الجـــــــــمـهـوريـــــة الجـــــــــزائريـة الديمـــــــــــقراطيـة الشـــــــعبيـة
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
وزارة التـــــــــــــعليـم العـــــــالي و البـــــــــحث العـــــــلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جـــــــــامعة غـــردايـــــة
Université de Ghardaia
كليـة العلـــوم و التكنولـوجيـا
Faculté des Sciences et de la Technologie
قسم الرياضيات و الإعلام الآلي
Département des Mathématiques et l'Informatique

MÉMOIRE

Présenté pour l'obtention du **diplôme** de **MASTER**

En : Informatique

Spécialité : Systèmes Intelligents pour l'Extraction de Connaissances

Par : BOUKHARI Rachida & ZERGAT Nour Elhouda Cherifa

Sujet

Développement d'une plate-forme générale pour le calcul des noyaux de séquences

Soutenu publiquement le 25/06/2018 devant le jury composé de :

M. Slimane OULED NAOUI	MCA	Univ. Ghardaia	Président
M. Djelloul ZIADI	MAA	Univ. Ghardaia	Examinateur
M. Khaled KECHIDA	MAA	Univ. Ghardaia	Examinateur
M. Slimane BELLAOUAR	MAB	Univ. Ghardaia	Directeur de mémoire

Année Universitaire 2017/2018

ملخــص

يوفر التعلم الآلي طرقا ذكية يمكنها تحليل البيانات واستخراج المعلومات آليا من مجموعة البيانات الضخمة. يمكن تطبيق الطرق التقليدية للتعلم الآلي على البيانات الخطية. إلا أن هذا النوع من البيانات نادر، في الواقع نجد أن العديد من البيانات غير قابلة للفصل خطيا او يمكن تمثيلها في شكل مركب (سلاسل، اشجار، مخططات...).

إن طرق النواة هي طرق فعالة للتعامل مع هذه الانواع من البيانات. بالإضافة إلى ذلك، يتم استخدام نوى السلاسل على نطاق واسع في تحليل البيانات المتسلسلة. و في هذا الصدد فقد كرس الباحثون جهدا كبيرا لنوى السلاسل التي تركز على مشاكل محددة، وهذا يؤدي إلى مجموعة متنوعة من نوى السلاسل.

في هذه المذكرة، هدفنا هو تقديم وتنمية منصة عامة لحساب نوى السلاسل. في الواقع، يمكن اعتبار هذه المنصة بمثابة مساهمة في توحيد خوارزميات التعلم الالي.

من أجل تضمين جيد لتطبيقنا في بيئة تعلم آلي، اخترنا EMINK كمنصة تطوير.

كلمات مفتاحية

التعلم الآلي، نوى السلاسل، التوحيد، KNIME.

Résumé

L'apprentissage automatique fournit des méthodes intelligentes capables d'analyser des données et d'extraire des informations automatiquement à partir des grandes masses de données. Les méthodes classiques peuvent être appliquées aux données linéairement séparables. Toutefois, les problèmes répondant à ces contraintes sont rares. En effet, la majorité des problèmes disposant de données non linéament séparables ou des données qui peuvent être représentées sous forme structurée (séquences, arbres, graphes, ...). Les méthodes à noyaux est une approche efficace pour faire face à ces problèmes.

Par ailleurs, les noyaux de séquences sont largement utilisés dans l'analyse des données séquentielles. Les chercheurs ont consacré un grand effort aux noyaux de séquences en se focalisant sur des problèmes spécifiques. Ceci conduit à une variété de noyaux de séquences.

Dans ce mémoire, notre objectif consiste à présenter et implémenter une plate-forme générale pour calculer les noyaux de séquences. En effet, ceci peut être considéré comme une contribution à l'unification des algorithmes d'apprentissage.

Pour assurer une bonne intégration de notre implémentation dans un environnement d'apprentissage automatique, nous avons choisi KNIME comme plate-forme de développement.

Mots-clés :

Apprentissage automatique, noyau de séquence, unification, KNIME.

Abstract

Machine learning provides intelligent methods of analyzing data and to extract information automatically from large data. The traditional methods of machine learning can be applied to linearly separable data. Indeed, the majority of problems have non-linearly separable data and that can be represented in a structured form (sequences, trees, graphs,...). Kernel methods are an effecient approach to deal with these problems.
Moreover, sequence kernels are widely used in the analysis of sequential data. The researchers have devoted a great effort to sequence kernels that focus on specific problems. This leads to a variety of sequence kernels.
In this thesis, our goal is to present and implement a general platform for computing sequence kernels. In fact, this can be seen as a contribution to the unification of learning algorithms.
To ensure a good integration of our implementation into a machine learning environment, we have chosen KNIME as a development platform.

Key words
Machine learning, sequence kernel, unification, KNIME.

Dédicace

Je dédie ce modeste travail à :

Mes parents qui représentent pour moi l'exemple des sacrifices du dévouement, l'honnêteté et qui ont fait de moi ce que je suis devenu et qu'ils trouvent en ce mémoire l'expression de mon éternelle affection avec mon amour infini.

A mes frères **Abdelmalek** et sa femme **Sara**, **Sayeh**, **Rafik**, **Khalil** et mes soeurs **Fatima** et ses enfants et **Nacira**.

En témoignage de l'attachement, de l'amour et de l'affection que je porte pour vous.

Je vous dédie ce travail avec tous mes vœux de bonheur, santé et de réussite.

A mes tantes, oncles, cousins, cousines.

Et a tous les membres de mes grandes familles **BOUKHARI** et **DJALOUD**.

A mes chers amies **Khadidja**, **Meriem**, **Romissa**.

A tous mes amis et mes collègues.

Rachida

Dédicace

À l'homme de ma vie, mon exemple éternel, mon soutien moral et source de joie et de bonheur, celui qui s'est toujours sacrifié pour me voir réussir, que dieu te garde dans son vaste paradis, à **vous mon Père**.

À la lumière de mes jours, la source de mes efforts, la flamme de mon cœur, ma vie et mon bonheur, à **Mima que j'adore.**

Aux personnes dont j'ai bien aimé la présence dans ce jour, à tous mes frères **Brahim** pour sa voiture qui m'as beaucoup aidé, à **Smail**, **Youcef** et mes sœurs **Soumia**, **Hadjer**, **Fatna**, **Radia** et mes neveux **Taha** et **Iyad** et mes nièces **Lilia** et **Mayar**, à mes tantes, oncles, cousins, cousines et à ma chére binôme **Rachida**, je dédie ce travail dont le grand plaisir leurs revient en premier lieu pour leurs conseils, aides, et encouragements.

Aux personnes qui m'ont toujours aidé et encouragé, qui étaient toujours à mes côtés, et qui m'ont accompagnaient durant mon chemin d'études supérieures, mes aimables amis, collègues d'étude, et frères de cœur, toi **Amina**, **Noussiba**, **Safa**, **Rabia**, **Romissa**, **Maria** et **Zohra**.

Et un dédicace spécial à toutes l'équipe administratif au lycée Ben Ammar moulay abdallah et surtout à mes chers élèves que je les aime tellement.

Nour Elhouda Cherifa

Remerciements

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

En second lieu, nous tenons à remercier très chaleureusement **Dr. Slimane BELLAOUAR** qui nous a permis de bénéficier de son encadrement.
Les conseils qu'il nous a prodigué, la patience, la confiance qu'il nous a témoignés ont été déterminants dans la réalisation de notre travail de recherche.

À Monsieur **Khaled. M** pour ses conseils et ses idées qui nous ont aidés à accomplir ce travail.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail.

Nous tenons à exprimer nos sincères remerciements à tous les professeurs qui nous ont enseigné et qui par leurs compétences nous ont soutenu dans la poursuite de nos études.

Nous tenons à ne remercier toute personne qui a participé de près ou de loin à l'exécution de ce modeste travail.

Table des matières

Table des figures

Liste des tableaux

Introduction

L'internet a évolué au cours des deux dernières décennies, le nombre d'utilisateurs de ce réseau a augmenté de façon exponentielle. À la suite de cela, le volume de données numériques ne cesse de croître. Ces développements ont conduit au phénomène d'explosion des données. Toutefois, face à un tel volume de données, l'extraction de l'information pertinente est devenue complexe et couteuse. Pour répondre à cette problématique, les techniques d'apprentissage automatique permettent d'extraire automatiquement les connaissances à partir des collections de données massives. Les algorithmes d'apprentissage automatique ont été appliqués à divers domaines tel que le traitement du langage naturel et de la parole, bioinformatique, diagnostic médical ....

Toutefois, les outils classiques d'apprentissage automatique sont devenus inadaptés pour le traitement de ces données. Ils sont appliqués seulement aux données vectorielles. En pratique, de nombreuses applications contiennent des données qui peuvent être représentées sous une forme structurées (séquences, graphes, arbres, ...). Les méthodes à noyaux constituent des approches efficaces pour prendre en charge ce type de données aussi que ces type d'algorithmes connaissent un énorme succès depuis quelques années. Les noyaux sont des fonctions de similarité, ils peuvent être utilisés avec des algorithmes d'apprentissage linéaires pour extraire des relations non-linéaires.

Les séquences parmi les types de données plus importants rencontrée dans plusieurs domaines tel que la bioinformatique et le traitement de langage naturel.

Dans la littérature, il existe une variété de noyaux de séquences chacun est centré sur un problème spécifique. Il est judicieux donc de penser à un outil d'unification de ces noyaux de séquences.

Dans ce mémoire, nous nous focalisons sur les noyaux de séquences et sur la pré-

sentation d'une plate-forme générale pour unifier le calcul des noyaux de séquences. Ceci en tenant compte de l'efficacité de calcul qui est une propriété clé des méthodes à noyaux.

Nous commençons par l'investigation d'une famille de noyaux de séquences : le noyau p-spectre [Leslie et al., 2002], le noyau toutes sous- séquences, le noyau sous-séquences de longueur fixe [Shawe-Taylor and Cristianini, 2004] et le noyau sous-séquences de mots (SSK) [Lodhi et al., 2002].

Par la suite, nous présentons une description de la plate-forme générale proposée pour calculer les noyaux de séquence dont le principe se base sur le triplet : noyau, série formelle et automate pondéré [Bellaouar et al., 2017a].

Alors que 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 A_s représentant toutes les sous-séquences du mot s. Le 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 faciliter l'accès et l'utilisation de cette plate-forme générale de calcul de noyaux de séquences, nous optons de l'intégrer dans la plate-forme open source KNIME d'analyse de données. Par le développement de notre propre nœud KNIME qui permet de calculer d'une manière aisée une variété de noyaux de séquences, il est clair, qu'il devient facile d'intégrer les noyaux de séquences dans des tâches d'apprentissage automatique.

Notre mémoire est organisé comme suit :

Le chapitre 1 introduit la notion d'apprentissage automatique et sa relation avec les méthodes à noyaux. Dans le chapitre 2, un état de l'art couvre les noyaux de séquences ainsi que la plate-forme générale pour le calcul des noyaux de séquences. Le chapitre 3 est dédié pour le développement de notre propre nœud dans l'outil KNIME qui calcul une variété de noyaux de séquences.

En fin, la conclusion clôture notre mémoire avec une synthèse de travail ainsi que des perspectives.

Chapitre 1

Apprentissage automatique et méthodes à noyaux

L'apprentissage automatique (AA) est une branche à l'intersection de plusieurs disciplines : l'intelligence artificielle, l'algorithmique et la recherche d'information, qui analyse un ensemble de données afin de dériver des règles qui constituent des connaissances pour une nouvelle analyse, aussi il permet de réaliser des tâches qui semblent difficiles voire impossibles à réalisé par des moyens algorithmiques classiques.
Dans ce chapitre, nous présentons la définition de l'apprentissage automatique et ses différentes types et exposons un domaine important en grand lien avec l'apprentissage, à savoir les méthodes à noyaux pour donner un aperçu général du méthodes à noyaux.

1.1 Apprentissage automatique

En 1950, le mathématicien Turing a posé la question « Les machines peuvent-elles penser ?» [Turing, 1995]. Afin de répondre à la question, Turing imagina une expérience de pensée devenue célèbre et souvent considérée comme le meilleur moyen de cerner l'intelligence d'une machine. Il l'appela le « jeu de l'imitation ». Le test de Turing consiste à mettre un humain en confrontation verbale avec un ordinateur et un autre humain. Si la personne qui a initié les conversations est incapable de déterminer lequel des participants est un ordinateur, le programme d'ordinateur peut être considéré comme ayant réussi le test. Cela signifie que l'ordinateur et l'humain vont

essayer d'avoir une apparence humaine. Dès lors, les scientifiques ont commencé à confier à leurs ordinateurs des épreuves de plus en plus complexes. Nous pouvons distinguer trois grandes périodes pour le développement de l'apprentissage automatique :

• Modélisation des réseaux neuronaux et les techniques de décision théorique.

• Apprentissage symbolique orienté-concept.

• Système d'approches des connaissances intensives explorant des taches d'apprentissage variées.

Nous citons quelques définitions d'apprentissage automatique dans le contexte suivant :

Definition 1.1.1 *En 1959, Arthur Samuel [Samuel, 1959] invente le terme "machine learning" comme le champs d'étude qui donne aux machines la faculté d'apprendre sans être explicitement programmé.*

Definition 1.1.2 *Selon Tom Mitchell, en 1997 [Mitchell, 1997], un programme est en mesure d'apprendre à partir d'une expérience* **E** *par rapport à une tâche* **T** *et une mesure de performance* **P**, *si sa tâche* **T** *telle que mesurée par* **P**, *s'améliore avec l'expérience* **E**.

Pour illustrer ces notions sur un exemple concret, nous considérons la conception d'un filtrage anti-spam. Le problème consiste à déterminer si chaque courriel entrant est un message utile à lire ou non. 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 être évalué sa performance en calculant P, le pourcentage des courriels correctement classifiés.

Les techniques d'apprentissage automatique sont utilisées dans de nombreux domaines et convient en particulier au problème de la prise de décision automatique, à titre d'exemples :

✓ Diagnostic médical tel que l' expansion de cancers ;

✓ reconnaissance des formes telles que la reconnaissance de visages ;

✓ détection des activités frauduleuses ;

✓ systèmes robotique ;

✓ catégorisation de textes, par exemple classification d'e-mails, classification de pages web.

1.2 Types d'apprentissage automatique

Les types d'apprentissage automatique sont divisés en trois grandes familles selon le type d'apprentissage qu'ils emploient :

1. Apprentissage supervisé.
2. Apprentissage non-supervisé.
3. Apprentissage semi-supervisé.

1.2.1 Apprentissage supervisé

L'apprentissage supervisé (supervised learning en anglais) est une technique d'AA qui consiste à établir des règles de comportement à partir d'un ensemble de données contenant des exemples de cas déjà étiquetés. Plus précisément, cet ensemble est une paire d'entrées sorties (X, Y) qui se sont vus associés une classe par un expert. L'objectif est alors d'apprendre à prédire, pour toute nouvelle entrée X, la sortie Y. Ce type d'AA est illustré par la Figure 1.1.

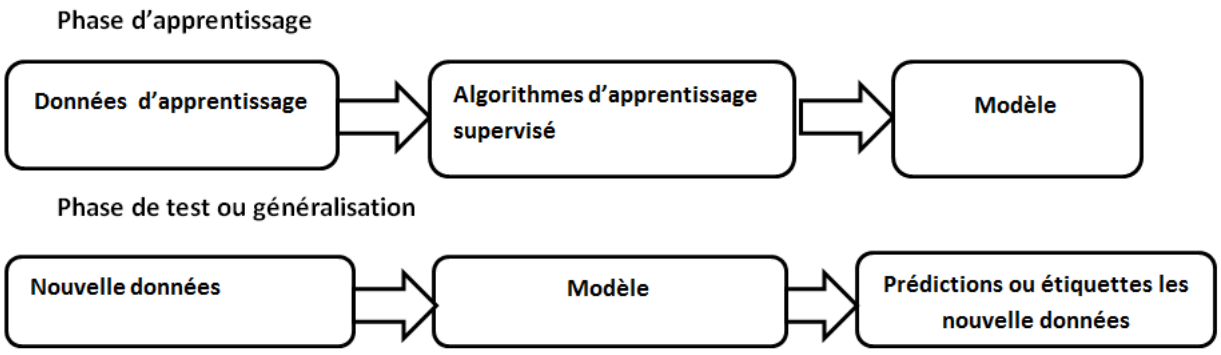


FIGURE 1.1 – Apprentissage supervisé. Extrait de [Huang, 2009] .

