\sum_{I^k_{>1} : w = s(I^k_{>1})} \lambda^{l(I^k_{>1})} w \Big) s_{k+1} s_k^{-1} \right] - \lambda^2 s_{k+1} \\
&= \lambda \Big(\sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \Big) s_{k+1} + \left[\Big(\sum_{I^k_{>1} : w = s(I^k_{>1})} \lambda^{l(I^k_{>1})} w \Big) s_{k+1} s_k^{-1} + \lambda s_{k+1} \right],
\end{aligned} \tag{6.10}$$

où $I^k_{>1}$ dénote les tuples I avec $i_k = s_k$ et $|I| > 1$.

Le premier terme de l'Équation 6.10 représente la série formelle caractérisant toutes les sous-séquences de s se terminant par s_k et s_{k+1} dénotée plus tard par les tuples $I^{k,k+1}$. Tandis que le second terme représente la série formelle caractérisant toutes les sous-séquences qui se terminent par s_{k+1} en excluant celles contenant s_k, dénotées plus tard par $I^{k^{-1},k+1}$.

Par conséquent, nous pouvons réécrire l'Équation 6.10 comme suit :

$$\mathfrak{S}_{k+1} = \sum_{I^{k,k+1} : w \in s(I^{k,k+1})} \lambda^{l(I^{k,k+1})} w + \sum_{I^{k^{-1},k+1} : w \in s(I^{k^{-1},k+1})} \lambda^{l(I^{k^{-1},k+1})} w. \tag{6.11}$$

Notez que $I^{k,k+1} \cup I^{k^{-1},k+1} = I^{k+1}$ (i.e. les tuples qui représentent toutes les sous-séquences du mot s et qui se terminent par s_{k+1}). Par conséquent, L'Équation 6.11 devient :

$$\mathfrak{S}_{k+1} = \sum_{I^{k+1} : w \in s(I^{k+1})} \lambda^{l(I^{k+1})} w. \tag{6.12}$$

L'Équation 6.12 coïncide exactement avec (6.7).

Nous concluons que $\mathfrak{S}_k = \sum_{I^k:w=s(I^k)} \lambda^{l(I^k)} w, k = 1..n.$ □

Ainsi, nous pouvons utiliser la Proposition 6.2 pour démontrer la validité de l'automate GASWA.

Lemme 6.3. *Soit* $s = s_1 s_2 \dots s_n$ *un mot sur* Σ^*. *L'automate GASWA* $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ *réalise la série formelle associée à l'espace de redescription de la projection correspondante au noyau toutes sous-séquences avec trous donnée par :*

$$\Phi : s \mapsto \Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)}, w \in \Sigma^*.$$

Démonstration. L'idée intuitive derrière cette preuve est que chaque état final q_k réalise une série associée à toutes les sous-séquences se terminant par $s_k, k = 1..n$. Nous pouvons organisé toutes les sous-séquences d'un mot s, $ss(I)$ comme suit :

$$\begin{aligned}
ss(I) = &\{\epsilon\} \\
&\uplus \{\{s_1\} \cup \{s_2\} \cup \dots \cup \{s_n\}\} \\
&\uplus \left\{w = s(I^2_{>1})\right\} \\
&\vdots \\
&\uplus \{w = s(I^n_{>1})\} \\
= &\{\epsilon\} \\
&\uplus \{s_1\} \\
&\uplus \left\{\left\{w = s(I^2_{>1})\right\} \cup \{s_2\}\right\} \\
&\vdots \\
&\uplus \{\{w = s(I^n_{>1})\} \cup \{s_n\}\} \\
= & \\
&\uplus \left\{w = s(I^1)\right\} \\
&\uplus \left\{w = s(I^2)\right\} \\
&\vdots \\
&\uplus \{w = s(I^n)\} \\
= &\{w = s(I_0)\} \uplus \left\{w = s(I^k), k = 1 \dots n\right\}
\end{aligned}$$

Par conséquent, nous pouvons reformuler l'Équation 6.3 associée à toutes les sous-séquences comme suit :

$$\begin{aligned} r_s &= \sum_{w \in ss(I)} \lambda^{l(I)} w \\ &= \sum_{I_0 : w = s(I_0)} \lambda^{l(I)} w + \sum_{k=1}^{n} \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \\ &= \epsilon + \sum_{k=1}^{n} \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \end{aligned} \tag{6.13}$$

Si nous revenons à GASWA, par définition, la série réalisée par un tel automate est bien la somme des séries réalisées dans les états finaux, soit :

$$\begin{aligned} r_{gaswa} &= \sum_{k=0}^{n} \mathfrak{S}_k \\ &= \mathfrak{S}_0 + \sum_{k=1}^{n} \mathfrak{S}_k \\ &= \mathfrak{S}_0 + \sum_{k=1}^{n} \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \\ &= \epsilon + \sum_{k=1}^{n} \sum_{I^k : w = s(I^k)} \lambda^{l(I^k)} w \end{aligned} \tag{6.14}$$

En comparant les Équations 6.13 et 6.14, nous pouvons conclure que l'automate GASWA réalise bien la série formelle inhérente à la projection associée au noyau toutes sous-séquences avec trous. □

Nous rappelons que tout au long du processus de construction de l'automate GASWA, le principe de l'efficacité de calcul du noyau associé à cette nouvelle représentation est pris en compte. Sachant que l'efficacité de calcul d'un tel noyau passe obligatoirement par une construction compacte de GASWA. Le corollaire 6.4 annonce cette propriété pour notre construction.

Corollaire 6.4. *Soit* $s = s_1 s_2 \dots s_n$ *un mot sur* Σ^*. *L'automate GASWA* $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ *représentant toutes les sous-séquence du mot* s *avec trous est compact.*

Démonstration. La taille de GASWA est linéaire en la longueur du mot s. $|Q_s| = 3n$ et $|E_s| = 6n - 4$, où $n = |s|$. □

Les résultats combinés du Lemme 6.3 et du Corollaire 6.4 conduisent au Théorème 6.5 qui synthétise les résultats de notre construction GASWA.

Théorème 6.5. *Soit* $s = s_1 s_2 \dots s_n$ *un mot sur* Σ^*. *L'automate GASWA* $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ *réalise la série formelle associée à l'espace de redescription de la projection correspondante au noyau toutes sous-séquences avec trous donnée par :*

$$\Phi : s \mapsto \Phi_w(s) = \sum_{I : w = s(I)} \lambda^{l(I)}, w \in \Sigma^*.$$

Et la construction GASWA est compacte.

6.3 Calcul du noyau de séquences à base d'automates pondérés

Le calcul du noyau toutes sous-séquences avec trous $K(s,t)$ peut être réalisé en deux étapes. La première étape consiste à construire l'automate intersection $A_{s,t} = A_s \cap A_t$ des deux automates pondérés $A_s = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ et $A_t = (\Sigma, Q_t, E_t, I_t, F_t, \lambda, \rho)$ représentant respectivement toutes les sous-séquences des mots s et t. La deuxième étape est l'évaluation de tous les calculs (chemins réussis qui représentent toutes les sous-séquences communes entre les mots s et t) de l'automate pondéré $A_{s,t}$. Ceci peut être réalisé par un algorithme généralisé de plus courte distance.

6.3.1 Intersection des automates pondérés

L'intersection des automates pondérés est une généralisation de l'intersection classique des automates finis (Hopcroft & Ullman 1979). Elle peut aussi être vue comme un cas particulier de la composition des transducteurs pondérés (Cortes *et al.* 2004; Mohri *et al.* 1996; Pereira & Riley 1997).

Les états de $A_{s,t}$ sont les paires des états à partir de A_s et les états à partir de A_t. Une transition $e_{s,t} = ((q_s, q_t), l(e_{s,t}), wt(e_{s,t}), (q'_s, q'_t))$ de $A_{s,t}$ est conçue à partir d'une transition $e_s = (q_s, l(e_s), wt(e_s), q'_s)$ de A_s et une transition $e_t = (q_t, l(e_t), wt(e_t), q'_t)$ de A_t, où $l(e_{s,t}) = l(e_s) = l(e_t)$ et $wt(e_{s,t}) = wt(e_s) \cdot wt(e_t)$ (le

produit du semi-anneau $\mathbb{S}$). Les états initiaux (finaux) de $A_{s,t}$ sont respectivement les paires des états initiaux (finaux) de A_s et de A_t.

Dans le pire des cas, toutes les transitions quittant q_s correspondent à toutes les transitions quittant q_t, ainsi la complexité spatiale et temporelles de l'intersection est $O(|A_s||A_t|)$.

Cependant, une généralisation naïve de cet algorithme pour le cas des ϵ-transitions peut générer des ϵ-chemins redondants. Dans le cas des automates pondérés sur un semi-anneau non idempotent, ces chemins redondants conduisent à un résultat incorrect. Pour faire face à ce problème pour l'algorithme de composition (Mohri *et al.* 1996; Pereira & Riley 1997), les auteurs ont étendu leur algorithme par le mécanisme de *filtrage-epsilon* portant sur les transducteurs pondérés. Nous avons adapté leur solution pour les automates pondérés.

Pour illustrer le problème sus-cité et la solution adaptée, nous discutons l'exemple d'intersection de deux automates A_s et A_t (Figure 6.4).