On distingue deux grandes familles des tâches réalisables en apprentissage supervisé à savoir : la classification et la régression, nous rappelons que nous parlons de la classification lorsque le label à prédire est discret, et nous parlons de la régression lorsque le label à prédire est continu.

Classification

La classification est une tâche très répandue dans l'AA supervisé. L'entrée X correspond à une instance d'une classe et la sortie Y qui est associée indique la classe. Les algorithmes d'AA de la technique de la classification s'appellent classifieurs. Par exemple, la reconnaissance de visage consistant à détecter un visage humain dans une image, l'entrée serait l'image numérique d'une personne, et la sortie indiquerait de quelle personne il s'agit (parmi l'ensemble de personnes que l'on souhaite voir le système reconnaître).

Régression

Lorsque l'entrée n'est pas associée à une classe mais dans le cas général à une ou plusieurs valeurs réelles (un vecteur), on parle alors de la régression.
Par exemple, on peut prédire le revenu des clients et de prospects en fonction de données comme leur : catégorie socioprofessionnelle, métier, adresse, âge, ou encore leur sexe.
Nous pouvons recueillir nos observations en faisant une enquête sur un panel de clients. Ensuite, nous utilisons une partie de ces observations pour entraîner un modèle capable de calculer ce revenu. Le reste du panel servira à mesurer la précision de l'algorithme en comparant une valeur réelle et une valeur prédite. Enfin, nous pouvons estimer le revenu des clients et prospects qui ne sont pas dans le panel.

1.2.2 Apprentissage non-supervisé

l'apprentissage non supervisé (parfois dénommé «clustering») est une méthode d'AA. Les données ne sont pas étiquetées, c'est à dire traiter le cas où l'on dispose seulement les entrées {X} sans les sorties {Y}. Il s'agit de regrouper les observations en différents groupes homogènes, en faisant en sorte que les données de chaque sous

ensemble partagent des caractéristiques communes.
Les données non étiquetées étant plus abondantes que les données étiquetées et les méthodes d'apprentissage automatique qui facilitent l'apprentissage non supervisé sont particulièrement utiles. Il y a deux types d'AA non-supervisé à savoir :

Partitionnement

Le partitionnement ou la segmentation est l'une des méthodes d'apprentissage non supervisée. Il est conçu pour regrouper les objets dans un petit nombres des classes homogènes et indépendantes (clusters), où chaque sous-ensemble possède des propriétés communes qui répondent souvent à des critères de proximité tels que la distance.
Il existe un large éventail de méthodes de partitionnement. Nous mentionnons à titre non exhaustif : les K-means, segmentation hiérarchique ascendante, segmentation hiérarchique descendante, ...

Règles d'association

Dans le domaine de l'extraction des données, la recherche des règles d'association est une méthode courante qui est étudiée de manière approfondie, dans le but de découvrir des relations d'intérêt statistiques entre deux ou plusieurs variables stockées dans des très grandes bases de données. Par exemple, l'analyse de panier du ménagères est l'une des applications typiques pour extraire des règles d'association. Elle permet d'analyser les tickets de caisse des clients afin de comprendre leurs habitudes de consommation. Ceci conduit à une meilleure organisation des promotions, optimiser la gestion des stocks, repenser la configuration des linéaires.
L'algorithme apriori [Agrawal et al., 1994] est le plus prévalent pour l'apprentissage des règles d'association.

1.2.3 Apprentissage semi-supervisé

L'apprentissage semi-supervisé fait partie de l'apprentissage automatique et il utilise un ensemble de données étiquetées et non-étiquetées. Il peut être considéré comme un apprentissage non supervisé avec des contraintes ou un apprentissage

supervisé avec des informations supplémentaires sur la distribution des exemples. Il se situe donc entre l'apprentissage supervisé et l'apprentissage non-supervisé. Il semble que les données non étiquetées, lorsqu'elles sont utilisées avec une petite quantité de données étiquetées, peuvent améliorer considérablement la précision de l'apprentissage.

La Figure 1.2 résume les différentes méthodes d'apprentissage automatique.

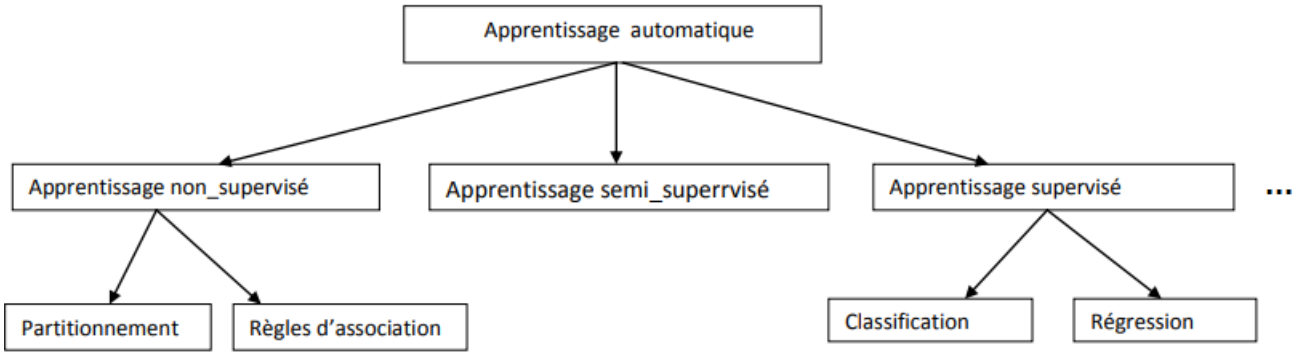


FIGURE 1.2 – Méthodes d'apprentissage automatique.

1.3 Algorithmes utilisés dans l'apprentissage automatique

L'existence de nombreux algorithmes d'apprentissage, nous fait poser la question « Quel algorithme d'apprentissage automatique devons-nous utiliser ? ». L'utilisation de tel ou tel algorithme dépend fortement de la tâche à effectuer (classification, estimation, ...). En effet, cela dépend aussi de la taille, de la qualité et de la nature des données.

Dans ce qui suit, nous présentons quelques algorithmes très connus dans l'apprentissage automatique.

K-plus proches voisins (K-PPV)

La méthode des plus proches voisins (noté parfois K-PPV ou K-NN pour (K-NearestNeighbor) est un algorithme très connue dans l'apprentissage supervisé, surtout dans le domaine de la classification des textes, il consiste à déterminer la liste des plus proches voisins parmi les individus déjà classés pour chaque nouvel individu que l'on veut classer. L'individu est affecté à la classe qui contient le plus d'individus parmi ces plus proches voisins selon une distances ou mesures de similarité (distance euclidienne par exemple). L'algorithme nécessite de connaître le nombre k de voisins [Hechenbichler and Schliep, 2004].
K-NN est simple à mettre en œuvre. Il ne fait aucun apprentissage, et stocke tout simplement tous les exemples d'apprentissage [Gupta et al., 2011].
En pratique, il est impossible de mettre en œuvre l'algorithme pour des dimensions élevées et des corpus d'exemples énormes. De plus, le coût de classification devient très élevé pour le plus proche voisin [Gupta et al., 2011].

Arbres de décision

Les arbres de décision (AD) [Quinlan, 1986] sont des structures simples très populaires en apprentissage supervisé. Ils sont des outils très puissants dans les cas où il est important de trouver des règles pour déterminer la classification. La structure des arbres est une représentation pour interpréter des résultats. Elle se compose d'un ensemble de trois catégories :

— Nœud racine à travers lequel l'arbre est accédé ;
— nœuds internes ayant des descendants ;
— nœuds terminaux (ou feuilles) : des nœuds qui n'ont pas de descendant.

Pour arriver à chaque nœud terminal, il y a une trajectoire qui commence par le nœud racine et il correspond à une règle de décision formée par une conjonction (ET) de plusieurs conditions de test.
Les principaux algorithmes des arbres de décision sont ID3 [Quinlan, 1986] et C4 [Salzberg, 1994].
Les arbres de décisions sont rapide en phase de construction. Ils sont également

simples à comprendre et à interpréter [Gupta et al., 2011].
Les arbres de décisions sont basés sur des heuristiques tels que l'algorithme glouton où les décisions sont prises à chaque nœud au niveau local [Gupta et al., 2011].

Réseaux de neurones

Les réseaux de neurones est une approche de l'apprentissage supervisé et de l'apprentissage non-supervisé. La conception de ces systèmes est inspiré du fonctionnement des neurones biologiques. Le réseau de neurone est généralement organisé en couches. Plus précisément, il est constitué de neurones connectés entre eux, de sorte que la sortie des neurones peut être une entrée pour un ou plusieurs autres neurones. En effet, la connexion entre ces neurones est dotée des poids. Chaque neurone est capable de réaliser quelques calculs élémentaires sur des données numériques [Christopher, 2016].
La Figure 1.3 représente la structure générale d'un réseau de neurone.

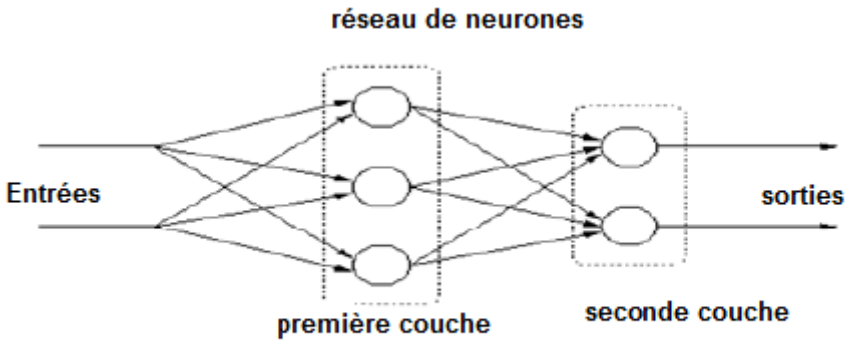


FIGURE 1.3 – Schéma général d'un réseau de neurone.

Les réseaux de neurones se présentent comme un modèle non linéaire, ce qui les rend flexibles dans la modélisation de relations des complexes dans le monde réel [Gupta et al., 2011].

1.4 Méthodes à noyaux

Les méthodes classiques de l'AA peuvent être appliquées sur des données linéairement séparables. Dans la pratique, de nombreuses applications requièrent des modèles non linéaires. Les méthodes à noyaux sont des techniques largement utilisées en apprentissage automatique. L'idée consiste à définir une fonction noyau qui permet de calculer un produit scalaire dans un espace de redescription des données de grande dimension. Dans cet espace implicite, les méthodes linéaires peuvent être utilisées.

1.4.1 Notion de noyau

Cette section est dédiée à la présentation de la fonction noyau et les concepts associés.

Fonction noyau

Soit $X = \{x_1, x_2, ...x_n\}$, un noyau est une fonction k que pour tout $x, x' \in X$ satisfaisant :

$$k(x, x') =< \phi(x), \phi(x') >,$$

où ϕ est une projection de X vers un espace de redescription F doté d'un produit scalaire :

$$\phi : x \mapsto \phi(x) \in F.$$

L'utilisation des fonctions noyaux comme un moyen de transformer la recherche de régularités non-linéaires en recherche de régularités linéaires grâce au passage par un espace de redescription virtuel F [Shawe-Taylor and Cristianini, 2004], fournit trois grandes idées découlant de l'approche générale :

1. Utiliser une application (projection) ϕ pour incorporer les exemples depuis un espace d'entrée X à un espace (vectoriel) de redescription F de haute dimension.
2. Dans F, les motifs (patterns) peuvent être découvertes en tant que relations linéaires.

3. Nous pouvons calculer le produit scalaire des exemples dans F directement à partir des données de l'espace d'entrée X en utilisant une fonction noyau [Shawe-Taylor and Cristianini, 2004].

La Figure 1.4 donne une illustration de ce principe à travers une transformation appliquée sur un jeu de données non linéairement séparable (Figure 1.4.a), pour lequel on peut facilement trouver un plan de séparation après reproduction sur le nouvel espace (Figure 1.4.b).

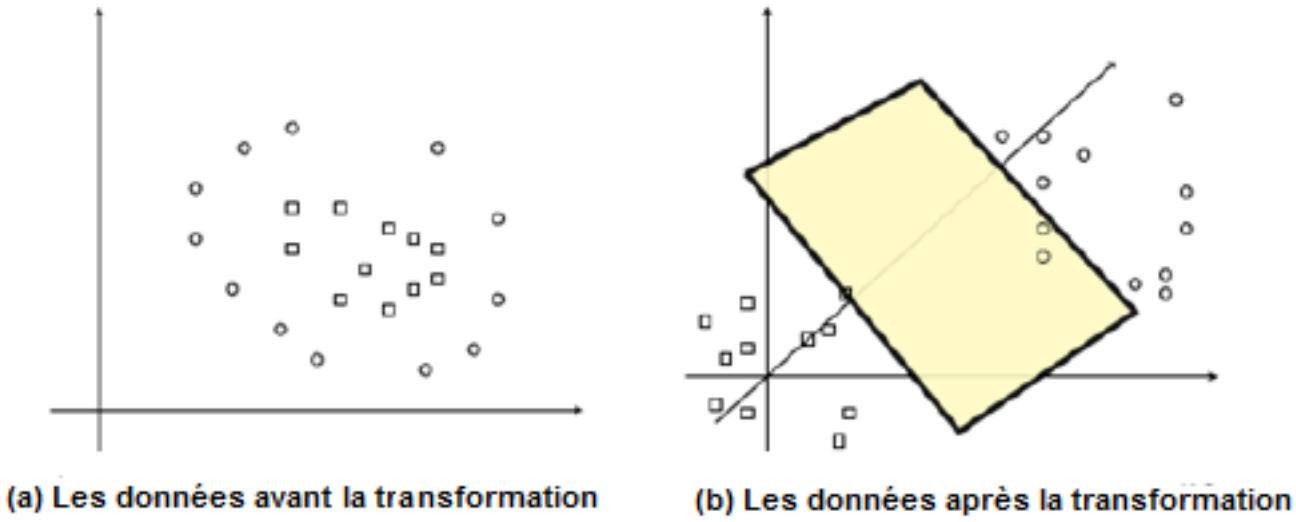


FIGURE 1.4 – Transformation non linéaire des données.

La Figure 1.5 montre la propriété de modularité offerte par les méthodes à noyaux. Dans un premier temps, les données sont traitées en utilisant un noyau pour créer une matrice de noyau ou une matrice du Gram. Ensuite, plusieurs algorithmes d'apprentissage peuvent être utilisés pour produire une fonction d'apprentissage. En d'autres termes, tous les noyaux peuvent être utilisés avec tout algorithme d'apprentissage.

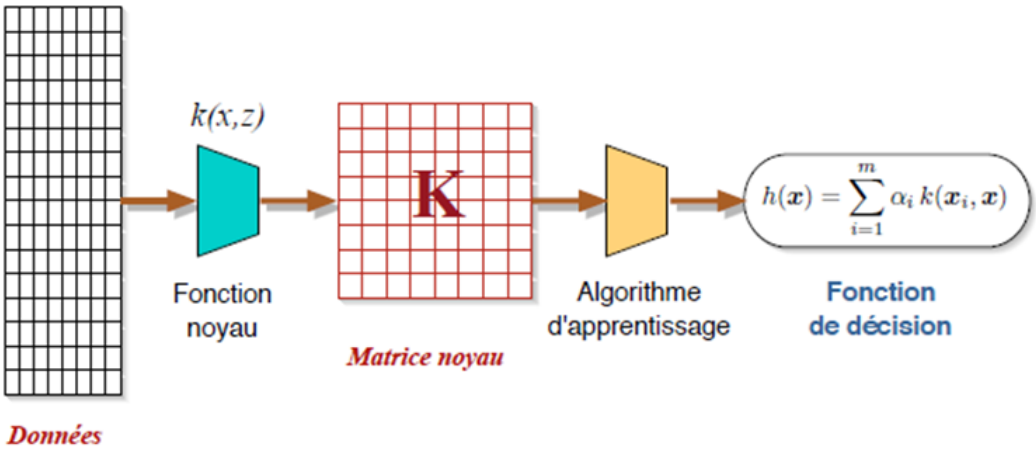


FIGURE 1.5 – Modularité dans les applications des méthodes à noyaux. Extrait de [Shawe-Taylor and Cristianini, 2004].