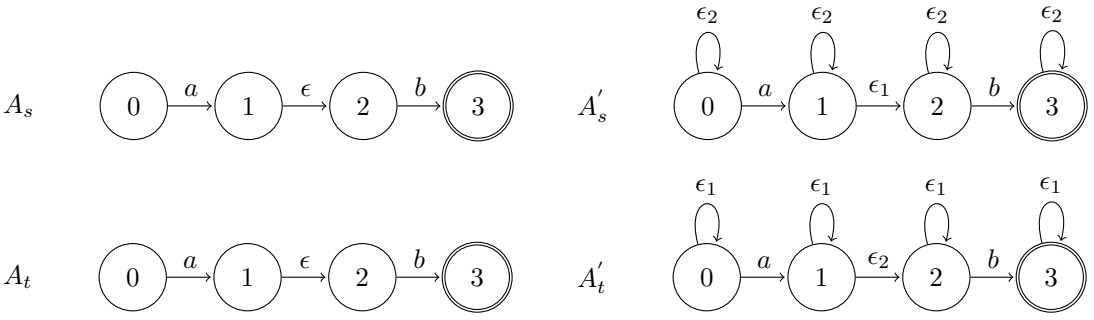


FIGURE 6.4 – Automates pondérés avec des transitions epsilon.

Pour comprendre comment ces ϵ-transitions fonctionnent, nous étendons A_s et A_t comme suit : les ϵ-transitions dans A'_s (A'_t) sont étiquetées ϵ_1 (ϵ_2), et les ϵ-transitions correspondant dans A'_s (A'_t) sont respectivement marquées par des boucles étiquetées ϵ_2 (ϵ_1). Une boucle dans un état d'un automate permet une attente dans le même état d'un tel automate tout en consommant des ϵ-libellés dans l'autre automate. Nous obtenons, ainsi, deux types de ϵ-mouvements dans les automates : (wait, move) et (move, wait) . La Figure 6.5 concrétise l'éventail des mouvements possibles pour l'intersection de A'_s et A'_t.

Nous pouvons observer, de façon remarquable, les chemins redondants menant à un résultat incorrect pour le cas pondéré. Pour être correct, nous

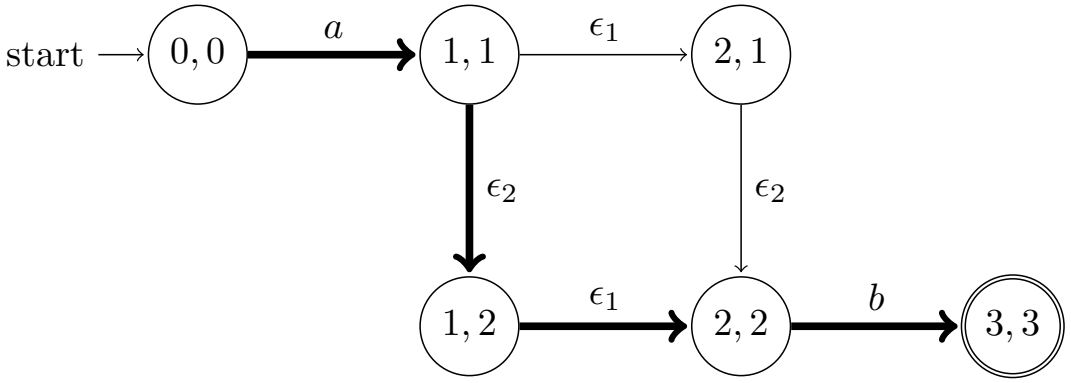


FIGURE 6.5 – ϵ-chemins redondants de l'intersection des automates A'_s et A'_t de la Figure 6.4.

devons garder un seul chemin. Nous avons adapté la solution filtre qui peut être encodée sous forme d'un automate d'états finis, tel que présenté dans la Figure 6.6.

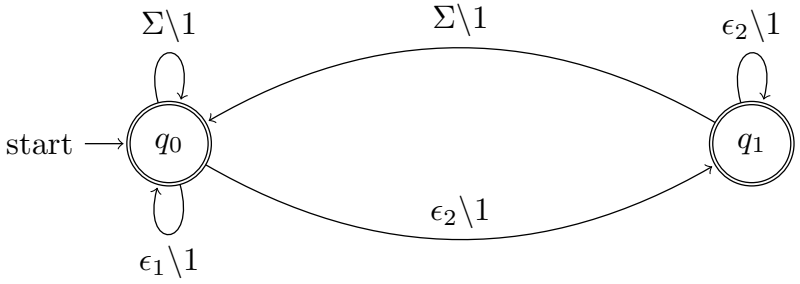


FIGURE 6.6 – Automate filtre pour éliminer les ϵ-chemins

Plus de détails pour la preuve et des discussions intuitives peuvent se trouver dans
citealpmohri96,Fernando97.

6.3.2 Évaluation du noyau

Afin d'évaluer le noyau de toutes les sous-séquences avec trous entre deux mots s et t, nous devons d'abord, construire l'automate pondéré $A_{s,t} = A_s \cap A_t \cap A_f$ sur un semi-anneau $\mathbb{S}$, où A_s et A_t sont respectivement les automates pondérés représentant toutes les sous-séquences des mots s et t. Tandis que A_f est l'automate filtre représenté dans la Figure 6.6. Par la suite, nous devons calculer la somme des poids de tous les calculs de $A_{s,t}$. Formellement, nous

devons calculer la distance :

$$dist(q_0) = \sum_{\pi \in P(q_0,F)} \lambda(o(\pi)) \cdot wt(\pi) \cdot \rho(d(\pi)). \tag{6.15}$$

Nous dénotons par $P(q_0, F)$ l'ensemble des chemins à partir de l'état initial q_0 à tous les états finaux. Par la suite, pour des preuves ultérieurs, nous utilisons indifféremment $dist(q_0)$ et $dist(A)$ (A est un automate pondéré), car leurs usages ne conduit à aucune confusion

En effet, le calcul de la distance de l'Équation 6.15 n'est qu'une généralisation du problème classique du plus court chemin à source unique (Mohri 2002). Dans notre cas, l'automate pondéré $A_{s,t}$ est acyclique, alors nous avons combiné l'algorithme du plus court chemin à source unique avec le tri topologique pour atteindre une complexité $O(|A_{s,t}|)$, dans les conditions où les opérations de la somme et de produit du semi-anneau $\mathbb{S}$ sont en $O(1)$.

6.4 Amélioration de l'évaluation des noyaux de séquences

Dans cette section, nous proposons une méthode pour améliorer le calcul du noyau toutes sous-séquences avec trous. Ensuite, nous présentons une expérimentation pour soutenir la méthode proposée.

6.4.1 Intersection par anticipation

Les automates pondérés sont utilisés dans plusieurs domaines tels que la bioinformatique et le traitement automatique de la langue naturelle. Dans plusieurs cas, ils comportent un nombre important d'états et de transitions.
Par ailleurs, l'intersection des automates pondérés est une composante clé dans ces applications telle que l'évaluation des noyaux. Ainsi, un défi consiste à concevoir un algorithme efficace pour l'intersection des automates pondérés.

Notre idée principale se résume en une intersection par anticipation. Autrement dit, on ne doit pas emprunter des ϵ-chemins qui ne coïncident pas, par la suite, par des symboles communs de l'alphabet Σ. Pour ce faire, nous étendons

GASWA par des informations concernant les étiquettes des chemins successeurs, sans affecter substantiellement, l'efficacité de l'espace mémoire.

L'extension GASWA consiste à associer une structure *bitset*, *workbs* de taille $|\Sigma|$ à toutes ses transitions. Cette structure bitset est indexée par les symboles de Σ. La valeur vraie d'une entrée $workbs\,[a]$ d'une transition e témoigne l'existence du symbole $a \in \Sigma$ comme étiquette dans une ou plusieurs transitions des chemins avec comme origine l'état $d(e)$.

L'Algorithme 10 (qui fait appel à la fonction décrite dans l'Algorithme 11) effectue un parcours en profondeur (depth first search, DFS) de l'automate pondéré pour établir une structure bitset pour toutes les transitions. La complexité temporelle d'un tel algorithme est donc linéaire en fonction de la taille de l'automate GASWA et il utilise un extra espace mémoire (sans inclure la taille de GASWA) proportionnel au nombre des transitions.

Algorithm 10: estblish_bitset(A). Établir des structures bitset pour les transitions de l'automate GASWA.

Input: Un automate GASWA A d'un mot donné.
Output: Automate GASWA A étendu avec des structures bitset.

```
1 Q ← states_of(A)
2 for all q ∈ Q do
3     if not marked[q] then
4         process_state(q)
```

L'Algorithme 12 illustre l'intersection bitset des automates pondérés GASWA étendus. Il exploite le mécanisme d'anticipation pré-établi. Les lignes 12-15 sont impliquées lorsqu'il existe une correspondance entre des ϵ-transitions des automates pondérés. Même s'il existe une telle correspondance, nous n'empruntons pas ces chemins, sauf s'il existe un symbole commun de Σ dans les transitions successeurs. Ceci peut être vérifié avec une complexité temporelle booléenne en appliquant une intersection booléenne des structures bitset correspondantes.

6.4.2 Expérimentation

Le but de cette expérimentation est de montrer l'impact de notre intersection bitset sur la taille de l'automate résultat et par conséquent sur le temps d'exécution du noyau toutes sous-séquences avec trous.

Algorithm 11: process_state(q). Traitement des transitions à partir d'un état donné.