Noyau semi-défini positif

Il est possible de construire des nouvelles fonctions noyaux et de vérifier leur validité. Un noyau est valide si et seulement si k est un noyau semi-défini positif [Shawe-Taylor and Cristianini, 2004].
Un noyau $k : X^2 \to \mathbb{R}$ semi-défini positif est une fonction symétrique

$$\forall (x, x') \in X \times X; k(x, x') = k(x', x),$$

qui vérifie pour tout $n \in \mathbb{N} : \forall (c_i, ..., c_n) \in \mathbb{R}^n, \forall (x_1, ..., x_n) \in X^n, \sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$.

Matrice de Gram

On appelle matrice de Gram [Shawe-Taylor and Cristianini, 2004] d'un noyau k par rapport à un ensemble d'entrées $X = (x_1, x_2, ..., x_n)$ la matrice K de taille $n \times n$ définie de la façon suivante :

$$\forall (x_i, x_j) \in X \times X, K_{ij} = k(x_i, x_j), 1 \leq i, j \leq n.$$

1.4.2 Exemples de noyaux

Dans cette section nous donnons quelques exemples de noyaux [Shawe-Taylor and Cristianini, 2004] bien connus pouvant servir de base pour la définition de nouveaux noyaux :

• Le noyau linéaire est un produit scalaire simple : $k(x, x') =< x, x' >$.

• Le noyau polynomial permet de représenter des frontières de décision par des polynômes de degré d. La forme générique de ce noyau est $k(x, x') = (a\times < x, x' > +b)^d$.

• Le noyau gaussien est un noyau très utilisé dans l'AA, qui s'évalue selon :

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

Chapitre 2

État de l'art sur les noyaux de séquences

Les méthodes à noyaux ont été proposées comme une alternative aux problèmes de classification des données non linéairement séparables. Dans la pratique, de nombreuses applications disposent de données qui peuvent être représentées sous forme structurées (séquences, arbres, graphes, ...). Les méthodes à noyaux par leur propriété de modularité constituent des approches efficaces pour prendre en charge ce type de données. Les séquences sont parmi les types de données les plus importants rencontrées dans plusieurs domaines. Dans la bioinformatique, par exemple l'ADN est représenté sous forme de séquences des nucléotides. Par ailleurs, il existe une variété d'algorithmes qui traitent les séquences, chacun d'eux dépend de problèmes spécifiques, ce qui a conduit à une réflexion sur la théorie de l'unification. Les chercheurs ont longtemps travaillé dur 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]. La théorie de l'unification a réussi dans les sciences naturelles, la physique et les mathématiques modernes, mais la mise en œuvre de cette théorie dans l'informatique en particulier et dans le domaine de l'extraction des données est très difficile [Yang and Wu, 2006].

Dans les années passées, un grand effort a octroyé aux les noyaux de séquences centré sur des problèmes individuels, ce qui a donné naissance à de nombreuses approches. Afin d'atteindre le but de la théorie d'unification (au moins d'une manière partielle), l'idée consiste à proposer une plate-forme générale pour calculer les noyaux de séquences [Bellaouar et al., 2017a].

Tout d'abord, ce chapitre introduit quelques concepts de base nécessaires à la com-

préhension de la suite de ce chapitre.
Par la suite, une revue de la littérature des noyaux de séquences est présentée. Enfin, le chapitre est clôturé par la description de la plate-forme générale pour calculer les noyaux de séquences.

2.1 Préliminaires

Cette section sert à représenter les informations préliminaires dont nous avons besoin. Nous commençons par introduire les définitions des concepts mot, sous-mot et sous-séquence, puis nous rappelons quelques concepts de base que nous utilisons dans la plate-forme générale pour calculer les noyaux de séquences tels que les structures algébriques et les automates.

2.1.1 Mot et sous-mot

Le terme mot et séquence est utilisé parfois de manière indifférente en traitement automatique de langage naturel et en bioinformatique, ceci dit, un alphabet est un ensemble fini de symboles appelé Σ.
Un mot (string) $s = s_1 s_2 \dots s_{|s|}$ est une séquence finie de symboles de Σ y compris la séquence vide noté ε, la seule chaîne de longueur 0.
Nous notons Σ^n l'ensemble de tous les mots de longueur n, et par Σ^* l'ensemble de tout les mots :

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

Nous utilisons $[s = t]$ pour désigner la fonction qui renvoie 1 si les mots s et t sont identiques, et 0 sinon.
Pour les mots s et t, nous notons $|s|$ la longueur du mot s et st le mot obtenu par la concaténation des mots s et t.
le mot t est un sous-mot de s s'il existe (peut-être vide) des mots u et v telles que $s = utv$.
Si $u = \varepsilon$, on dit que t est un préfixe de s, alors que si $v = \varepsilon$, t est connu comme un

suffixe de s.
Pour $1 \leq i \leq j \leq |s|$, le mot $s(i:j)$ est le sous-mot $s_i \dots s_j$ de s. Les sous-mots de longueur k sont également appelées k-grams [Shawe-Taylor and Cristianini, 2004].

2.1.2 Sous-séquence

Nous disons que u est une sous-séquence (substring) du mot s, s'il existe des indices $I = (i_1, \dots, i_{|u|})$, avec $1 \leq i_1 < \dots < i_{|u|} \leq |s|$, tel que $u_j = s_{i_j}$, pour $j = 1, \dots, |u|$.
Nous dénotons par $|I| = |u|$ le nombre d'indices dans la séquence, tandis que la longueur $l(I)$ de la sous-séquence est $l(I) = i_{|u|} - i_1 + 1$, c'est-à-dire le nombre de symboles de mot s couvèrent par la sous-séquence u [Shawe-Taylor and Cristianini, 2004].
À titre d'exemple, soit l'alphabet $\Sigma = \{a, b, \dots, z\}$, considérons le mot $s =$ "$kernels$". Nous avons $|s| = 7$, alors que $s(1:3) =$ "ker" est un préfixe de s, $s(4:7) =$ "$nels$" est un suffixe de s, et avec $s(2:5) =$ "$erne$" tout les trois sont des sous-mots de s. $s(1,2,4,7) =$ "$kens$" est une sous-séquence de longueur 4, dont la longueur $l(1,2,4,7)$ dans s est 7. Un autre exemple de sous-séquences est $s(2,4,6) =$ "enl". avec $l(2,4,6) = 5$.

2.1.3 Monoïde

Un monoïde est une structure algébrique qui peut être définie comme un ensemble avec une loi de composition interne associative et d'un élément neutre. Soit un alphabet Σ est un ensemble non vide des symboles appelés lettres. On peut concaténer des mots à partir de ces lettres, l'opération de concaténation $(\cdot)$ est associative, la structure $(\Sigma^*, \cdot, \varepsilon)$ est le monoïde libre [Kuich and Salomaa, 1986].

2.1.4 Semi annaux

Le semi annaux [Kuich and Salomaa, 1986] est une structure algébrique $(\mathbb{S}, +, \cdot, 0, 1)$ possédant les propriétés suivantes :

1. $(\mathbb{S}, +, 0)$ est un monoïde commutatif ;

2. $(\mathbb{S}, \cdot, 1)$ forme un monoïde ;
3. l'opération $(.)$ est distributive par rapport à $(+)$;
4. pour tout $a \in \mathbb{S} : 0 \cdot a = a \cdot 0 = 0$.

2.1.5 Automate sur un alphabet

Un automate A sur un alphabet Σ est défini par le quintuplet (A, Q, E, I, F) constitué des éléments suivants :

1. Q est un ensemble fini d'états.
2. $E \subseteq Q \times \Sigma \times Q$ est un ensemble de transitions.
3. $I \subseteq Q$ est un ensemble d'états initiaux.
4. $F \subseteq Q$ est l'ensemble des états finaux (ou acceptant).

Un chemin est une séquence de transitions. Il est dit réussi s'il commence par un état initial et se termine par un état final [Kuich and Salomaa, 1986].

2.1.6 Automate pondéré

Soit A un automate pondéré (en anglais *weighted automaton* (WA)) sur un semi-anneau $\mathbb{S}$ est une 7-tuple $A = (\Sigma, Q, E, I, F, \lambda, \rho)$ où Σ est un alphabet, Q est un ensemble fini d'états, E un sous-ensemble fini $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{S} \times Q$ qui sont les transitions de A, $I \subseteq Q$ l'ensemble d'états initiaux, $F \subseteq Q$ est l'ensemble d'états finaux, λ: $I \to \mathbb{S}$ la fonction de poids initiaux, ρ: $F \to \mathbb{S}$ la fonction de poids finaux.
Pour une transition $e = (p,\ a,\ s,\ q) \in E$, on appelle $o(e) = p$ son origine ou l'état précédent, $l(e) = a$ son étiquette, $wt(e) = s$ son poids et $d(e) = q$ son état de destination.
Un chemin (*path*) $\pi = e_1 e_2 \dots e_k$ est une séquence de transitions. Le poids du chemin π est $wt(\pi) = wt(e_1) \cdot wt(e_2) \dots \cdot wt(e_k)$. Un chemin réussi (calcul) d'un A est un chemin d'un état initial à un état final [Kuich and Salomaa, 1986].

2.1.7 Transducteur pondéré

Un transducteur pondéré $\mathbb{T}$ sur un semi-anneau $\mathbb{S}$ est défini par un 8-tuple $\mathbb{T} = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ où Σ est l'alphabet d'entrée, Δ est l'alphabet de sortie, Q est un ensemble fini d'états du transducteur, $I \subseteq Q$ l'ensemble des états initiaux, $F \subseteq Q$ l'ensemble des états finaux, $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times \mathbb{S} \times Q$ un ensemble finis de transitions, $\lambda : I \to \mathbb{S}$ est la fonction de pondération des états initiaux, et $\rho : F \to \mathbb{S}$ est la fonction de pondération des états finaux [Mohri, 2009].

2.1.8 Série formelle

Soient Σ un alphabet et $\mathbb{S}$ un semi-anneau, on appelle série formelle r une application de Σ^* dans $\mathbb{S}$. L'image par r d'un mot $w \in \Sigma^*$ notée (r, w) est appelée le coefficient de w dans r. La série r, elle-même, est écrite comme une somme formelle :

$$r = \sum_{w \in \Sigma^*} (r, w) w.$$

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. $\mathbb{S}(\Sigma)$ désigne l'ensemble des polynômes [Kuich and Salomaa, 1986].

2.2 Noyaux de séquences

Les séquences sont considérées comme des données structurées. Les données structurées sont des objets discrets qui peuvent être décomposés tels que les arbres et les graphes.

Les séquences sont généralement rencontrées dans les domaines liés à la bioinformatique et dans le traitement automatique de langages naturels. Les noyaux de séquences sont largement utilisées dans l'analyse de données séquentielles.

Nous passons en revue les principaux types de noyaux proposés ces dernières années pour le traitement des séquences.

2.2.1 Noyau *p*-spectre

Le noyau p-spectre (ou p-gram) [Leslie et al., 2002] est un noyau simple pour le traitement de séquences. Il permet d'évaluer le nombre de sous-séquences contigües de taille p que deux séquences ont en communs. On définit le spectre d'ordre p (ou p-spectre) d'une séquence s comme l'histogramme des fréquences de tout ses sous-mots (contigües) de longueur p.
La contigüité lors de la comparaison de deux mots en utilisant le noyau p-spectre joue un rôle important pour calculer la similarité [Aseervatham, 2007]. Nous pouvons définir un noyau comme le produit scalaire de leurs p-spectres.
L'espace de redescription F associé au noyau p-spectre est indexé par $I = \Sigma^p$. Avec

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

Le noyau p-spectre est alors défini par :

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

Par exemple pour les deux mots construits sur l'alphabet A,C,G,T :
s = GAGTTCTAAT.
t = GGATCACTAA.
Les sous-mots de longueur p=3 présentes dans ces deux mots sont respectivement :
GAG, AGT, GTT, TTC, TCT, CTA, TAA, AAT
GGA, GAT, ATC, TAC, CAC, ACT, CTA, TAA
Les sous-mots en commun sont : CTA et TAA, on a $\Phi^3_{CTA}(s) = 1$, $\Phi^3_{TAA}(s) = 1$, $\Phi^3_{CTA}(t) = 1$, $\Phi^3_{TAA}(t) = 1$, ce qui donne un produit scalaire $k_3(s,t) = 2$.
La complexité de ce noyau est : $O(|\Sigma^p|)$ [Shawe-Taylor and Cristianini, 2004].

Évaluation avec le noyau auxiliaire

A l'aide du noyau auxiliaire connu sous le nom de noyau k-suffixe, on peut évaluer le calcul du noyau p-spectre.
Le noyau k-suffixe $k_k^S(s,t)$ est défini par :

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

Certainement, l'évaluation de $k_k^S(s,t)$ nécessite $O(k)$ comparaisons. Par conséquente, le noyau p-spectre devient en fonction de la version k-suffixe :

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

L'évaluation de $k_p(s,t)$ peut se réaliser dans un temps $O(p|s||t|)$ [Shawe-Taylor and Cristianini, 2004].

À titre d'exemple, l'évaluation du noyau à 3-spectre entre les deux mots s= " statistics" et t= "computation" en utilisant le noyaux 3-suffixe est illustrée dans les Tables 2.1 et 2.2.

TABLE 2.1 – Noyau 3-suffix entre les mots s et t.

$DP : K_3^S$	ε	c	o	m	p	u	t	a	t	i	o	n
ε	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	1	0	0	0
i	0	0	0	0	0	0	0	0	0	1	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0

2.2.2 Noyau toutes sous-séquences (All-subsequences)

Le noyau toutes sous-séquences [Shawe-Taylor and Cristianini, 2004] prend en considération toutes les sous-séquences non contigües de toute taille, ceci peut être considéré comme un avantage pour ce noyau car il est capable de capturer tous les

TABLE 2.2 – Noyau 3-spectre entre les mots s et t [Shawe-Taylor and Cristianini, 2004].

$DP : k_3$	ε	c	o	m	p	u	t	a	t	i	o	n
ε	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	1	1	1	1
i	0	0	0	0	0	0	0	0	1	2	2	2
s	0	0	0	0	0	0	0	0	1	2	2	2
t	0	0	0	0	0	0	0	0	1	2	2	2
i	0	0	0	0	0	0	0	0	1	2	2	2
c	0	0	0	0	0	0	0	0	1	2	2	2
s	0	0	0	0	0	0	0	0	1	2	2	2

motifs communs à deux séquences. L'espace de redescription associé est indexé par Σ^*.

Avec :

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

C'est bien le nombre d'occurrences du mot u comme une sous-séquences dans le mot s.

Le noyau associé :

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle = \sum_{u \in \Sigma^*} \Phi_u(s)\Phi_u(t). \tag{2.1}$$

L'espace de redescription est infini entraînant un temps de calcul important.

Par exemple, la projection des mots "bar", "baa", "car" et "cat" dans l'espace de redscription (toutes sous-séquences) est données par la Table 2.3.

La matrice noyau est représenter dans la Table 2.4.

TABLE 2.3 – Projection des mots "bar","baa","car" et "cat" dans l'espace de redescription (toutes sous-séquences).

ϕ	ϵ	a	b	c	r	t	aa	ar	at	ba	br	bt	ca	cr	ct	bar	baa	car	cat
bar	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0	0
baa	1	2	1	0	0	0	1	0	0	2	0	0	0	0	0	0	1	1	0
car	1	1	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0
cat	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1

TABLE 2.4 – La matrice noyau toutes sous-séquences, des mots "bar", "baa", "car" et "cat".

K	bar	baa	car	cat
bar	8	6	4	2
baa	6	12	3	3
car	4	3	8	4
cat	2	3	4	8

Évaluation du noyau toutes sous-séquences

Le calcul explicite du produit scalaire dans cet espace de dimension infinie devient infini. D'où la nécessite d'une méthode plus efficace (récursive) :

$$\Phi_u(s)\Phi_u(t) = \sum_{I:u=s(I)} 1 \sum_{J:u=t(J)} 1 = \sum_{(I,J):u=s(I)=t(J)} 1. \tag{2.2}$$

En utilisant le résultat de l'Équation (2.2), le noyau toutes sous-séquences peut être reformulé comme suit :

$$k(s,t) = \sum_{u\in\Sigma^*} \sum_{(I,J):u=s(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1. \tag{2.3}$$

Si l'on ajoute un symbole a au mots s, le nombre de sous-séquence communes peut être donné par :

$$\sum_{(I,J):sa(I)=t(J)} 1 = \sum_{(I,J):s(I)=t(J)} 1 + \sum_{u:t=uav} \sum_{(I,J):s(I)=u(J)} 1. \tag{2.4}$$

L'Équation 2.4 mène à une définition récursive des noyaux toutes sous-séquences.

$$\begin{cases} k(s,\varepsilon) = 1 \\ k(sa,t) = k(s,t) + \sum_{j:t_j=a} k(s,t(1:j-1)). \end{cases}$$

Grâce à la propriété de la symétrie des noyaux, nous pouvons définir une relation de récurrence analogue qui peut être donnée pour $k(s, ta)$:

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

Par exemple, nous considérons le calcul de noyau entre les mots s= "gatt" et t= "cata", où nous ajoutons le symbole a = "a" au mot "gatt" pour obtenir le mot sa = "gatta". Les rangées des tables sont indexées par les couples (I, J) tels que $sa(I) = t(J)$, avec ceux qui n'impliquent pas le symbole final de sa ont énumérés dans la Table 2.5, tandis que ceux impliquant le symbole final sont données par la Table 2.6.

TABLE 2.5 – Les sous-séquences qui ne terminent pas par le symbole "a" entre les mots s= "gatta" et t = "cata".

g a t t a	$sa(I) = t(J)$	c a t a
0 0 0 0 1	ε	0 0 0 0
0 0 0 0 1	at	0 1 1 0
0 1 0 1 0	at	0 1 1 0
0 1 0 0 0	a	0 1 0 0
0 0 1 0 0	t	0 0 1 0
0 0 0 1 0	t	0 0 1 0
0 1 0 0 0	a	0 0 0 1

TABLE 2.6 – Les sous-séquences qui se terminent par le symbole "a" dans les mots s = "gatta" et t = "cata".

g a t t a	$sa(I) = t(J)$	c a t a
0 0 0 0 0	a	0 1 0 0
0 1 1 0 0	a	0 0 0 1
0 1 0 0 1	aa	0 1 0 1
0 0 0 1 1	ta	0 0 1 1
0 0 1 0 1	ta	0 0 1 1
0 1 1 0 1	ata	0 1 1 1
0 1 0 1 1	ata	0 1 1 1

Par conséquent, nous voyons que les 14 lignes impliquant que $k(sa,t) = 14$ sont constituées de 7 lignes de la première table donnant $k(s,t) = 7$, plus $k(\text{"}gatt\text{"},\text{"}c\text{"}) = 1$ donnée par la première rangée de la deuxième table et $k(\text{"}gatt\text{"},\text{"}cat\text{"}) = 6$ correspondant aux lignes 2 à 7 de la deuxième table.
Une évaluation récursive de ce noyau est très couteuse, d'où le recours à la technique de la programmation dynamique qui consiste à sauvegarder les résultats intermédiaires dans une table de programmation dynamique (Table 2.7) dont chaque ligne correspond à un symbole du mot s et chaque colonne correspond à un symbole du mot t.

TABLE 2.7 – Table de programmation dynamique pour le calcul d'un noyau.

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

Les entrées $k_{ij} = k(s(1:i);t(1:j))$ donnent les évaluations des noyaux pour les préfixes de deux mots. La première ligne et la première colonne sont données par le cas de base de la récurrence.
La formule de récurrence nous permet de calculer l'entrée (i,j) comme la somme des entrées $(i-1,j)$ avec toutes les entrées $(i-1,k-1)$ avec $1 \leq k < j$ dont $t_k = s_i$.
Le nombre des opérations pour évaluer l'entrée (i,j) est $O(j)$, puisque nous devons vérifier tous les symboles de t jusqu'à la position j. Par conséquent, pour remplir le tableau complet, ceci nécessitera $O(p|s||t|^2)$ d'opérations [Shawe-Taylor and Cristianini, 2004].
Nous prenons le même exemple des mots s = "gatt" et t = "cata", la table de la programmation dynamique pour le calcul d'un noyau entre le mot ''gatta '' et ''cata'' est illustré par la Table 2.8.

Dans une perspective d'améliorer la complexité d'évaluation du noyau toutes sous-

TABLE 2.8 – Table de la programmation dynamique pour le calcul du noyau toutes sous-séquences entre le mot ''gatta '' et ''cata".

DP	ε	g	a	t	t	a
ε	1	1	1	1	1	1
c	1	1	1	1	1	1
a	1	1	2	2	2	3
t	1	1	2	4	6	7
a	1	1	3	5	7	14

séquences, nous pouvons observer que lorsque nous remplissons la ligne i, la somme

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

requise lors que nous atteignons la position j aurait peut-être pré-calculée dans un tableau P comme suit :

```
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
```

En utilisant le tableau P, nous pouvons maintenant calculer la ligne suivante avec une boucle simple :

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

Nous résumons maintenant l'Algorithme 1.

La boucle de 5 à 14 comporte deux boucles en séquentiel. Les deux boucles internes de 8 à 12 et de 13 à 14 s'exécutent dans un temps $O(|t|)$ d'où une complexité totale de l'algorithme en $O(|s||t|)$. Il est à noter que cette manière d'évaluation du noyau toutes sous-séquences permet de résoudre un problème plus général qui consiste

Algorithm 1: Évaluation du noyau toute sous-séquence.

Input: Les mots s et t.

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

```
1  n ← length(s)
2  m ← length(t)
3  for j = 1 : m do
4      DP[0, j] ← 1
   /* Traitement des lignes                                          */
5  for i = 1 : n do
       /* pré-calcul et remplissage du tableau P                     */
6      last ← 0
7      P [0] ← 0
8      for k = 1 : m do
9          P [k] ← P [last]
10         if t_k = s_i then
11             P [k] ← P [last] + DP[i − 1, k − 1]
12             last ← k
       /* renseignement de la ligne suivante                         */
13     for k = 1 : m do
14         DP[i, k] ← DP[i − 1, k] + P[k]
```

à calculer les noyaux $k(s(1:i); t(1:j))$ pour toute $1 \leq i \leq |s|$ et pour toute $1 \leq j \leq |t|$. Autrement dit, à la fin de l'évaluation du noyau toutes sous-séquences, on peut 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.

Par exemple la Table 2.9 illustre cette astuce d'amélioration pour le calcul du noyau toutes sous-séquences pour les deux mots s ="gatta" et t = "cata".

2.2.3 Noyau sous-séquences de longueur fixe

Le noyau sous-séquences de longueur fixe permet de limiter la recherche de sous-séquences à des sous séquences ayant une longueur fixe de taille p pour réduire la dimension de l'espace associé. En effet c'est un compromis entre le noyau p-spectre et le noyau toutes sous-séquences. L'espace associé au noyau sous-séquences de lon-

TABLE 2.9 – Table de la programmation dynamique avec un tableau P pour accélérer le calcul au niveau de chaque ligne.

DP	ε	g	a	t	t	a
ε	1	1	1	1	1	1
p	0	0	0	0	0	0
c	1	1	1	1	1	1
P	0	0	1	1	1	2
a	1	1	2	2	2	3
P	0	0	0	2	4	4
t	1	1	2	4	6	7
P	0	0	1	1	1	7
a	1	1	3	5	7	14

gueur fixe p est indexé par Σ^p [Shawe-Taylor and Cristianini, 2004], avec :

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

Le noyau associé est défini par :

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

Évaluation du noyau sous-séquences de longueur fixe

De même que précédemment, pour réaliser une évaluation efficace, il est nécessaire de faire la dérivation similaire à celle de noyau toutes sous-séquences en tenant compte de l'exigence de limitation qui implique l'utilisation des indices I^p de longueur p.

Le noyau associé peut être défini comme suit :

$$k(s,t) = \langle \Phi(s), \Phi(t) \rangle = \sum_{u \in \Sigma^p} \sum_{(I,J):u=s(I)=t(J)} 1 = \sum_{(I,J) \in I_p \times I_p : s(I)=t(J)} 1. \tag{2.5}$$

De la même manière récursive exprimée dans l'Équation (2.4), nous obtenons :

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

Cela entraîne la définition récursive du noyau sous-séquences de longueur fixe comme suit :

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

Le pseudo code de noyau sous-séquences de longueur fixe est donné par l'Algorithme 2 dont la complexité est en $O(p|s||t|)$.

Algorithm 2: 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 $k_p(s,t) = DP[n,m]$ avec n = |s| et m =|t|.

```
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]
```

2.2.4 Noyau sous séquences de mots (SSK)

Les noyaux étudiés dans les sections précédentes traitant les sous-séquences ne prennent pas en considération la distance séparant les éléments non contigües, par exemple : le mot "gon" se produit en tant que sous-séquence des mots "gone", "going" et "galion", mais nous considérons la première occurrence comme plus importante car elle est contigüe, tandis que la dernière occurrence est la plus faible des trois.

[Lodhi et al., 2002] développent un noyau sous-séquences de mots (String Subsequence Kernel en anglais, SSK) qui adopte une nouvelle méthode qui permet de comparer deux mots en fonction du nombre d'occurrences des sous-séquences communes qu'elles contiennent. Chaque sous-séquences est pondérée en fonction de la contiguïté de cette sous-séquences dans le mot.

Le noyau SSK est paramétré par deux valeurs λ et p, $\lambda \in]0, 1]$ est un paramètre de pénalisation d'écart mesurant la distance des éléments non contiguë de la sous séquence et p est la taille des sous-séquences.

La fonction de projection $\phi^p(s)$ pour une séquence s est défini pour tout $u \in \Sigma^p$ par :

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

Le noyau associé :

$$k_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi_u^p(s).\phi_u^p(t) = \sum_{u \in \Sigma^p} \sum_{I:u=s(I)} \sum_{J:u=t(J)} \lambda^{l(I)+l(J)}$$

À titre d'exemple, nous considérons les mots "cat", "car", "bat", et "bar" et $p = 2$ ainsi, la longueur des sous-séquences et un facteur de pénalisation λ. La Table 2.10 montre la projection de ces mots dans l'espace de redescription.

Donc le noyau non normalisé entre $k(cat, car) = \lambda^4$, alors que la version normalisée est obtenue comme suit : $k(car, car) = k(cat, cat) = 2\lambda^4 + \lambda^6$et donc $k(car, cat) = \lambda^4/(2\lambda^4 + \lambda^6) = 1/(2 + \lambda^2)$.

L'implémentation directe de ce noyau conduit à une complexité temporelle et spatiale $O(|\Sigma^p|)$. Pour un calcul efficace du noyau SSK, un noyau de suffixe est défini

TABLE 2.10 – Projection de "cat", "car", "bat", et "bar" dans une espace de redescription, pour des sous-séquences de longueur $p = 2$.

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

par la projection suivante :

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

avec I_p^k désignant l'ensemble des p-tuples des indices I avec $i_p = k$.

Le noyau associé est défini comme :

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

Le noyau SSK en termes de sa version de suffixe :

$$k_p(s,t) = \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} k_p^S(s(1:i), t(1:j)) \tag{2.6}$$

Nous pouvons donc évaluer la version du suffixe pour le cas où $p = 1$ comme suit :

$$k_1^S(s,t) = [s_{|s|} = t_{|t|}]\ \lambda^2.$$

Maintenant on peut introduire un noyau récursif pour la version suffixe du noyau en observant que, pour les mots s et t et les symboles a et b, le calcul de similarité entre sa et ta si $a = b$ est donné par :

$$k_p^S(sa, tb) = [a = b] \sum_{i=1}^{|s|} \sum_{j=1}^{|t|} \lambda^{2+|s|-i+|t|-j} k_{p-1}^S(s(1:i), t(1:j)). \tag{2.7}$$

Si nous devons implémenter une évaluation naïve du noyau SSK en utilisant la définition récursive de l'Équation 2.7 cela conduit à une complexité $O(p(|s|^2|t|^2))$. Donc,

il est nécessaire de considérer des implémentations efficaces. Il existe plusieurs technique pour améliorer la complexité de ce noyau : approche de la programmation dynamique, approche de la programmation éparse, approche de la programmation dynamique a base de trie [Shawe-Taylor and Cristianini, 2004], approches géométriques [Bellaouar et al., 2017b].

2.3 Noyau de séquence d'automate pondéré

Dans la littérature sur les noyaux, il existe une variété d'algorithmes qui traitent les séquences. A savoir le noyau p-spectre, le noyau toutes sous-séquences, le noyau sous-séquences de longueur fixe et le noyau SSK. Chacun des algorithmes est centré sur des problèmes individuels et pour chaque algorithme un effort important a été consacré à l'implémentation. Ceci était dit, la tentative d'unifier ces travaux semble un effort très intéressant. En effet, ceci s'inscrit dans le cadre de la contribution à une théorie d'unification. [Bellaouar et al., 2017a] proposent une plate-forme générale pour le calcul des noyaux de séquences en considèrent l'efficacité de calcul du noyau et la robustesse du modèle proposé.

2.3.1 Idée générale

[Bellaouar et al., 2017a] proposent une approche fondée sur la relation entre trois concepts : noyau, série formelle et automate pondéré. Les auteurs ont commencé par le noyau toutes sous-séquences avec pondération (Gappy All subsequence). L'espace de redescription de ce noyau est indexé par toutes les sous-séquences, en pondérant les séquences en fonction de leur taille comme pour la méthode du noyau SSK [Bellaouar et al., 2017a].
Par conséquent, le composant d'une sous-séquence w de s est représenté par :

$$\Phi_w(s) = \sum_{I:w=s(I)} \lambda^{l(I)},\ w \in \Sigma^* \tag{2.8}$$

Ce qui est perceptible, c'est que cette équation 2.8 qui exprime la projection des mots dans un espace de haute dimension du noyau toutes sous-séquences avec pondération est juste une définition d'une série formelle, où $(r, w) = \sum_{I:w=s(I)} \lambda^{l(I)}$.

En conséquence, l'équation $r = \sum_{w\in\Sigma^*} \sum_{I:w=s(I)} \lambda^{l(I)} w$ est considérée comme une nouvelle représentation de la séquence dans l'espace de redescription. De plus, cela peut être vu comme le comportement $\|A\|$ d'un automate pondéré (WA).

$$(\|A\|, w) = \sum_{\pi:l(\pi)=w} wt(\pi) = \sum_{I:w=s(I)} \lambda^{l(I)} = \Phi_w(s). \tag{2.9}$$

De ce qui précède, nous concluons qu'il est possible de représenter la projection de $r = \sum_{w\in\Sigma^*} \sum_{I:w=s(I)} \lambda^{l(I)} w$ par un automate pondéré. Pour des raisons d'efficacité, la construction de cet automate pondéré doit être compacte.