Input: Un état q.
Output: Établir des structures bitset pour les transitions à partir d'un état q.

```
1  marked[q] ← true
2  /* adj(q) est l'ensemble de toutes les transitions
   quittant l'état q                                    */
3  foreach t ∈ adj(q) do
4      if l(t) = ϵ then /*Empile la structure bitset de la ϵ-transition courante
       pour un traitement ultérieur*/
5          wbs ← bitset(t)
6          push(stackbs, wbs)
7          process_state(d(q))
8      else /*Dépile la structure bitset de la ϵ-transition récemment utilisée
       pour une mise à jour*/
9          wbs ← pop(stackbs)
10         wbs[l(t)] ← true
11         push(stackbs, wbs)
```

Les tests sont exécutés sur un processeur Intel Core i7 à 2.40 GHZ avec 16 Go de RAM sous Windows 10, 64 bit. Tous les algorithmes testés sont implémentés en Java.

En vue de conduire l'expérimentation, nous avons construit un ensemble de données à partir de la collection Reuters-21578 des articles de presse. Après les tâches de pré-traitement et la syllabification des mots, nous avons collecté 423 paires de documents représentés sous forme de séquences de syllabes.

Pour chaque paire de documents, nous construisons les automates GASWA associés qui seront utilisés par la suite pour évaluer le noyau toutes sous-séquences avec trous. En fait, nous construisons d'abord les automates pondérés résultats de l'intersection standard et celle d'anticipation (utilisant la structure bitset proposée) sur lesquels nous appliquons un algorithme généralisé du plus court chemin à source unique pour le calcul proprement dit du noyau.

Pour chaque paire de documents, nous enregistrons la taille ($|A| = |Q| + |E|$) et le temps d'exécution inhérent à l'automate pondéré résultat. La Table 6.2 résume les résultats de l'expérimentation en terme de la moyenne des quantités sus-citées, ceci en tenant compte de deux cas de figures. La première implique toutes les paires de documents, tandis que le second ne considère que les paires de documents où la valeur du noyau n'est pas assez importante (moins de 1000).

Algorithm 12: wa_intersect(A_s, A_t). Intersection des automates pondérés GASWA étendus avec des structures bitset.

Input: Automates GASWA étendus A_s et A_t.
Output: $A_{s,t} = A_s \cap A_t$.
/* $I_s = \{q_{0s}\}$ and $I_t = \{q_{0t}\}$ */
$I_{s,t} \leftarrow (q_{0s}, q_{0t})$
$\lambda_{s,t}(q_{0s}, q_{0t}) \leftarrow \lambda_s(q_{0s}) \cdot \lambda_t(q_{0t})$
$Enqueue(S, (q_{0s}, q_{0t}))$
while $S \neq \emptyset$ **do**
  $(q_s, q_t) \leftarrow Dequeue(S)$
  if $(q_s, q_t) \in F_s \times F_t$ **then**
    $F_{s,t} \leftarrow F_{s,t} \cup (q_s, q_t)$
    $\rho_{s,t}(q_s, q_t) \leftarrow \rho_s(q_s) \cdot \rho_t(q_t)$
  foreach $(e_s, e_t) \in Adj(q_s, q_t)$ **do**
    if $l(e_s) = l(et_t)$ **then**
      $borrow\_path \leftarrow true$
      if $l(e_s) = \epsilon$ **then**
        $bs_s \leftarrow bitset(e_s)$
        $bs_t \leftarrow bitset(e_t)$
        $borrow\_path \leftarrow bs_s \cap bs_t$
      if $borrow\_path$ **then**
        if $(d(e_s), d(e_t)) \notin Q_{s,t}$ **then** /*Transitions non traitées*/
          $Q_{s,t} \leftarrow Q_{s,t} \cup \{(d(e_s), d(e_t))\}$
          $Enqueue(S, (d(e_s), d(e_t)))$
        $E_{s,t} \leftarrow$
        $E_{s,t} \cup \{(q_s, q_t), l(e_s), weight(e_s) \cdot weight(e_t), (d(e_s), d(e_t))\}$

Il est clair que la taille de l'automate pondéré résultant de l'intersection par anticipation est moins que la moitié de la taille de celui résultant de l'intersection standard. Par conséquent le temps d'exécution de l'évaluation du noyau utilisant notre approche plus rapide que l'évaluation du noyau utilisant l'intersection standard.

Dans les deux situations, nous pouvons constater une amélioration relative de la taille de l'automate pondéré résultant de l'intersection par anticipation et par conséquent son impact sur le temps d'exécution de calcul du noyau, en particulier lorsque la valeur du noyau n'est pas assez grande. En fait, ce résultat est en harmonie avec l'explication naturelle stipulant que pour des séquences moins similaires, l'intersection standard des automates GASWA correspondants comprend plus de chemins non-réussis. Dans ce cas de figure, la tâche d'élagage

TABLE 6.2 – Performances de l'évaluation du noyau : Intersection par anticipation vs. Intersection Standard

	Nombre de paires de documents	**Valeur noyau**	**Taille moyenne**	**Temps d'exécution moyen (seconde)**
Standard	423	tout	126138	2.91
Anticipation			110336	2.27
Ratio (Anticipation/Standard)			87%	78%
Standard	171	< 1000	28488	0.58
Anticipation			17195	0.34
Ratio (Anticipation/Standard)			60%	58%

de notre approche est clairement visible (un gain, environ de 40% dans la taille moyenne et dans le temps d'exécution moyen).

Nous pouvons argumenter ce résultat comme suit : La correspondance d'un état q_i de l'automate GASWA A_s avec un état q_j de A_t sera rejetée par notre algorithme d'intersection par anticipation si les suffixes $s(i+1:|s|)$ et $t(j+1:|t|)$ sont disjoints. Ce cas est plus probable avec des alphabets de grande taille.

6.5 Relations avec d'autres noyaux de séquences

Dans le dessein de montrer l'aspect unification de notre modèle, dans la présente section nous nous focalisons sur l'exhibition de la relation entre certains noyaux de séquences couramment utilisés et notre noyau de séquence à base d'automates pondérés.

Noyau toutes sous-séquences

Le noyau toutes sous-séquences (Shawe-Taylor & Cristianini 2004) est introduit dans la section 3.4.4. Nous rappelons que $k(s,t)$ est défini comme suit :

$$k(s,t) = \sum_{u \in \Sigma^*} \sum_{(I,J):u=s(I)=t(J)} 1. \tag{6.16}$$

Proposition 6.6. *Soient s et t deux mots dans Σ^*, le noyau toutes sous-séquences $K(s,t)$ peut être défini comme un noyau de sous-séquences à base d'automates pondérés $dist(A_{s,t})$, $A_{s,t} = A_s \cap A_t$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t.*

Démonstration. Nous construisons les automates pondérés A_s et A_t associés respectivement à toutes les sous-séquences des mots s et t. Nous considérons le comportement de l'automate pondéré $A_{s,t} = A_s \cap A_t$ qui reflète la valeur du noyau $dist(A_{s,t})$:

$$\begin{aligned}
\|A_{s,t}\| &= \sum_{u \in \Sigma^*} (\|A_{s,t}\|, u)\ u. \\
&= \sum_{u \in \Sigma^*} (\|A_s\| \odot \|A_t\|, u)\ u \\
&= \sum_{u \in \Sigma^*} (\|A_s\|, u)(\|A_t\|, u)\ u && \text{Produit Hadamard} \\
&= \sum_{u \in \Sigma^*} \sum_{I:u=s(I)} \lambda^{l(I)} \sum_{J:u=t(J)} \lambda^{l(J)}\ u && \text{par définition} \\
&= \sum_{u \in \Sigma^*} \sum_{(I,J):u=s(I)=t(J)} \lambda^{l(I)+l(J)}\ u
\end{aligned}$$

Il suffit de mettre λ à 1, pour obtenir :

$$\|A_{s,t}\| = \sum_{u \in \Sigma^*} \sum_{(I,J):u=s(I)=t(J)} 1\ u. \tag{6.17}$$

La série formelle de l'équation (6.17) correspond exactement à la définition du noyau toutes sous-séquences $k(s,t)$ de l'équation (6.16). □

Noyau sous-séquences de mots

Le noyau sous-séquences de mots (Lodhi *et al.* 2002) est introduit dans la Section 3.4.6. Nous rappelons que $k_p(s,t)$ est défini comme suit :

$$k_p(s,t) = \sum_{u \in \Sigma^p} \sum_{(I,J):u=s(I)=t(J)} \lambda^{l(I)+l(J)}. \tag{6.18}$$

Proposition 6.7. *Soient s et t deux mots dans Σ^* et p la longueur des sous-séquences. Le noyau sous-séquences de mots $k_p(s,t)$ peut être défini comme un noyau de sous-séquences à base d'automates GASWA $dist(A)$, $A = A_s \cap A_t \cap A_{pgram}$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t et A_{pgram} est l'automate pondéré correspondant à tous les p-grams sur Σ^*.*