2.3.2 Construction d'un automate pondéré de toutes les sous-séquences

Dans cette section, nous discutons la définition d'un automate pondéré WA compact et non-déterministe associé à la projection de noyau toutes sous-séquences avec trous (Gappy All Subsequence Weighted Automaton, GASWA) d'un mot $s = s_1 s_2 \dots s_n$ sur Σ^* [BELLAOUAR, 2018]. L'automate GASWA est défini par le 7-tuplet $A = (\Sigma, Q_s, E_s, I_s, F_s, \lambda, \rho)$ tel que :

- $Q_s = \{q_0\} \cup \{q_i, q'_i, q''_i / 1 \leq i \leq n\}$
- $E_s = \{(q_0, \lambda, s_1, q_1), (q_0, \lambda, s_1, q'_1), (q_0, 1, \varepsilon, q''_1)\} \cup \{(q'_i, \lambda, s_{i+1}, q_{i+1}) / 1 \leq i \leq n-1\}$
  $\cup \{(q'_i, \lambda, s_{i+1}, q'_{i+1}) / 1 \leq i \leq n-1\} \cup \{(q'_i, \lambda, \varepsilon, q'_{i+1}) / 1 \leq i \leq n-1\} \cup$
  $\{(q''_i, \lambda, s_{i+1}, q_{i+1}) / 1 \leq i \leq n-1\}$
  $\cup \{(q''_i, 1, \varepsilon, q''_{i+1}) / 1 \leq i \leq n-1\}$
- $I_s = \{q_0\}$.
- $F_s = \{q_i / 0 \leq i \leq n\}$.
- Les fonctions de pondération initiale et final λ, ρ sont des scalaires égaux à 1.

Nous pouvons distinguer trois niveaux d'états $q_0, \dots, q_n$; $q'_1, \dots, q'_n$ et $q''_0, \dots, q''_{n-1}$. Le premier niveau représente le mot s et λ^1 est le poids de la transition entre 2 états correspondant à la distance entre 2 symboles consécutifs du mots. Pour permettre les trous, le premier niveau est étendu avec des ε-transitions $e_i = (q'_{i-1}, \varepsilon, \lambda, q'_{i+1})$, $i = 1....n-1$. Il est à noter que les ε-transitions entre $(q_0; q'_1)$ n'existe pas car l'effet de cette transition est identique à celui de commencer la sous-séquence à partir de l'état q''_1. Pour décider de mettre fin à une sous-séquence à un symbole donné il faut rajou-

ter des transitions du premier niveau à la seconde $e_i = (q_i, s_{i+1}, \lambda, q_{i+1}), i = 0....n-1$. Pour commencer avec n'importe quel symbole de s , il faut créer les transitions du troisième niveau : $e_i = (q''_i, \varepsilon, \lambda^0, q''_{i+1}),\ i = 0 \dots n-2$, avec $q''_0 = q_0$. Le poids $\lambda^0 = 1$ indique l'absence d'un trou ou d'une distance entre deux symboles.
De l'état $q''_i,\ i = 1 \dots n-1$, on peut soit décider de terminer une sous-séquence avec la transition $e_i = (q''_i, s_{i+1}, \lambda, q_{i+1})$ ou de continuer la sous-séquence à partir de l'état q'_{i+1} par la transition $e_i = (q''_i, s_{i+1}, \lambda, q'_{i+1})$. Ceci mène à la construction d'un automate compact (Figure 2.1) [Bellaouar et al., 2017a].

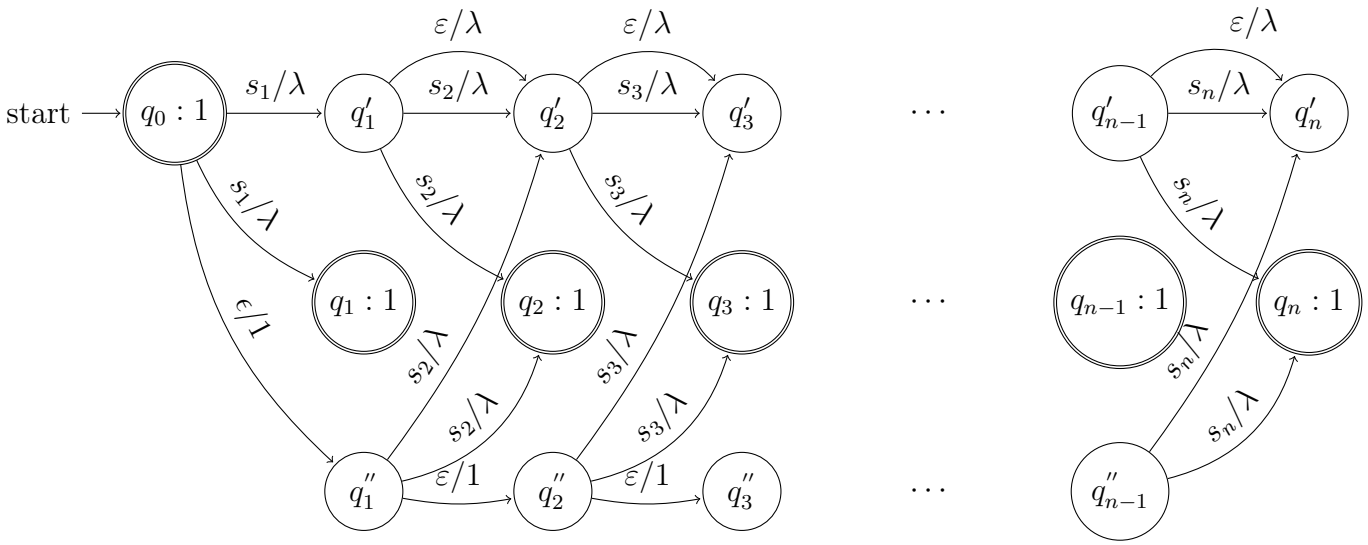


FIGURE 2.1 – Automate pondéré pour toutes les sous-séquences d'un mot $s = s_1 s_2 \dots s_n$. . Extrait de [Bellaouar et al., 2017a]

À titre d'illustration, nous construisons des GASWA pour les mots s= ''acb" et t="abc". L'espace de redescription associé à s et t peut être représenté par la Table 2.11.

TABLE 2.11 – L'espace de redescription des mots s = "acb" et t = "abc" associés au noyau toutes sous-séquences avec trous.

Sous-séquence	ε	a	c	b	ac	ab	cb	bc	acb	*abc*
Coefficient(acb)	1	λ	λ	λ	λ^2	λ^3	λ^2	0	λ^3	0
Coefficient(abc)	1	λ	λ	λ	λ^3	λ^2	0	λ^2	0	λ^3

Les projections des mot s = "acb" et t="abc" peuvent être représentées respective-

ment par les séries formelles des Équations 2.10 et 2.11.

$$r_s = 1 + a\lambda + c\lambda + b\lambda + ac\lambda^2 + ab\lambda^3 + cb\lambda^2 + acb\lambda^3. \tag{2.10}$$

$$r_t = 1 + a\lambda + c\lambda + b\lambda + ab\lambda^2 + ac\lambda^3 + bc\lambda^2 + abc\lambda^3. \tag{2.11}$$

La Figure 2.2 montre l'automate GASWA associé au mot s.

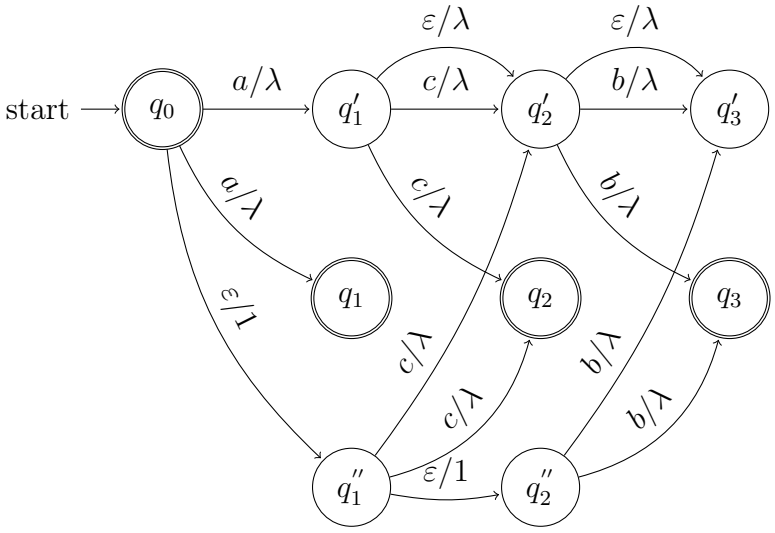


FIGURE 2.2 – Automate GASWA associé au mot s = "acb".

La Figure 2.3 montre l'automate GASWA associé au mot t.

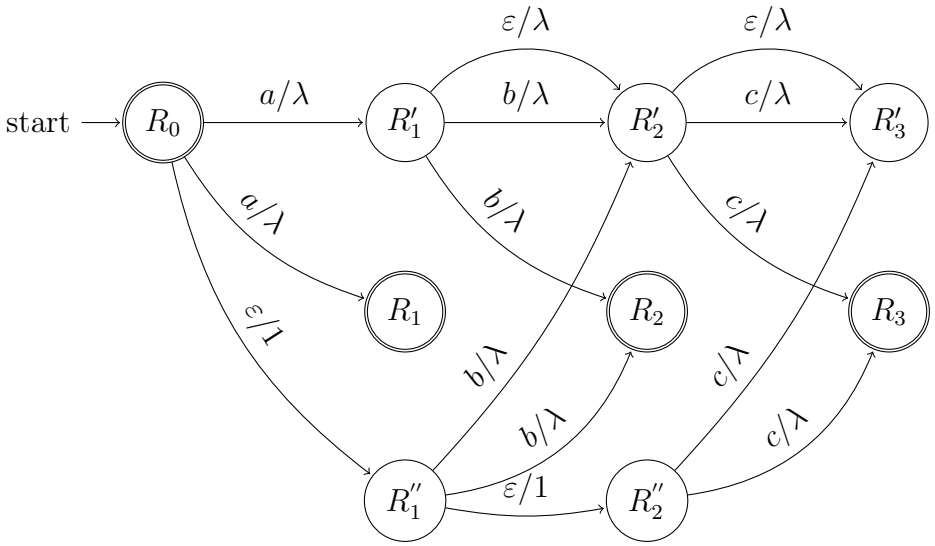


FIGURE 2.3 – Automate GASWA associé au mot t= "abc"

2.3.3 Calcul du noyau de séquence

Le calcul du noyau de séquences entre deux mots s et t à base d'automates pondérés s'effectue en deux étapes. La première est l'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 $A_t = (\Sigma, Q_t, E_t, I_t, F_t, \lambda, \rho)$ associés respectivement aux mots s et t. La deuxième étape est l'évaluation de tous les calculs de $A_{s,t}$ [Bellaouar et al., 2017a].

Intersection des automates pondérés

Nous pouvons considérer l'intersection de WA est une généralisation de l'intersection classique des automates finis [Hopcroft and Ullman, 1979]. Elle peut aussi être considérée comme un cas particulier de composition des transducteurs pondérés [Mohri et al., 1996].

Les états de $\mathrm{A}_{s,t}$ sont les paires des états à partir de A_s et les états à partir de A_t. Une transition $\mathrm{e}_{s,t} = ((q_s, q_t), l(e_{s,t}), wt(e_{s,t}), (q'_s, q'_t))$ de $\mathrm{A}_{s,t}$ est conçue a partir d'une transition $\mathrm{e}_s = (q_s, l(e_s), wt(e_s), q'_s)$ de A_s et une transition $\mathrm{e}_t = (q_t, l(e_t), wt(e_t), q'_t)$ 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 $\mathrm{A}_{s,t}$ sont respectivement les pairs des états initiaux (finaux) de A_s et A_t.

La complexité spatiale et temporelles de l'intersection est $O(p(|A_s||A_t|))$.

Malheureusement, l'application de cette intersection donnant des résultats incorrects dans le cas des ε-transitions peut générer des chemins redondants.

Pour faire face à ce problème, [Bellaouar et al., 2017a] ont étendu leur algorithme avec le mécanisme de ε-filtrage adopté de [Mohri et al., 1996].

Afin de comprendre le problème inhérent aux ε-transitions avec précision, nous discutons l'intersection de A_s et A_t associés respectivement aux mots $s =$ "acb" et $t =$ "abc". Pour illustrer davantage comment ces ε-transitions fonctionnent, nous augmentons A_s et A_t pour obtenir respectivement A'_s et A'_t comme suit : les ε-transitions dans $A'_s(A'_t)$ sont étiquetés $\varepsilon_1(\varepsilon_2)$, et les ε-transitions boucles correspondantes dans $A'_s(A'_t)$ sont marqués respectivement par des boucles étiquetées ε_2 (ε_1) (Figure 2.4 et Figure 2.5).

L'automate A_s après l'étape d'augmentation est illustré par la Figure 2.4.

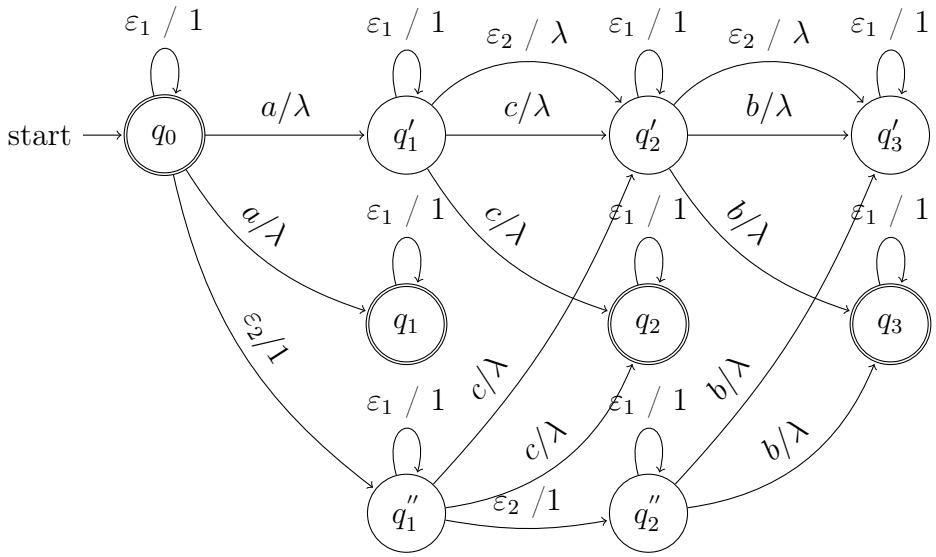


FIGURE 2.4 – L'automate A'_s résultat de l'augmentation de A_s.

L'automate A_t après l'étape d'augmentation devient comme suit (Figure 2.5).

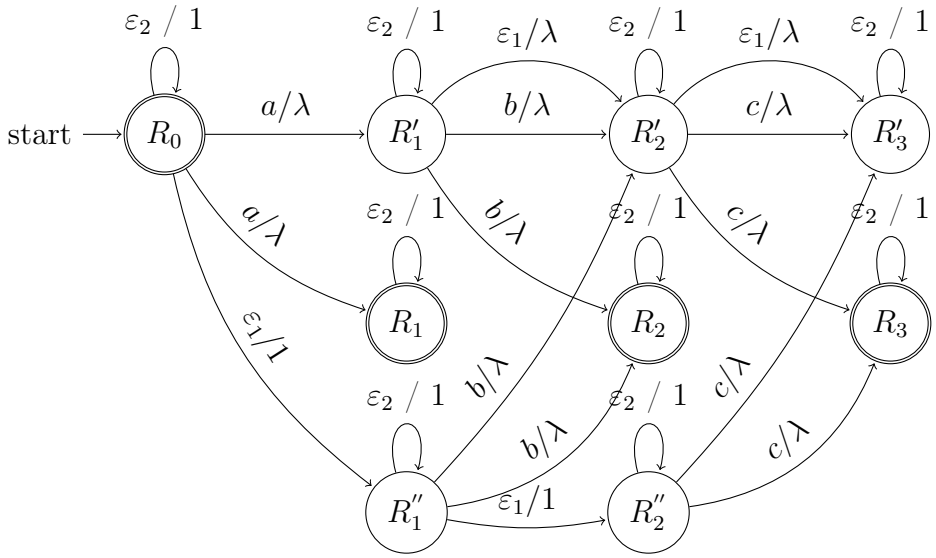


FIGURE 2.5 – L'automate A'_t résultat de l'augmentation de A_t.

Suite à cette augmentation, nous obtenons deux types de ε-mouvements dans les automates : (wait, move) et (move, wait).
La Figure 2.6 matérialise les 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, il doit garder un seul chemin. Donc la solution est d'utiliser le filtre qui peut être représenté par un automate pondéré (Figure 2.7) [Bellaouar et al., 2017a].

Évaluation du noyau

Principalement, pour évaluer le noyau toutes sous-séquences avec trous entre deux mots s et t, nous devons passer par la phase de construction des WA $A_{s,t} = A_s \cap A_t \cap A_f$, où A_s et A_t représentent respectivement l'automate du mot s et du mot t et A_f est l'automate filtre. Enfin, nous devons additionner les poids sur tous les calculs du WA $A_{s,t}$. Formellement, ceci se manifeste le calcul par de la somme $dist(q_0)$ (Équation 2.12) de l'ensemble des chemins $P(q_0, F)$ à partir de l'état initial jusqu'aux états finaux [Bellaouar et al., 2017a].

$$dist(q_0) = \sum_{\pi \in P(q_0,F)} \lambda(o(\pi)) \cdot wt(\pi) \cdot \rho(d(\pi)). \tag{2.12}$$

la Figure 2.8 montre l'automate intersection $A'_{s,t} = A'_s \cap A'_t \cap A_f$ dont le résultat de $k(s,t) = dist(q_0) = 6$ on prend la valeur $\lambda = 1$.
Il est à noter que dans une perceptive d'améliorer l'efficacité de calcul de noyaux de séquence, les auteurs de [Bellaouar et al., 2017a] ont proposé un nouveau algorithme d'intersection dit intersection par anticipation qui mis en œuvre une structure bitset.

2.3.4 Relations avec d'autres noyaux de séquences

Pour montré l'aspect unification de la plate-forme proposée on a essayé d'exprimer les différents noyaux de séquences étudiés dans les sections précédentes par le biais des noyaux à base d'automates pondérés [Bellaouar et al., 2017a].

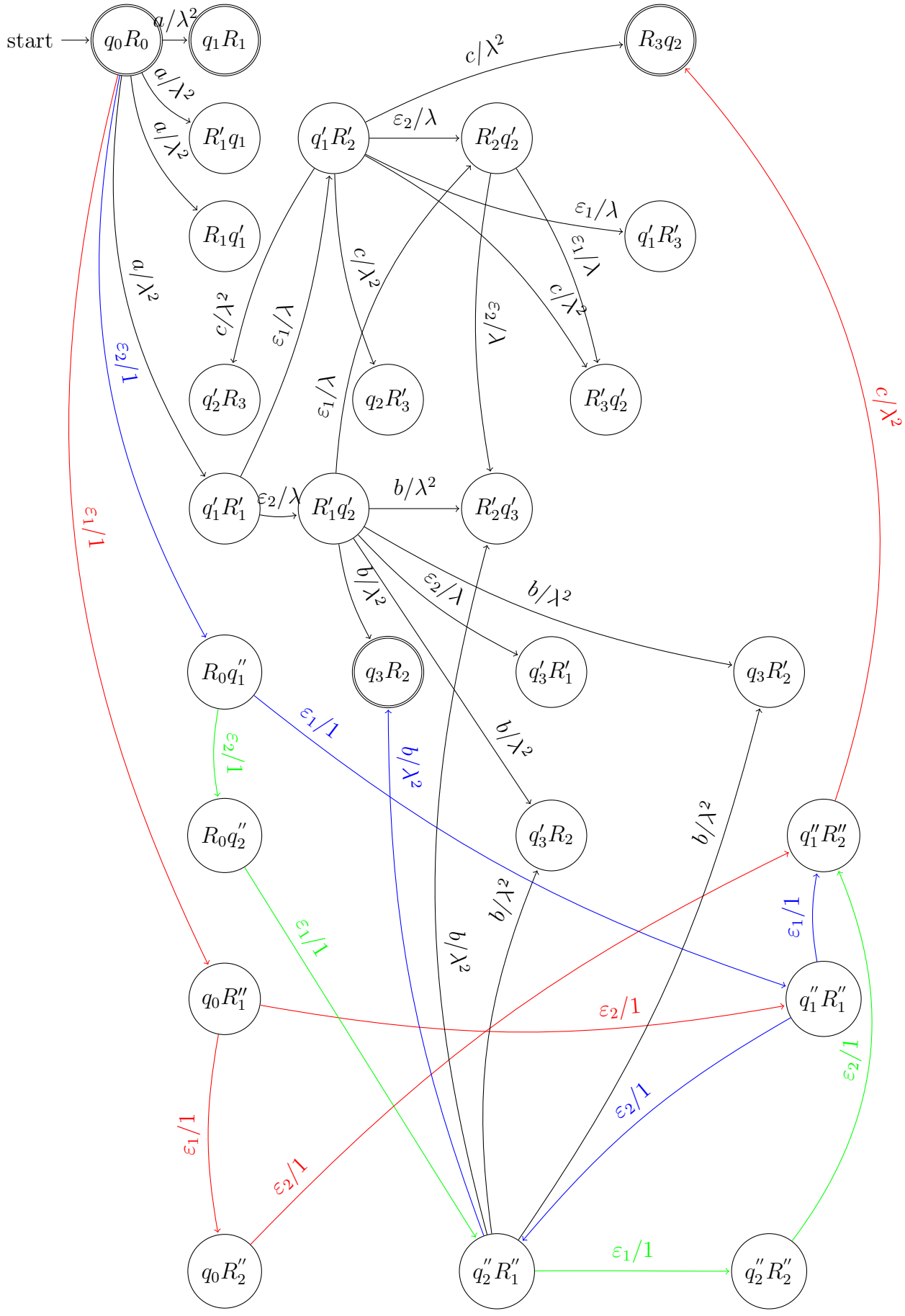


FIGURE 2.6 – Automate pondéré $A'_{s,t} = A'_s \cap A'_t$ montrant des chemins redondants.

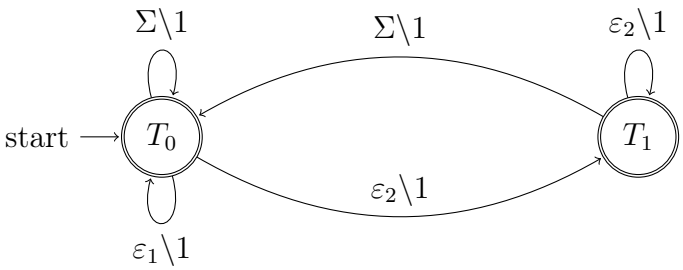


FIGURE 2.7 – Automate filtre pour éliminer les chemins redondants. Extrait de [Bellaouar et al., 2017a]

Noyau toutes sous-séquences

Le noyau toutes sous-séquences (section 2.2.2) entre deux mots s et t dans Σ^*, peut être défini comme un noyau de sous-séquence à base d'automates pondérés $dist(A_{s,t})$, où $A_{s,t} = A_s \cap A_t$, A_s et A_t sont des automates GASWA associés respectivement aux mots s et t avec la valeur de pondération $\lambda = 1$ [Bellaouar et al., 2017a].

Noyau sous séquences de mots (SSK)

Le noyaux SSK de longueur p (section 2.2.4) entre deux mots s et t dans Σ^* peut être exprimé comme un noyau de sous-séquence à base d'automates pondérés dist $dist(A_{s,t})$ où $A = A_s \cap A_t \cap A_{pgram}$, tel que A_s et A_t sont les GASWA représentant respectivement toutes les sous-séquences des mots s et t et le A_{pgram} est un WA associe à tous les p-grams sur Σ (Figure 2.9) [Bellaouar et al., 2017a].

Noyau sous-séquences de longueur fixe

Soient s et t deux mots dans Σ^* et p la longueur des sous-séquences. Nous pouvons définir le noyau sous-séquences de longueur fixe comme un noyau de sous-séquences à base d'automates pondérés $dist(A)$, où $A = A_s \cap A_t \cap A_{pgram}$, A_s et A_t sont respectivement les GASWA représentant toutes les sous-séquences des mots s et t et le A_{pgram} est un WA associe à tous les p-gram sur Σ [Bellaouar et al., 2017a].

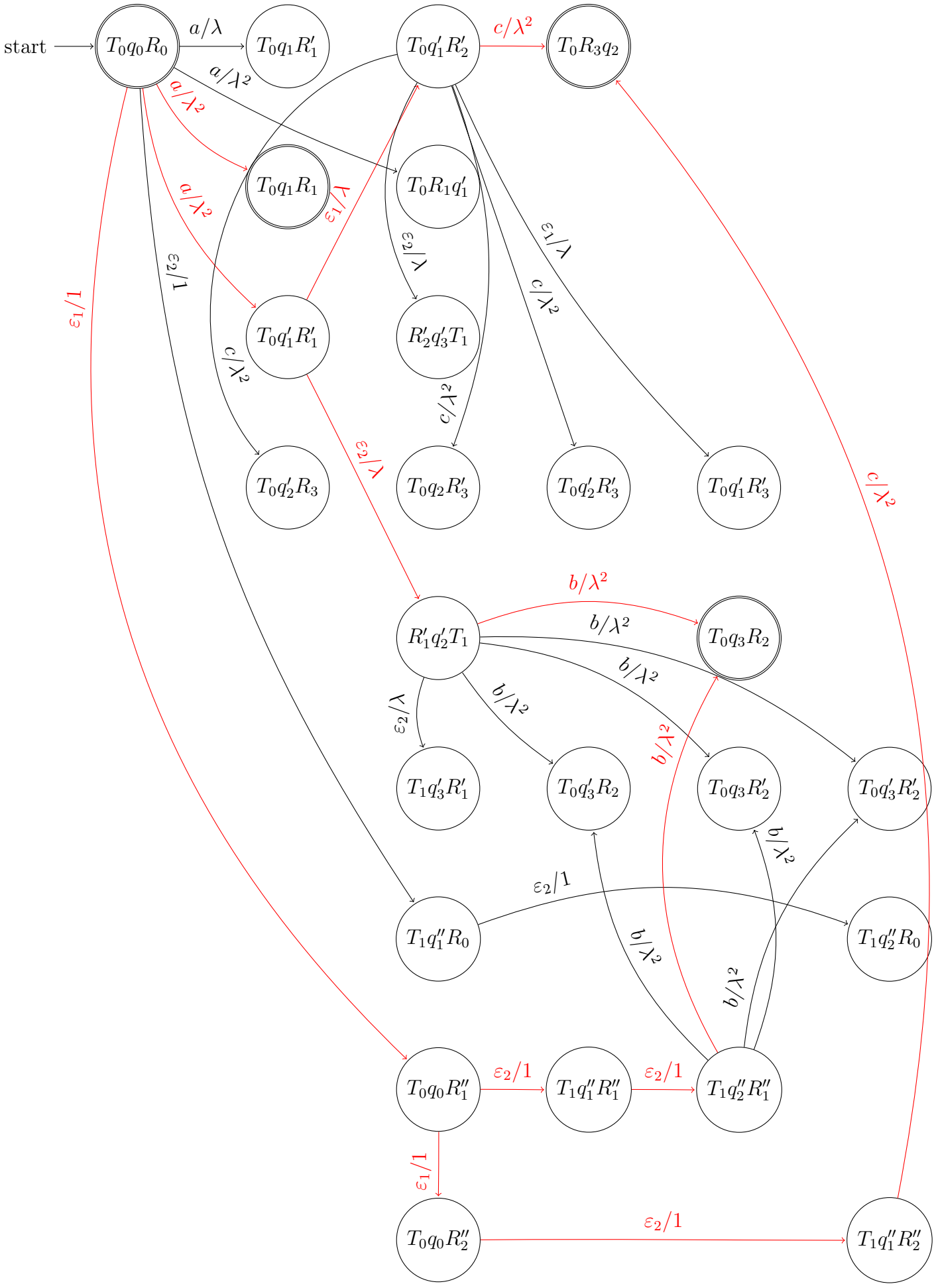


FIGURE 2.8 – Automate intersection $A'_{s,t} = A'_s \cap A'_t \cap A_f$.

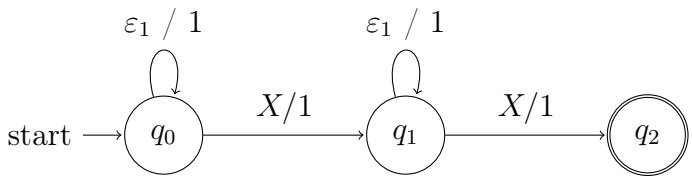


FIGURE 2.9 – Automate 2-gram, X représente un élément de Σ. Extrait de [BELLAOUAR, 2018]

Autres noyaux de sous-séquence

Il est facile de concevoir un autre type d'automates inhérent à un modèle particulier ($A_{pattern}$) qui peut être utile pour certaines applications. Pour calculer un nouveau type de noyau de séquence associé à un tel modèle, il suffit d'effectuer l'opération d'intersection($A_s \cap A_t \cap A_{pattern}$).

Pour les sous-mots, l'approche reste valide. Il suffit de construire un wa pour toutes les sous-mots et cela peut être réalisé en tant que construction spécifique de la construction GASWA en éliminent les ε-transitions du premier niveau. la Figure 2.10 montre une telle construction associée aux sous-mots [Bellaouar et al., 2017a].

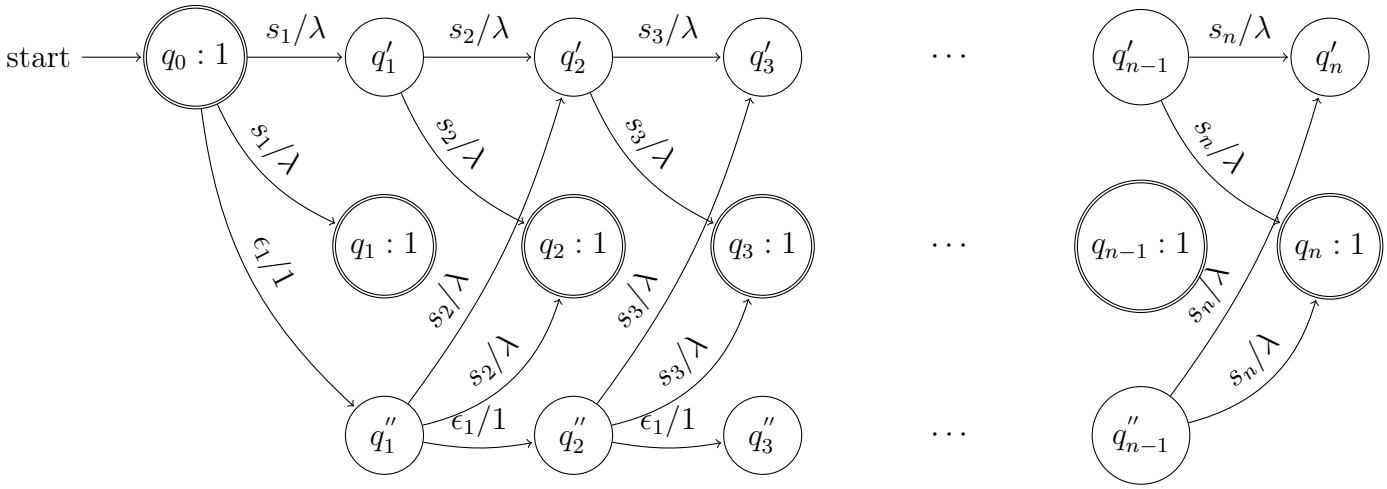


FIGURE 2.10 – Automate pondéré associé aux sous-mots. Extrait de [BELLAOUAR, 2018]

Chapitre 3

Implémentation de la plate-forme des noyaux de séquences

Ce chapitre se focalise sur l'implémentation de la plate-forme générale, évoquée dans le chapitre 2, pour calculer les noyaux de séquences.
Pour ce faire, nous avons opté pour la plate-forme analytique KNIME.
Cette implémentation est réalisée sur un PC Intel Core $i5$ avec un processeur de 2.7 GHz et 4 GB de mémoire centrale fonctionnant sous le système d'exploitation Windows 7.