Démonstration. Nous construisons les automates GASWA A_s et A_t associés respectivement aux mots s et t et l'automate A_{pgram} représenté, à titre d'illustration pour $p = 2$, par la Figure 6.7. Nous considérons le comportement de l'automate pondéré $A = A_s \cap A_t \cap A_{pgram}$ qui reflète la valeur du noyau $dist(A)$:

$$\begin{aligned}
\|A\| &= \sum_{u \in \Sigma^*} (\|A\|, u)\ u. \\
&= \sum_{u \in \Sigma^*} (\|A_s\| \odot \|A_t\| \odot \|A_{pgram}\|, u)\ u \\
&= \sum_{u \in \Sigma^*} (\|A_s\|, u)(\|A_t\|, u)(\|A_{pgram}\|, u)\ u && \text{produit Hadamard} \\
&= \sum_{u \in \Sigma^*} \sum_{I:u=s(I)} \lambda^{l(I)} \sum_{J:u=t(J)} \lambda^{l(J)} \sum_{u \in \Sigma^p} 1\ u && \text{par définition} \\
&= \sum_{u \in \Sigma^*} \sum_{(I,J):u=s(I)=t(J)} \lambda^{l(I)+l(J)} \sum_{u \in \Sigma^p} 1\ u \\
&= \sum_{u \in \Sigma^p} \sum_{(I,J):u=s(I)=t(J)} \lambda^{l(I)+l(J)}\ u
\end{aligned} \tag{6.19}$$

La série formelle de l'équation (6.19) correspond exactement à la définition du noyau sous-séquences de mots $k_p(s,t)$ de l'équation (6.18). □

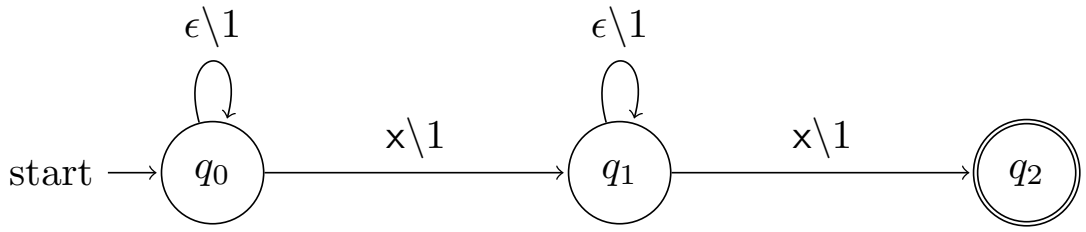


FIGURE 6.7 – Automate 2-$gram$ (A_{pgram}), x représente un élément de Σ.

Noyau sous-séquences de longueur fixe

Le noyau sous-séquences de longueur fixe (Shawe-Taylor & Cristianini 2004) est introduit dans la section 3.4.5. Nous rappelons que $k_p(s,t)$ est défini comme suit :

$$k_p(s,t) = \sum_{u \in \Sigma^p} \sum_{(I,J):u=s(I)=t(J)} 1. \tag{6.20}$$

Proposition 6.8. *Soient s et t deux mots dans Σ^* et p la longueur des sous-séquences. Le noyau sous-séquences de longueur fixe $K_p(s,t)$ peut être défini comme un noyau de sous-séquences (sans pénalisation) à base GASWA $dist(A)$, $A = A_s \cap A_t \cap A_{pgram}$, où A_s et A_t sont des automates GASWA associés respectivement aux mots s et t et A_{pgram} est l'automate pondéré correspondant à tous les p-grams sur Σ^*.*

Démonstration. La même preuve que celle du noyau sous-séquences de mots. Il suffit de mettre λ égale à 1. □

Noyaux Rationnels

Les noyaux rationnels sont aussi une plate-forme pour les noyaux de séquences. Ils utilisent les transducteurs pondérés et l'opération de composition pour définir de tels noyaux. Les transducteurs et l'opération de composition sont très utiles pour représenter et combiner différents niveaux de représentations. Par exemple, dans le traitement de la parole, il existe différents niveaux d'informations acoustiques, phonétiques et linguistiques (Pereira & Riley 1997). Par contre pour les noyaux, il existe un seul niveau d'information (les symboles de l'alphabet). Pour cette raison, nous jugeons que les automates pondérés correspondent étroitement aux méthodes à noyaux. Sinon, nous pouvons utiliser des

modèles de représentation des séquences plus puissants tels que les transducteurs algébriques, mais ceci peut affecter sensiblement la complexité de calcul des méthodes à noyaux.

Autres noyaux de séquences

Le modèle à base d'automate GASWA présente une plate-forme générale pour calculer les noyaux de séquences. Nous avons montré comment il est facile d'adapter les noyaux de séquences à base d'automates pondérés pour traiter une variété de noyaux de séquences. Cette adaptation peut être réalisée soit en modifiant le paramètre de pénalisation λ, soit en impliquant d'autres automates (tel que l'automate des p-gram) pour restreindre le calcul du noyau toutes sous-séquences avec trous.

En outre, il est facile de concevoir d'autres automates inhérents à un motif spécifique ($A_{pattern}$ sur un alphabet Σ) qui peut être utile pour certaines applications. Dans le but de calculer un nouveau noyau de séquences associé à un tel motif, il suffit d'effectuer l'opération d'intersection ($A_s \cap A_t \cap A_{pattern}$).

Par ailleurs, si l'on s'intéresse aux sous-mots (et non pas sur les sous-séquences), notre approche reste valide. Tout ce que nous avons à faire consiste à construire un automate pondéré pour tous les sous-mots. En effet, nous gardons la même construction que celle pour toutes les sous-séquences, mais nous devons éliminer les ϵ-transitions $e_i = (q_{i-1}, \epsilon, \lambda, q_i)$ $i = 1..n-1$ à partir du premier niveau tel que illustré dans la Figure 6.8.

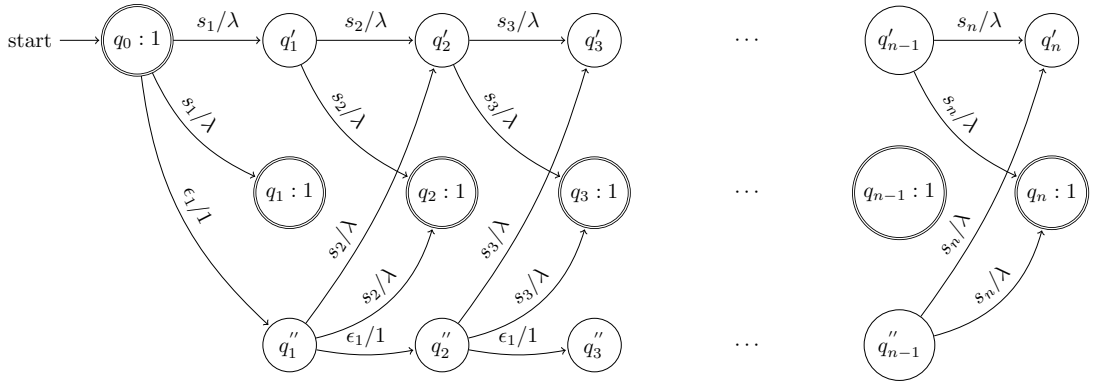


FIGURE 6.8 – Automate pondéré pour tous les sous-mots d'un mot $s = s_1 s_2 \ldots s_n$.

6.6 Généralisation pour un noyau d'ensembles de séquences

La pierre angulaire de notre plate-forme générale est la construction compacte de l'automate pondéré représentant toutes les sous-séquence d'un mot s, présentée dans la Section 6.2. Dans cette section, nous présentons une généralisation de cette idée pour un ensemble de mots afin de créer un nouveau noyau, nommé *noyau d'ensembles de séquences*.

L'idée clé derrière cette généralisation consiste à construire un automate pondéré préfixe (APP) représentant toutes les sous-séquences d'un ensemble de mots. Pour illustrer cette idée, nous considérons l'ensemble de mots $D = \{abb, abc, bcd, bce\}$. Le processus complet de notre construction est décrit comme suit :

- Construire un APP représentant un ensemble de mots. Chaque chemin dans un tel automate représente un mot dans D. Un APP assure une représentation compacte de l'ensemble D. Cette première étape permet de produire les états du premier niveau $q_0, \ldots, q_n$. Dans le pire des cas, $n = \sum_{s \in D} |s|$. Chaque transition est pondérée par le facteur de pénalisation λ. Il représente le poids de la distance entre deux symboles consécutifs. Ensuite, nous devons étendre le premier niveau par des ϵ-transitions pour permettre et pénaliser les trous.
  Finalement, nous renforçons chaque état par le nombre de passes par cet état. Une telle valeur représente le nombre d'occurrences du symbole de la transition incidente. La Figure 6.9 illustre cette construction.
  Par exemple, la valeur 4 emmagasinée dans l'état q_0 représente le nombre des ϵ dans l'ensemble D. Alors que le nombre de la sous-séquence a dans D est 2. Il est sauvegardé dans l'état q_1.
- Créer un deuxième niveau pour APP afin de générer les sous-séquences des mots de l'ensemble D. Ce niveau comporte les états finaux $q'_0, \ldots, q'_n$ avec les transitions $e_i = (q_i, l(e_i), \lambda, q'_{i+1}),\ i = 0..n-1$, où $l(e_i)$ est l'étiquette de la transition entre les états (q_i, q_{i+1}). Pour prendre en compte la multiplicité des sous-séquences, nous devons transférer les valeurs emmagasinées dans les états du premier niveau aux états associés du deuxième niveau comme des poids finaux. C'est à dire, chaque valeur sauvegardée dans

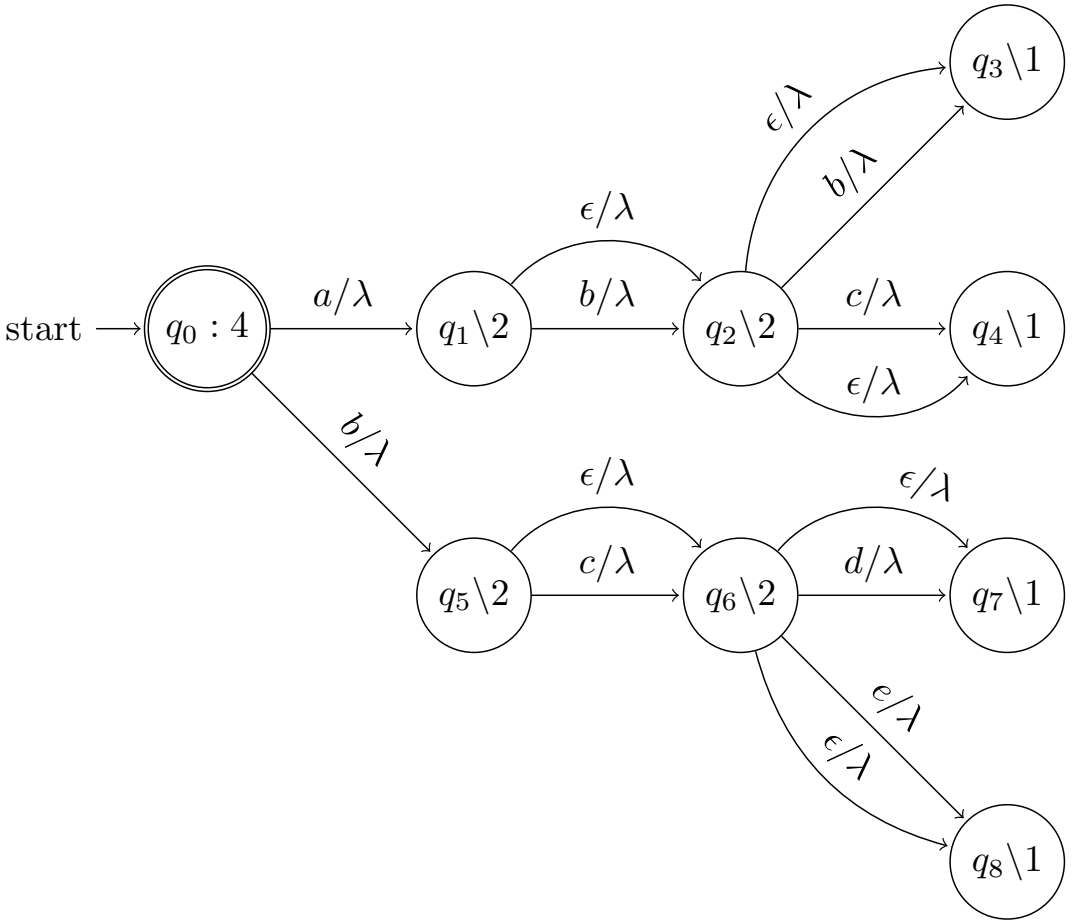


FIGURE 6.9 – Automate pondéré préfixe représentant l'ensemble de mots $D = \{abb, abc, bcd, bce\}$.

l'état $q_i,\ i = 1..n$ est assignée à l'état final q'_i. La Figure 6.10 montre ce processus.

- Construire les états de troisième niveau $q''_0, \ldots, q''_k$, où $k = \max\{|s| : s \in D\} - 1$. Pour commencer par n'importe quel symbole d'un mot dans D, nous devons créer les transitions $e_i = (q''_i, \epsilon, 1, q''_{i+1}),\ i = 0..k-1$. À partir de l'état $q''_i,\ i = 1..k$, nous pouvons décider de terminer une sous-séquence avec la transition $e_i = (q''_i, l(e_i), \lambda, q'_j)$, où $l(e_i)$ est une étiquette parmi les symboles dans la position $i+1$ dans un mot de l'ensemble D. Les états q'_j sont les états finaux correspondants. Dans notre exemple, nous pouvons construire deux transitions à partir de q''_1 aux états finaux, $e_{11} = (q''_1, b, \lambda, q'_2)$ et $e_{12} = (q''_1, c, \lambda, q'_6)$.
  Nous pouvons aussi continuer les sous-séquences à partir des états q_j à travers les transitions $e_i = (q''_i, l(e_i), \lambda, q_j)$, où $l(e_i)$ est l'étiquette parmi les

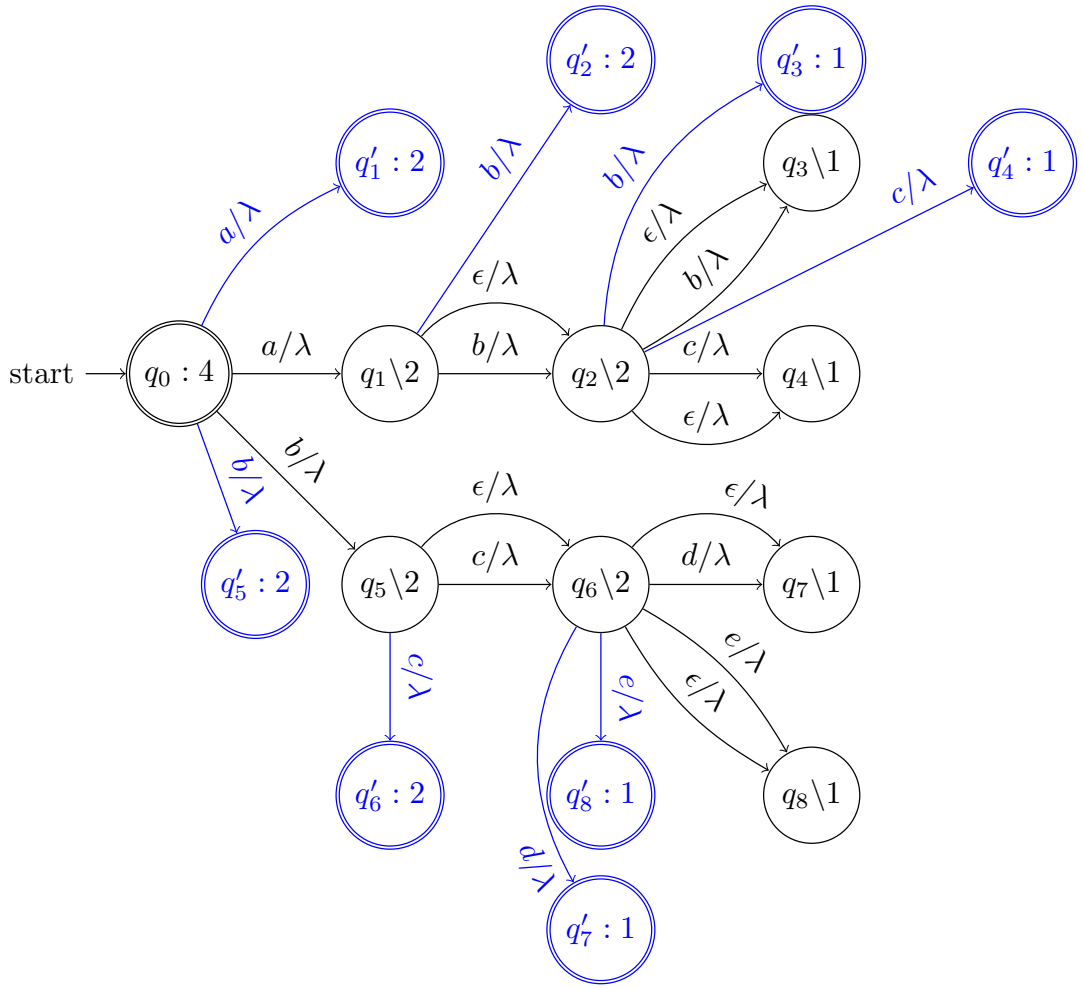


FIGURE 6.10 – Le APP renforcé par les états du deuxième niveau.

symboles à la position $i + 1$ dans un mot de l'ensemble D et q_j sont les états correspondants du premier niveau. Dans notre exemple, à partir de l'état q''_1, nous pouvons construire deux transitions $e_{11} = (q''_1, b, \lambda, q_2)$ et $e_{12} = (q''_1, c, \lambda, q_6)$. Le processus complet est illustré par la Figure 6.11.

Le processus, décrit ci-dessus, est l'algorithme de construction d'un APP représentant toutes les sous-séquences d'un ensemble de mots. En ce qui concerne le noyau d'ensembles de mots, notre approche à base d'automates pondérés demeure inchangée. Autrement dit, si nous disposons de deux ensembles de mots D_1 et D_2, nous devons effectuer, d'abord l'intersection $A_{D_1} \cap A_{D_2}$, où A_{D_i} représente toutes les sous-séquences de l'ensemble de mots D_i. Par la suite, nous appliquons l'algorithme de plus court chemin à source unique généralisé pour évaluer effectivement le noyau.