3.1 Représentation des automates pondérés

Un automate fini est représenté par un graphe dont les sommets sont les états, et les arcs sont les transitions.
En informatique, il existe plusieurs implémentations possibles pour les graphes qui dépendent de la structure de donnée choisie. On peut implémenter les graphes à l'aide de la liste d'adjacence ou la matrice d'adjacence.
Une représentation par liste d'adjacence d'un graphe associe à chaque sommet du graphe la collection de ses voisins.
Dans notre implémentation nous avons choisi une représentation par liste d'adjacence. La représentation par liste d'adjacence de graphe consiste en un tableau de liste des états. Le tableau comporte autant de cases qu'il y a des états. Chaque case en magasine l'information que l'état soit finale ou non. Chacune des cases pointe vers une liste des états s'il existe une transition entre eux. Autrement dit, on ne

stocke que ceux dont on a besoin. Dans notre cas, chaque maillon de la liste contient le numéro de l'état, le symbole de l'alphabet et le paramètre λ et la distance entre deux symboles consécutifs. En effet, ces deux derniers champs sont utilisés comme pondération de la transition.
À titre d'exemple, la Figure 3.1 montre l'automate pondéré correspondant à la séquence 123, et la Figure 3.2 montre la structure de données liste d'adjacence associée.

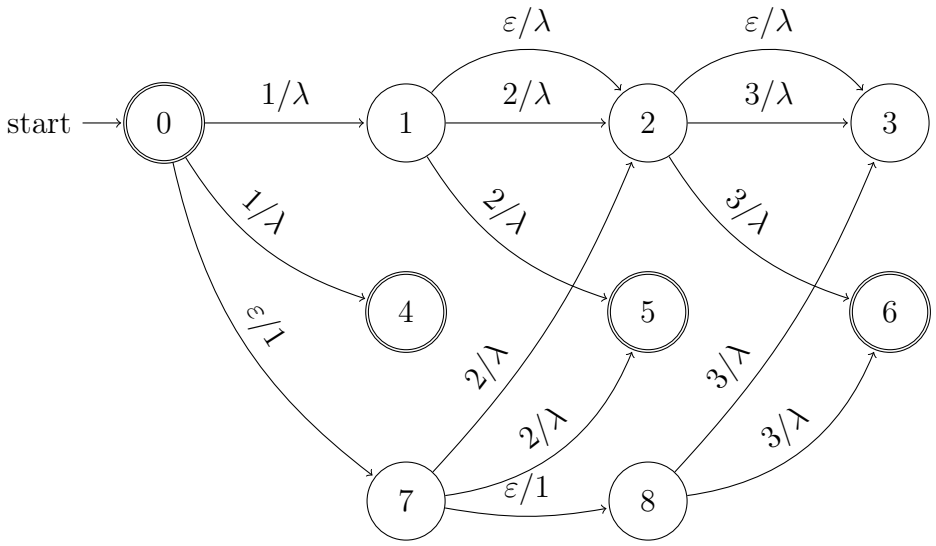


FIGURE 3.1 – Automate pondéré correspondant à la séquence 123..

De plus cette structure de données (liste d'adjacence) est utilisée pour représenter, au delà de l'automate des séquences, l'automate filtre, l'automate p-grams et l'automate intersection.
L'annexe A comprend les classes Java utilisées pour représenter les divers automates pondérés utilisés.

3.2 La plate-forme KNIME

KNIME, acronyme de Konstanz Information Miner est un logiciel gratuite open source développé par l'Université de Constance(Konstanz) en Allemagne. KNIME est écrit en Java et édité avec Eclipse.
KNIME est une plate-forme modulaire pour la conception et l'exécution de flux

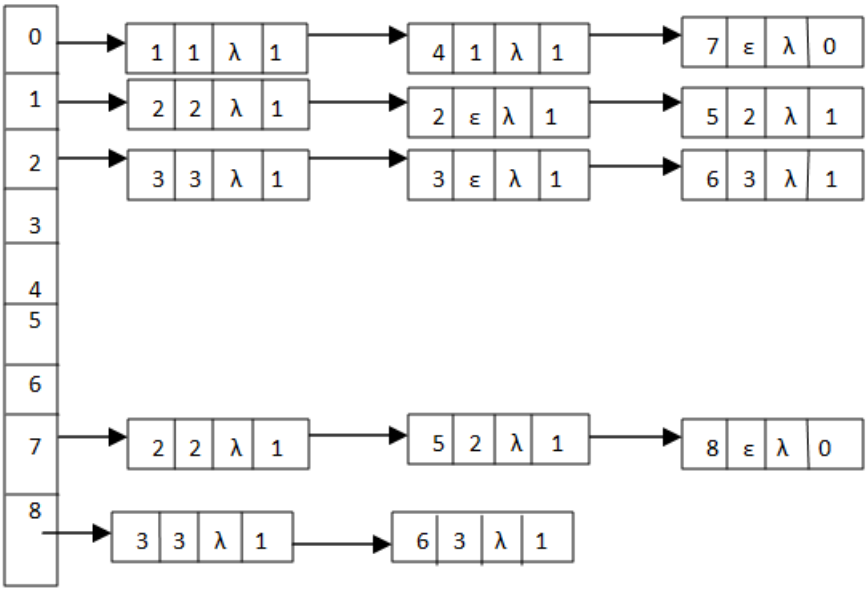


FIGURE 3.2 – Liste d'adjacence associée à l'automate pondéré correspondant à la séquence 123.

de travail(workflow) en utilisant des composants prédéfinis appelés nœuds KNIME. Il permet l'intégration de nombreuses méthodes d'apprentissage automatique et de data mining. Il est également efficace dans le prétraitement des données, c'est-à-dire l'extraction, la transformation et le chargement des données. Son pipeline modulaire en fait un outil d'exploration des données orientées flux de données. Il n'y a pas de restrictions par rapport au volume de données, ainsi les projets qui manipulent de grands volumes de données dépendent entièrement des ressources de l'utilisateur (taille disque, RAM, ...) et non pas de la plate-forme KNIME.

La plate-forme KNIME peut facilement interagir avec d'autres logiciels comme le Weka et son point fort est le fait qu'il nous offre la possibilité d'écrire notre propre code Java dans certains nœuds.

La Figure 3.3 montre le vue globale du plate-forme KNIME.

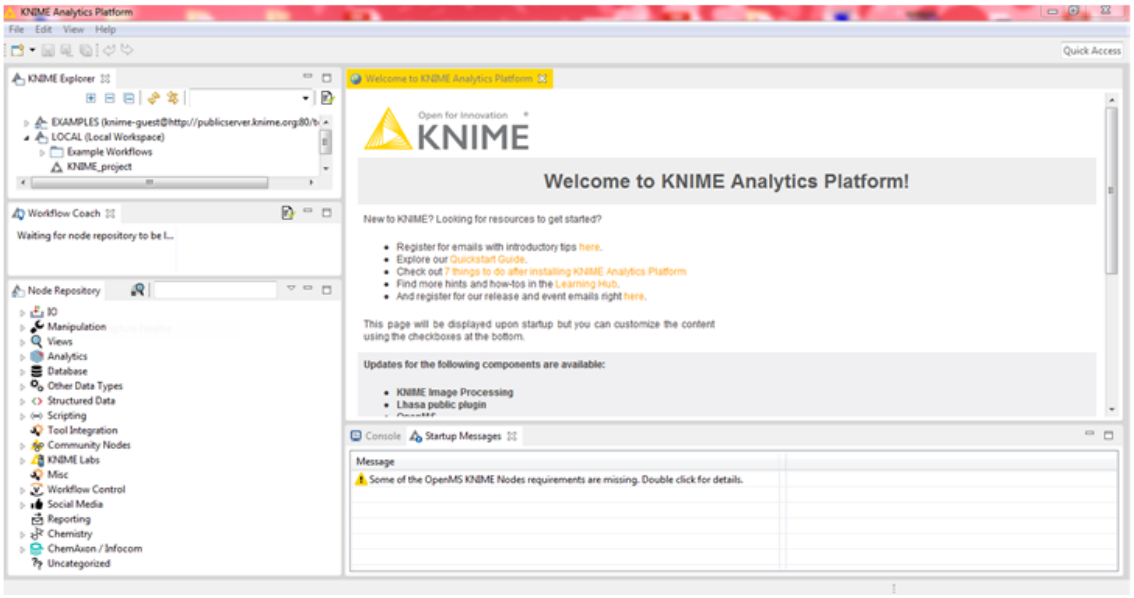


FIGURE 3.3 – Vue global du plate-forme KNIME.

Pour installer ce logiciel, il faut récupérer ce dernier sur la page de téléchargement de l'éditeur à l'adresse https ://www.knime.com/downloads.

3.2.1 Implémentation d'un nœud Sequence-Kernel

Dans cette section, nous nous intéressons à la mise en œuvre de la plate-forme générale étudiée dans la section 2.3. En effet, nous nous focalisons sur l'implémentation de l'automate pondéré qui est utilisé pour le calcul des noyaux de séquences d'une manière unifiée.
Par ailleurs, ceci peut être considéré comme une contribution au développement de la plate-forme KNIME. Ceci étant dit, nous créons un nouveau nœud, Sequence-Kernel, qui calcule les noyaux de séquences.
Pour programmer un nouveau nœud, nous devons d'abord télécharger et installer KNIME SDK via le lien https ://www.knime.com/download-knime-analytics-platform-sdk.
Après installation, nous ouvrons KNIME SDK et nous créons un nouveau projet en suivant les étapes ci-dessous :

1. Appuyer sur File → New → Other.
2. Choisir le wizard « Create a new KNIME Node-Extension» (Figure 3.4).

3. Entrer les informations du nœud (Nom du projet, Nom de l'auteur, . . .)(Figure 3.5).

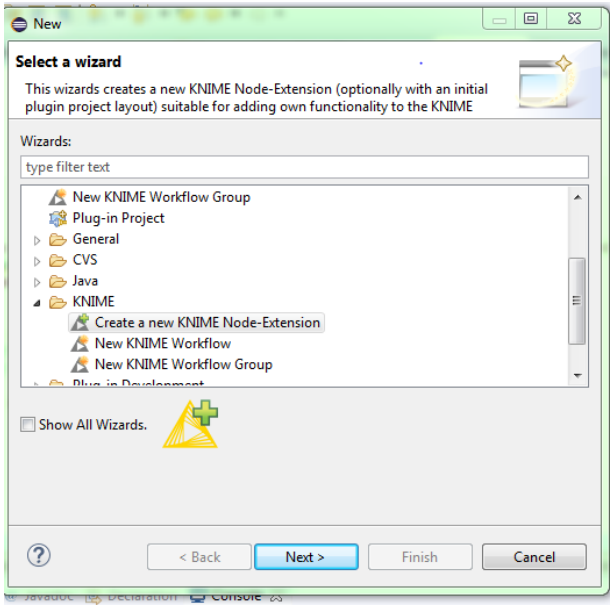


FIGURE 3.4 – Choix d'une option de création d'un nœud.

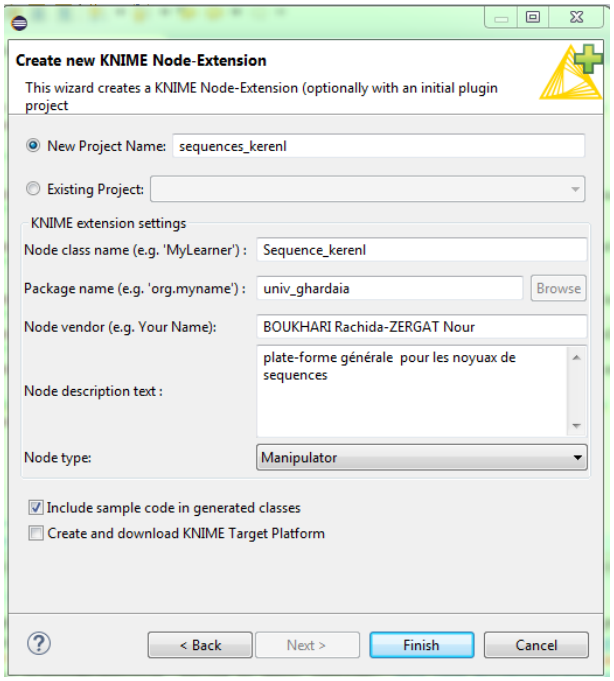


FIGURE 3.5 – Les informations à renseigner pour créer un nouveau nœud.

Le nouveau nœud, Sequence-Kernel, se manifeste sous forme d'un projet Java comportant quatre classes de base (Figure 3.6).

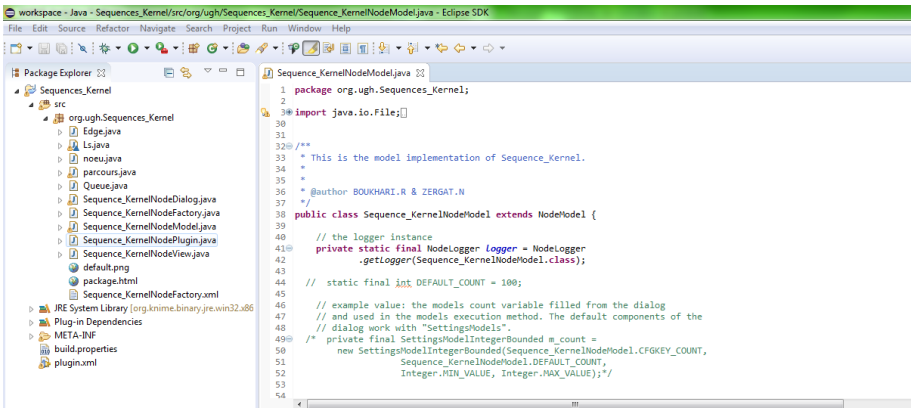


FIGURE 3.6 – Les classes Java de base associés au nœud Sequence-Kernel.

NodeDialog (optionnel)

KNIME fournit un moyen pratique pour aider l'utilisateur à spécifier ses paramètres de base à travers une boîte de dialogue. Le calcul des noyaux de séquences requiert des paramètres utilisateur spéciaux comme l'existence des trous, la taille des sous-séquences, le paramètre de pénalisation, ...

Nous utilisons deux composants de dialogue qui permettent à l'utilisateur d'entrer des valeurs :

✓ DialogComponentStringSelection pour paramétrer les trous, la taille des sous-séquences et la pondération.

✓ DialogComponentNumberEdit pour définir les valeurs de la taille des sous-séquences et la valeur de pondération.

Ces valeurs peuvent être facilement récupérées dans le NodeModel.

La Figure 3.7 illustre notre boite de dialogue.

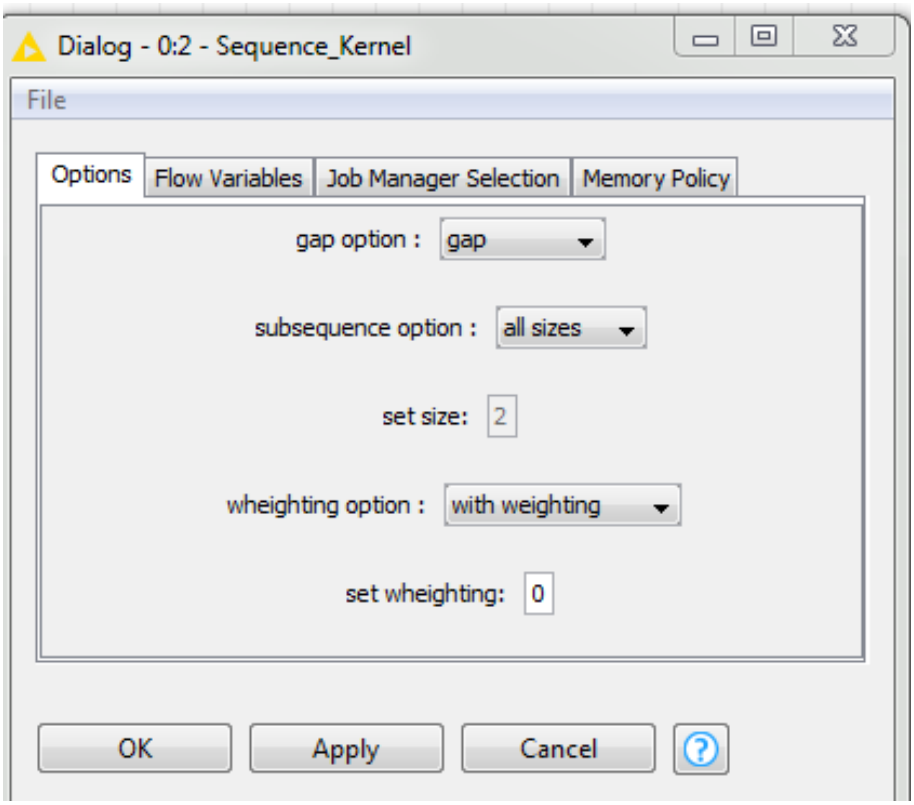


FIGURE 3.7 – Boite de dialogue pour les paramètres utilisateur du nœud Sequence-Kernel.