De plus, ce nouveau noyau peut être vu comme un noyau d'arbres, où les

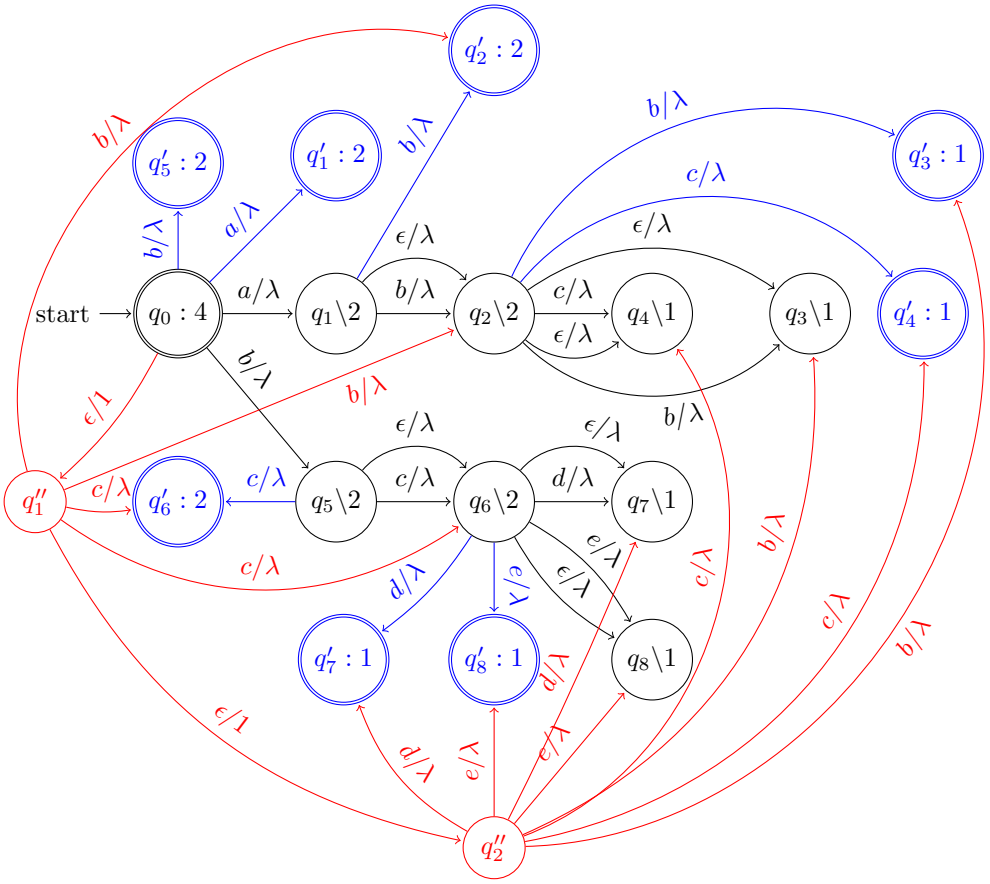


FIGURE 6.11 – Le APP final représentant toutes les sous-séquences des mots de l'ensemble $D = \{abb, abc, bcd, bce\}$.

sous structures sont des sous-chemins de l'APP. Autrement dit, des chemins permettant des « trous » (Il est possible de sauter des nœuds) avec un système de pénalisation.

Ce nouveau noyau pourrait être très utile dans de nombreuses tâches de l'apprentissage automatique, notamment quand nous avons à comparer deux ensembles d'objets. À titre d'exemple, dans la classification du texte, nous pouvons considérer un document comme un ensemble de phrases, de paragraphes ou toute autre granularité selon les applications.

6.7 Conclusion

Ce chapitre s'est voulu de répondre à la question de recherche « unification des méthodes de calcul des noyaux de séquences ».

Nous avons commencé par la présentation de quelques informations préliminaires inhérentes aux automates pondérés, la base de notre modèle d'unification.

Ensuite, nous nous sommes focalisés sur la présentation de notre approche dont le but consiste à développer une plate-forme générale pour calculer efficacement les noyaux de séquences. Nous avons commencé par puiser l'idée générale de notre modèle dont le processus de construction est décrit par la suite. Notre modèle est baptisé *GASWA* pour Gappy All Subsequence Weighted Automaton ; il est associée au noyau toutes sous-séquences avec trous.

Nous avons, ensuite, démontré la validité et l'efficacité de notre construction *GASWA* en utilisant les séries formelles.

Une fois le modèle construit, nous avons décrit le processus de calcul du noyau toutes sous-séquences avec trous. Afin d'améliorer le temps d'exécution de ce processus de calcul, nous avons proposé une nouvelle méthode d'intersection des automates pondérés dite *intersection par anticipation*. La méthode proposée est validée par une étude empirique dont les résultats révèlent une amélioration relative dans la taille de l'automate intersection, ainsi son impact sur le temps de calcul du noyau.

Dans le dessein de mettre en exergue l'aspect unification de notre modèle, nous avons démontré comment décrire les noyaux de séquences couramment utilisés à l'aide de notre modèle. Par ailleurs, nous avons décrit un scénario simple pour créer un nouveau noyau de séquences.

En fin, nous avons montré que notre modèle à base d'automates pondérés est généralisable pour un ensemble de mots donnant naissance à un nouveau noyau que nous avons nommé *noyau d'ensembles de séquences*. En effet, un tel noyau peut être vu comme un noyau d'arbre.

Chapitre 7

Conclusion et perspectives

Le but de cette thèse était de surmonter certains des inconvénients des méthodes à noyaux sur des données structurées. En particulier, nous avons affronté les problèmes d'efficacité et d'unification. D'une part, la complexité des noyaux de convolution est très élevée et ne permet pas le calcul de la fonction noyau sur des structures très complexes. Ceci peut empêcher leurs applications dans des scénarios réels. Le développement des techniques d'évaluation efficaces des fonctions noyaux reste un problème ouvert et c'est l'un des principaux objectifs de la présente thèse. D'autre part, l'existence d'une panoplie d'algorithmes pour l'évaluation des noyaux de mots et d'arbres, chacun avec sa propre technique. La proposition d'une plate-forme générale pour calculer les noyaux de séquences qui peut être utile pour la définition d'un nouveau noyau sur des *ensembles de séquences*, en tenant compte de l'efficacité de calcul qui est une propriété clé des méthodes à noyaux, constitue une seconde préoccupation de cette thèse.

Dans ce qui suit, nous rappelons brièvement les différentes étapes de la recherche, les éléments principaux des démarches entreprises pour répondre aux problématiques sus-citées ainsi que les perspectives de recherche.

Nous avons d'abord investigué dans les familles de noyaux de mots et d'arbres ainsi que l'analyse des algorithmes de leurs construction et les applications où ils y sont utilisés.

Nous avons développé une méthode d'évaluation efficace pour le noyau sous-séquence de mots (string subsequence kernel, SSK), qui est largement utilisé dans plusieurs tâches de l'apprentissage automatique. L'idée de base de notre approche consiste à réduire le calcul du noyau SSK à un problème géométrique connu sous le nom de requête d'intervalles (range query problem). Plus précisément, nous avons fait appel à un arbre d'intervalles en couches (layered range tree, LRT) dont nous avons appliqué les algorithmes de géométrie calculatoires correspondants.

Dans une perspective d'améliorer notre approche, nous avons étendu la structure de données arbre d'intervalles en couches (LRT) à un arbre d'intervalles de somme en couches (layered Range Sum Tree, LRST) doté des opérations d'agrégation. De plus, nous avons présenté des évaluations empiriques de notre approche contre l'approche à base de programmation dynamique et celle à base de programmation dynamique éparse, à la fois sur des données synthétiques et des données d'articles de presse. les résultats des expérimentations ont montré l'efficacité de notre approche pour les alphabets de grande taille, à l'exception des mots très courts.

Outre les différentes améliorations que nous avons cité, nous pouvons envisager pour notre approche géométrique deux extensions : en fait, il serait très intéressant d'étendre la nouvelle structure de données LRST de telle façon qu'elle soit dynamique. Cela peut nous aider à calculer le noyau SSK d'une manière incrémentale sans être obligé de créer à chaque fois une nouvelle LRST lors de l'évolution de la longueur des sous-séquences. Un autre axe intéressant consiste à combiner la structure de données LRST avec le paradigme de programmation dynamique. Nous estimons que l'utilisation des techniques d'intersection rectangulaire est une bonne piste, bien que cela semble être une tâche non triviale. De plus, nous estimons qu'il est intéressant d'explorer l'apprentissage profond (deep learning) pour les noyaux de mots et d'arbres.

En réponse à la deuxième problématique, nous avons utilisé le modèle des automates pondérés (weighted automata) pour représenter toutes les sous-séquences d'un mot. Le calcul du noyau toutes sous-séquences $K(s,t)$ entre deux mots s et t est le comportement de l'automate pondéré intersection $A_{s,t} = A_s \cap A_t$, où A_s et A_t correspondent, respectivement, à toutes les sous-séquences de s et t. Ainsi, tel qu'exprimé, il peut être généralisé pour un ensemble de séquences.

Pour une évaluation efficace du noyau, nous avons proposé une nouvelle technique d'intersection d'automates (intersection par anticipation) qui interdit l'emprunt des ϵ-chemins qui ne correspondent pas dans les automates A_s et A_t.

Les résultats des expérimentations ont révélé que l'évaluation du noyau toutes sous-séquences utilisant notre technique proposée est plus rapide que celle utilisant l'intersection standard.

Nous avons étudié aussi la relation entre notre plate-forme générale et une variété de noyaux de séquences. En fin, nous avons décrit un processus de création d'un nouveau *noyau d'ensembles de séquences* qui peut être vu comme un noyau d'arbre.

À titre de travail de recherche complémentaire pour notre nouveau *noyau d'ensembles de séquences*, nous proposons de l'évaluer dans une tâche d'apprentissage automatique où il est indispensable de faire une similarité entre des ensembles d'objets ou d'arbres. Nous conjecturons qu'un tel noyau va donner des résultats satisfaisants, à la fois sous la forme d'un noyau d'ensembles de séquences ou d'un noyau d'arbres.

Bibliographie

[Abu-Mostafa *et al.* 2012] Y.S. Abu-Mostafa, M. Magdon-Ismail et H.T. Lin. Learning from data : A short course. AMLBook.com, 2012.

[Allauzen & Mohri 2009] Cyril Allauzen et Mehryar Mohri. *N-Way Composition of Weighted Finite-State Transducers*. Int. J. Found. Comput. Sci., vol. 20, no. 4, pages 613–627, 2009.

[Althubaity *et al.* 2008] A. Althubaity, A. Almuhareb, S. Alharbi, A. Al-Rajeh et M. Khorsheed. *KACST Arabic Text Classification Project : Overview and Preliminary Results*. In Proceedings of The 9th IBIMA conference on Information Management in Modern Organizations, January 2008.

[Bellaouar *et al.* 2014] Slimane Bellaouar, Hadda Cherroun et Djelloul Ziadi. *Efficient List-Based Computation of the String Subsequence Kernel*. In Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings, pages 138–148, 2014.

[Bellaouar *et al.* 2017] Slimane Bellaouar, Hadda Cherroun, Attia Nehar et Djelloul Ziadi. *Weighted Automata Sequence Kernel*. In Proceedings of the 9th International Conference on Machine Learning and Computing, ICMLC 2017, pages 48–55, New York, NY, USA, 2017. ACM.

[Bellaouar *et al.* 2018] Slimane Bellaouar, Hadda Cherroun et Djelloul Ziadi. *Efficient geometric-based computation of the string subsequence kernel*. Data Mining and Knowledge Discovery, vol. 32, no. 2, pages 532–559, Mar 2018.

[Bentley & Maurer 1980] Jon Louis Bentley et Hermann A. Maurer. *Efficient Worst-Case Data Structures for Range Searching*. Acta Inf., vol. 13, pages 155–168, 1980.

[Bentley & Sedgewick 1997] Jon L. Bentley et Robert Sedgewick. *Fast Algorithms for Sorting and Searching Strings*. In Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '97, pages 360–369, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[Bentley 1975] Jon Louis Bentley. *Multidimensional Binary Search Trees Used for Associative Searching*. Commun. ACM, vol. 18, no. 9, pages 509–517, Septembre 1975.

[Bentley 1979] Jon Louis Bentley. *Decomposable Searching Problems.* Inf. Process. Lett., vol. 8, no. 5, pages 244–251, 1979.

[Berg *et al.* 2008] Mark de Berg, Otfried Cheong, Marc van Kreveld et Mark Overmars. Computational geometry : Algorithms and applications. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. édition, 2008.

[Blaschke & Valencia 2001] Christian Blaschke et Alfonso Valencia. *Can bibliographic pointers for known biological data be found automatically ? Protein interactions as a case study*. Comparative and Functional Genomics, vol. 2, no. 4, pages 196–206, 2001.

[Bloehdorn & Moschitti 2007] Stephan Bloehdorn et Alessandro Moschitti. Combined syntactic and semantic kernels for text classification, pages 307–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Bloehdorn 2008] Stephan Bloehdorn. *Kernel Methods for Knowledge Structures*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2008.

[Cancedda *et al.* 2003] Nicola Cancedda, Eric Gaussier, Cyril Goutte et Jean Michel Renders. *Word Sequence Kernels*. J. Mach. Learn. Res., vol. 3, pages 1059–1082, Mars 2003.

[Collins & Duffy 2002] Michael Collins et Nigel Duffy. *New Ranking Algorithms for Parsing and Tagging : Kernels over Discrete Structures, and the Voted Perceptron*. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02, pages 263–270, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[Cornuéjols & Miclet 2010] Antoine Cornuéjols et Laurent Miclet. Apprentissage artificiel : Concepts et algorithmes. Eyrolles, Juin 2010.

[Cortes *et al.* 2002] Corinna Cortes, Patrick Haffner et Mehryar Mohri. *Rational Kernels*. In Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada], pages 601–608, 2002.

[Cortes *et al.* 2004] Corinna Cortes, Patrick Haffner, Mehryar Mohri, Kristin Bennett et Nicolò Cesa-bianchi. *Rational kernels : Theory and algorithms*. Journal of Machine Learning Research, vol. 5, pages 1035–1062, 2004.

[Cristianini *et al.* 2002] N. Cristianini, J. Shawe-Taylor et H. Lodhi. *Latent Semantic Kernels*. Journal of Intelligent Information Systems, vol. 18, no. 2-3, pages 127–152, 2002.

[Culotta & Sorensen 2004] Aron Culotta et Jeffrey Sorensen. *Dependency Tree Kernels for Relation Extraction*. In Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[Da San Martino 2009] G. Da San Martino. *Kernel Methods for Tree Structured Data*. PhD thesis, University of Bologna, Padova, 2009.

[Drucker *et al.* 1999] Harris Drucker, Donghui Wu et V. N. Vapnik. *Support vector machines for spam categorization*. IEEE Trans. Neural Networks, vol. 10, no. 5, pages 1048–1054, 1999.

[Duda *et al.* 2001] Richard O. Duda, Peter E. Hart et David G. Stork. Pattern classification (2nd ed). Wiley, 2001.

[Haussler 1999] David Haussler. *Convolution Kernels on Discrete Structures*. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

[Hoare 1996] Tony Hoare. Unification of theories : A challenge for computing science, pages 49–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Hopcroft & Ullman 1979] J. Hopcroft et J. Ullman. Introduction to automata theory, languages, and computation. Addison-Wesley, 1979.

[Huang 2009] Te-Ming Huang. Kernel based algorithms for mining huge data sets : Supervised, semi-supervised, and unsupervised learning. Springer-Verlag, Berlin, Heidelberg, 2009.

[Isozaki & Kazawa 2002] Hideki Isozaki et Hideto Kazawa. *Efficient Support Vector Classifiers for Named Entity Recognition*. In Proceedings of the 19th International Conference on Computational Linguistics - Volume 1, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[Joachims & Cristianini 2001] Thorsten Joachims et Nello Cristianini. *Composite Kernels for Hypertext Categorisation*. In Proceedings of the International Conference on Machine Learning (ICML, pages 250–257. Morgan Kaufmann Publishers, 2001.

[Joachims 1998] Thorsten Joachims. *Text Categorization with Suport Vector Machines : Learning with Many Relevant Features*. In Proceedings of the 10th European Conference on Machine Learning, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag.

[Kandola *et al.* 2003] Jaz Kandola, John Shawe-taylor et Nello Cristianini. *Learning Semantic Similarity*. In NIPS, pages 657–664. MIT Press, 2003.

[Kashima & Koyanagi 2002] Hisashi Kashima et Teruo Koyanagi. *Kernels for Semi-Structured Data*. In Proc. of ICML, pages 291–298. Morgan Kaufmann, 2002.

[Kate & Mooney 2006] Rohit J. Kate et Raymond J. Mooney. *Using String-kernels for Learning Semantic Parsers*. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, ACL-44, pages 913–920, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[Kimura *et al.* 2011] Daisuke Kimura, Tetsuji Kuboyama, Tetsuo Shibuya et Hisashi Kashima. *A Subpath Kernel for Rooted Unordered Trees*. Transactions of the Japanese Society for Artificial Intelligence, vol. 26, no. 3, pages 473–482, 2011.

[Kuboyama *et al.* 2008] Tetsuji Kuboyama, Kouichi Hirata et Kiyoko F. Aoki-Kinoshita. *An Efficient Unordered Tree Kernel and Its Application to Glycan*

Classification. In Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings, pages 184–195, 2008.

[Kuich & Salomaa 1986] Werner Kuich et Arto Salomaa. Semirings, automata, languages. EATCS monographs on theoretical computer science. Springer-Verlag, Berlin, New York, Heidelberg, 1986.

[Kuksa *et al.* 2009] Pavel P. Kuksa, Pai hsi Huang et Vladimir Pavlovic. *Scalable Algorithms for String Kernels with Inexact Matching*. In D. Koller, D. Schuurmans, Y. Bengio et L. Bottou, editeurs, Advances in Neural Information Processing Systems 21, pages 881–888. Curran Associates, Inc., 2009.

[Lampert 2009] CH. Lampert. *Kernel Methods in Computer Vision*. Foundations and Trends in Computer Graphics and Vision, vol. 4, no. 3, pages 193–285, Septembre 2009.

[Leslie *et al.* 2002] C Leslie, E Eskin et W S Noble. *The spectrum kernel : a string kernel for SVM protein classification*. Pac Symp Biocomput, pages 564–575, 2002.

[Leslie *et al.* 2003] C. Leslie, E. Eskin et W. Noble. *Mismatch String Kernels for SVM Protein Classification*. In Neural Information Processing Systems 15, pages 1441–1448, 2003.

[Lo Conte *et al.* 2000] Loredana Lo Conte, Bart Ailey, Tim J. P. Hubbard, Steven E. Brenner, Alexey G. Murzin et Cyrus Chothia. *SCOP : a Structural Classification of Proteins database*. Nucleic Acids Research, vol. 28, no. 1, pages 257–259, 2000.

[Lodhi *et al.* 2002] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini et Chris Watkins. *Text classification using string kernels*. J. Mach. Learn. Res., vol. 2, pages 419–444, Mars 2002.

[Marsland 2009] Stephen Marsland. Machine learning : An algorithmic perspective. Chapman & Hall/CRC, 1st édition, 2009.

[Mitchell 1997a] Thomas M. Mitchell. Machine learning. McGraw-Hill, Inc., New York, NY, USA, 1 édition, 1997.

[Mitchell 1997b] Tom M. Mitchell. Machine learning. McGraw Hill series in computer science. McGraw-Hill, 1997.

[Mitchell 1998] M. Mitchell. An introduction to genetic algorithms. A Bradford book. Bradford Books, 1998.

[Mohri *et al.* 1996] Mehryar Mohri, Fernando C. N. Pereira et Michael Riley. *Weighted Automata in Text and Speech Processing*. In Proceedings of ECAI-96, Workshop on Extended finite state models of language, Budapest, Hungary. John Wiley and Sons, 1996.

[Mohri *et al.* 2012] M. Mohri, A. Rostamizadeh et A. Talwalkar. Foundations of machine learning. Adaptive computation and machine learning series. MIT Press, 2012.

[Mohri 2002] Mehryar Mohri. *Semiring Frameworks and Algorithms for Shortest-Distance Problems*. Journal of Automata, Languages and Combinatorics, vol. 7, no. 3, pages 321–350, 2002.

[Mooney & Bunescu 2006] Raymond J. Mooney et Razvan C. Bunescu. *Subsequence Kernels for Relation Extraction*. In Y. Weiss, P. B. Schölkopf et J. C. Platt, editeurs, Advances in Neural Information Processing Systems 18, pages 171–178. MIT Press, 2006.

[Moschitti 2006] Alessandro Moschitti. *Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees*. In ECML, pages 318–329. Machine Learning : ECML 2006, 17th European Conference on Machine Learning, Proceedings, September 2006.

[Murphy 2012] Kevin P. Murphy. Machine learning : A probabilistic perspective. The MIT Press, 2012.

[Naisbitt 1982] J. Naisbitt. Megatrends : Ten new directions transforming our lives. Numéro 158 de Megatrends : Ten New Directions Transforming Our Lives. Warner Books, 1982.

[Nehar *et al.* 2014] Attia Nehar, Abdelkader Benmessaoud, Hadda Cherroun et Djelloul Ziadi. *Subsequence kernels-based Arabic text classification*. In 11th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2014, Doha, Qatar, November 10-13, 2014, pages 206–213, 2014.

[Pereira & Riley 1997] Fernando Pereira et Michael Riley. *Finite State Language Processing*. In chapter Speech Recognition by Composition of Weighted Finite Automata. The MIT Press, 1997.

[Pfoh *et al.* 2013] Jonas Pfoh, Christian Schneider et Claudia Eckert. *Leveraging String Kernels for Malware Detection*. In Network and System Security : 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings, Lecture Notes in Computer Science. Springer, Juin 2013.

[Piskorski & Yangarber 2013] Jakub Piskorski et Roman Yangarber. *Information Extraction : Past, Present and Future*. In Thierry Poibeau, Horacio Saggion, Jakub Piskorski et Roman Yangarber, editeurs, Multi-source, Multilingual Information Extraction and Summarization, Theory and Applications of Natural Language Processing, pages 23–49. Springer Berlin Heidelberg, 2013.

[Rieck *et al.* 2008] Konrad Rieck, Ulf Brefeld et Tammo Krueger. *Approximate Kernels for Trees*. Rapport technique FIRST 5/2008, Fraunhofer Institute FIRST, September 2008.

[Rieck *et al.* 2010] Konrad Rieck, Tammo Krueger, Ulf Brefeld et Klaus-Robert Müller. *Approximate Tree Kernels*. Journal of Machine Learning Research, vol. 11, pages 555–580, 2010.

[Rousu & Shawe-Taylor 2005] Juho Rousu et John Shawe-Taylor. *Efficient Computation of Gapped Substring Kernels on Large Alphabets*. J. Mach. Learn. Res., vol. 6, pages 1323–1344, 2005.

[Sakarovitch & Thomas 2009] Jacques Sakarovitch et Reuben (traducteur) Thomas. Elements of automata theory. New York, N.Y. Cambridge University Press, 2009.

[Salton *et al.* 1975] G. Salton, A. Wong et C. S. Yang. *A Vector Space Model for Automatic Indexing*. Communications of the ACM, vol. 18, no. 11, pages 613–620, 1975.

[Samet 1990] Hanan Samet. The design and analysis of spatial data structures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Samuel 1959] A. L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. IBM J. Res. Dev., vol. 3, no. 3, pages 210–229, Juillet 1959.

[Saunders *et al.* 2002] Craig Saunders, Hauke Tschach et John S. Taylor. *Syllables and other String Kernel Extensions*. In Proc. 19th International Conference on Machine Learning (ICML'02), pages 530–537, 2002.

[Seewald & Kleedorfer 2007] Alexander K. Seewald et Florian Kleedorfer. *Lambda pruning : an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering*. Adv. Data Analysis and Classification, vol. 1, no. 3, pages 221–239, 2007.

[Shalev-Shwartz & Ben-David 2014] Shai Shalev-Shwartz et Shai Ben-David. Understanding machine learning : From theory to algorithms. Cambridge University Press, New York, NY, USA, 2014.

[Shawe-Taylor & Cristianini 2004] John Shawe-Taylor et Nello Cristianini. Kernel methods for pattern analysis. Cambridge University Press, New York, NY, USA, 2004.

[Shin & Kuboyama 2008] Kilho Shin et Tetsuji Kuboyama. *A generalization of Haussler's convolution kernel : mapping kernel*. In Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008, pages 944–951, 2008.

[Shin & Kuboyama 2010] Kilho Shin et Tetsuji Kuboyama. *A Generalization of Haussler's Convolution Kernel - Mapping Kernel and Its Application to Tree Kernels*. J. Comput. Sci. Technol., vol. 25, no. 5, pages 1040–1054, 2010.

[Siolas & d'Alche Buc 2000] Georges Siolas et Florence d'Alche Buc. *Support Vector Machines Based on a Semantic Kernel for Text Categorization*. In IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN), volume 5, pages 205–209. IEEE Computer Society, Washington, DC, USA, 2000.

[Strang 2009] Gilbert Strang. Introduction to linear algebra. Wellesley-Cambridge Press, Wellesley, MA, fourth édition, 2009.

[Sutton & Barto 1998] Richard S. Sutton et Andrew G. Barto. Reinforcement learning : An introduction. MIT Press, 1998.

[Turing 1995] A. M. Turing. *Computers &Amp ; Thought*. chapitre Computing Machinery and Intelligence, pages 11–35. MIT Press, Cambridge, MA, USA, 1995.

[Valencia & Blaschke 2002] Alfonso Valencia et Christian Blaschke. *The Frame-Based Module of the SUISEKI Information Extraction System*. IEEE Intelligent Systems, vol. 17, pages 14–20, 2002.

[Valiant 1984] L. G. Valiant. *A Theory of the Learnable*. Commun. ACM, vol. 27, no. 11, pages 1134–1142, Novembre 1984.

[Vapnik & Chervonenkis 1971] V. N. Vapnik et A. Ya. Chervonenkis. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. Theory of Probability & Its Applications, vol. 16, no. 2, pages 264–280, 1971.

[Vapnik & Chervonenkis 1974] V. N. Vapnik et A. Ya. Chervonenkis. Theory of pattern recognition. Nauka, Moscow, 1974.

[Vapnik 1982] V. N. Vapnik. Estimation of dependences based on empirical data : Springer series in statistics (springer series in statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Vishwanathan & Smola 2002] S. V. N. Vishwanathan et Alexander J. Smola. *Fast Kernels for String and Tree Matching*. In Proceedings of the 15th International Conference on Neural Information Processing Systems, NIPS'02, pages 585–592, Cambridge, MA, USA, 2002. MIT Press.

[Wong *et al.* 1985] S. K. M. Wong, Wojciech Ziarko et Patrick C. N. Wong. *Generalized Vector Spaces Model in Information Retrieval*. In Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '85, pages 18–25, New York, NY, USA, 1985. ACM.

[Yang & Wu 2006] Qiang Yang et Xindong Wu. *10 CHALLENGING PROBLEMS IN DATA MINING RESEARCH*. International Journal of Information Technology and amp ; Decision Making, vol. 05, no. 04, pages 597–604, 2006.

[Zaki *et al.* 2005] Nazar M. Zaki, Safaai Deris et Rosli Illias. *Application of String Kernels in Protein Sequence Classification*. Applied Bioinformatics, vol. 4, no. 1, pages 45–52, 2005.

[Zhang *et al.* 2007] Min Zhang, Wanxiang Che, AiTi Aw, Chew Lim Tan, Guodong Zhou, Ting Liu et Sheng Li. *A Grammar-driven Convolution Tree*

Kernel for Semantic Role Classification. In ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic, 2007.

[Zhang *et al.* 2008] Min Zhang, Wanxiang Che, Guodong Zhou, AiTi Aw, Chew Lim Tan, Ting Liu et Sheng Li. *Semantic Role Labeling Using a Grammar-Driven Convolution Tree Kernel*. IEEE Trans. Audio, Speech & Language Processing, vol. 16, no. 7, pages 1315–1329, 2008.