L'annexe B présente le source Java associé à la boite de dialogue Sequence-Kernel de la Figure 3.7.
Si l'utilisateur choisi l'option " all sizes" et "without weighting", nous désactivons DialogComponentNumberEdit associés aux set size et set weighting.

NodeModel

La classe NodeModel contient l'algorithme principal et gère le flux des données. Avant de s'occuper du détail du traitement et de la gestion de flux au sein du NodeModel, une phase d'interfaçage avec le DialogModel doit être réalisée pour assurer la prise en charge des paramètres utilisateurs fournit au niveau du DialogModel. Ceci est assuré par la définition de SettingModels.
À titre d'exemples pour les paramètres utilisateurs longueurs des sous-séquences nous avons défini, au niveau du NodeModel, les SettingModels et les configurations associées de l'annexe C.
Ces données d'entrées sont manipulées en utilisant l'argument exec de type ExecutionContext pour, enfin produire des donnés de sortie de type BufferedDataTable.

Les données d'entrées (inData) se présentent sous-forme d'un tableau mots. Le traitement consiste à collecter les paires de mots pour ensuite calculer le noyau de séquence correspondant selon les paramètres utilisateurs spécifiés.
Ceci étant dit, la classe NodeModel comprend les méthodes suivantes :
✓ Configure (DataTableSpec []).
✓ Execute (BufferedDataTable [], ExecutionContext.
✓ LoadInternals (File, ExecutionMonitor).
✓ LoadValidatedSettingsFrom (NodeSettingsRO).
✓ Reset().
✓ SaveInternals (File, ExecutionMonitor).
✓ SaveSettingsTo (NodeSettingsWO).
✓ ValidateSettings (NodeSettingsRO).
Dans notre contexte, nous nous intéressons uniquement aux méthodes impliquées par notre implémentation :

1) Méthode Execute

La méthode Execute (BufferedDataTable inData, ExecutionContext exec) est appelée après l'exécution réussie de tous les nœuds précédents, de sorte que toutes les données sont disponibles sur les ports d'entrées. Les données d'entrées sont disponible dans l'argument de tableau des données inData de type BufferedDataTable et doivent être valides. Autrement dit, inData ne doit pas être nulle ni contient d'éléments nuls.
La table de sortie contient deux colonnes : dans la première rangée de la colonne, nous trouvons des mots composées et la deuxième est la valeur du noyau entre eux. Elle joue le rôle de matrice noyau.

2) Méthode Configure

La méthode Configure, pour chaque nœud, est appelée dès que l'entrée a été connectée. Cette méthode à pour objectif de vérifier si le nœud est exécutable avec les paramètres actuels. Dans le cas où les paramètres requis sont manquants, la méthode Configure renvoie un message d'erreur. Il est à noter que nous avons utilisée cette méthode par défaut.

3) Méthode ValidateSettings

La méthode ValidateSettings permet de transformer les settings à partir du NodeModel vers le NodeDialog. Ceci est réalisé en implémentant les méthodes validateSettings, loadvalidatesettings et savesettings. Dans la méthode ValidateSettings, une vérification est réalisé pour assurer la validité et la présences des paramètres utilisateurs.

4) Méthode LoadvalidateSettings/ SaveSettings

La méthode Loadvalidatesettings permet de charger les valeurs des paramètres utilisateurs aux variables membres correspondantes du NodeModel. La méthode SaveSetting permet de gérer le cas des paramètres de l'utilisateur non encore définis, ou le cas des paramètres invalides.
La méthode ValidateSettings est toujours appelée avant les deux méthodes Loadvalidatesettings et Savesetting.

NodeView (facultatif)

Le NodeView affiche des informations sur le résultat de l'algorithme principal du NodeModel d'une manière spécifique.

NodeFactory

Le NodeFactory regroupe les classes NodeModel, NodeDialog, NodeView. Il contient trois méthodes :
✓ createNodeDialogPane () : créer ici la boîte de dialogue pour le nœud.
✓ CreateNodeModel () : créer et retourner une nouvelle copie du noeud NodeModel.
✓ CreateNodeView () : créer et renvoyer une nouvelle vue du nœud.

3.3 Test du nœud Sequence-Kernel

Afin de tester notre nœud Sequence-Kernel, nous devons démarrer un nouveau "runtime workbench" (qui est essentiellement une instance d'éclipse démarrée à partir de notre éclipse en cours d'exécution). Pour ce faire, les étapes suivantes sont entreprises :
-Dans le menu "Run", choisir "Run configurations".
-Dans la boîte de dialogue, sélectionner "Eclipse Application " sur la gauche et cliquer sur le bouton "New launch configuration" comme indiqué dans la Figure 3.8.

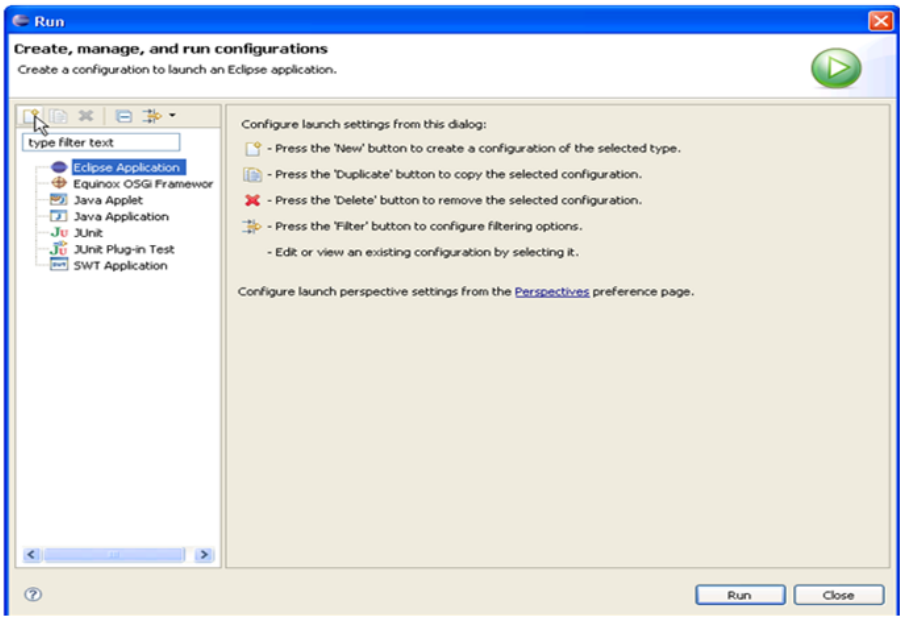


FIGURE 3.8 – Boîte de dialogue de création, de gestion et de configuration de l'exécution.

-Nous pouvons modifier les paramètres d'exécution de la configuration, comme activer les assertions ou donner plus de mémoire au processus Java nouvellement créé. À travers l'onglet arguments sous "VM arguments ", nous pouvons entrer "-ea -Xmx512M " afin d'utiliser des assertions (-ea) et augmenter l'espace mémoire disponible à 512 Mo.

- Lancer la configuration en utilisant "Run". Dans runtime workbench, nous ouvrons la perspective KNIME, ceci est fait par"Window" - "Open Perspective" - "Konstanz Information Miner". Nous remarquerons qu'il contient notre nouveau

nœud est défini dans le référentiel des nœuds.
Il est à signaler que les étapes précédentes sont effectuées une seule fois. Ultérieurement, pour de nouveaux test, il suffit de redémarrer runtime workbench en cliquant sur le bouton Run.

3.4 Exporter et déployer le Sequence-Kernel

Après le test de notre nœud Sequence-Kernel, il est temps de déployer le nœud en tant que plugin. Ceci est réalisé par les étapes suivantes :

1. Cliquer sur File et choisir Exporter.
2. Dans la fenêtre qui apparaît, sélectionne le dossier "Plug-in Development" et choisir l'option "Plug-ins et fragments déployables".
3. Après avoir appuyé sur le bouton "Next", l'assistant affiche une liste avec tous les plugins disponibles dans notre espace de travail et celui que vous avez cliqué avec le bouton droit est sélectionné. Ainsi, il nous suffit de sélectionner un répertoire déploient de notre plugin.
4. Si vous sélectionnez un répertoire, le plugin est exporté vers cette destination dans un répertoire appelé "plugin". Ce répertoire contient ensuite le plugin en tant que fichier.jar.
5. Une fois le plugin exporté, il peut être installé dans n'importe quelle installation KNIME en copiant notre "plugin" dans le dossier " plugins " de l'installation de KNIME.

3.5 Utilisation du nœud Sequence-Kernel dans KNIME

Après l'exportation de plugin, nous ouvrons la plate-forme KNIME et nous créons un nouveau workflow (menu FILE /NEW). Nous remarquerons qu'il contient également notre propre nœud (Sequence-Kernel) dans le référentiel de nœuds (Figure 3.9).

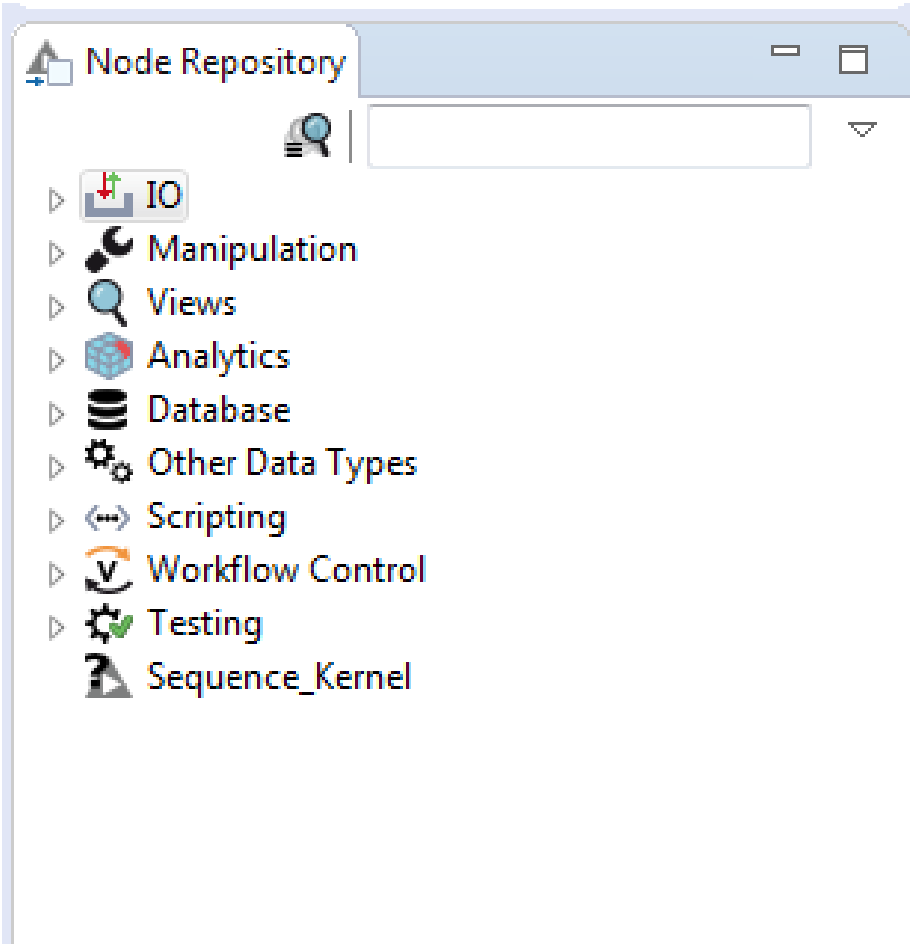


FIGURE 3.9 – Référentielle de nœuds comprenant le nœud Sequence-Kernel.

Dans ce qui suit, nous présentons un jeu de test d'utilisation de notre nœud pour calculer une variété de noyaux de séquences.
Les données en entrées (ensemble de mots) se présentent sous forme d'un fichier texte, où chaque ligne représente un mot.
Nous utilisons le nœud CSVReader pour charger le ficher de test. La Figure 3.10 illustre le paramétrage de nœud CSVReader et la Figure 3.11 illustre la table résultat de ce nœud appliqué sur le fichier notre cor.txt, contient 6 mots crées aléatoirement.

Avant de passer au calcul proprement dit du noyau de séquences nous considérons deux cas.
Dans le cas où les mots sont de même tailles, nous appelons directement notre nœud qui calcule le noyau de séquences (Figure 3.12).

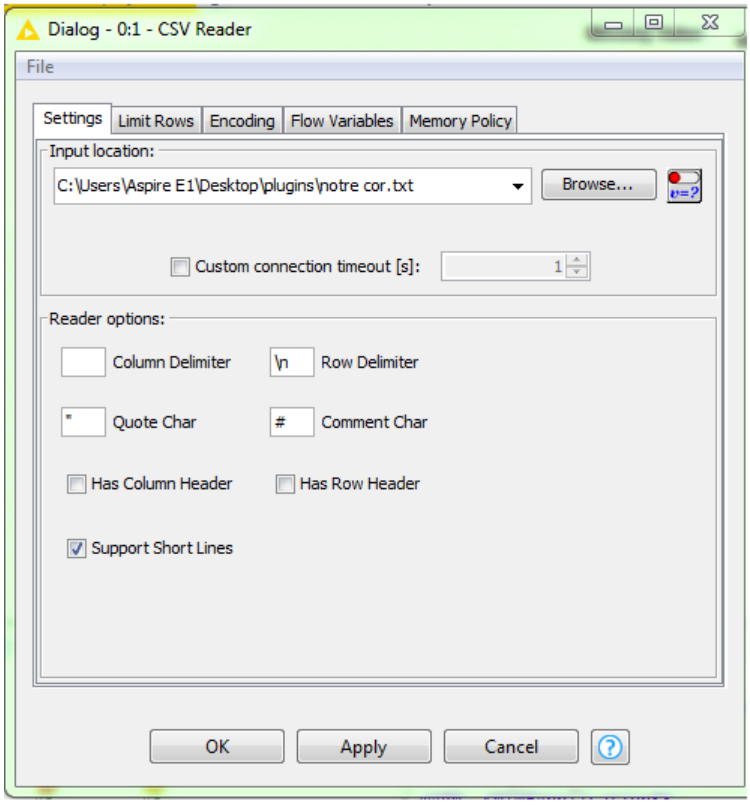


FIGURE 3.10 – Paramétrage du nœud CSVReader.

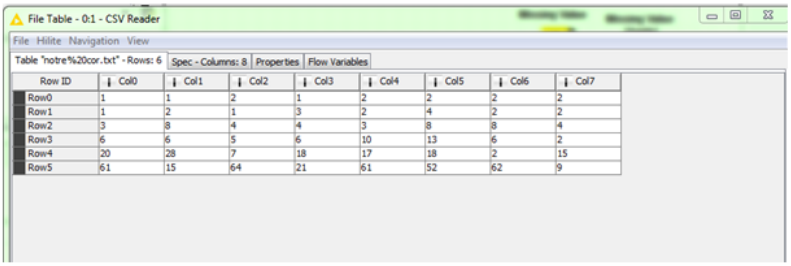


FIGURE 3.11 – Table de sortie du nœud CSVReader.

Si nous voulons calculer le noyau toutes sous-séquences entre les mots de ficher, dans la boîte de dialogue, nous sélectionnons les propriétés affichées dans la Figure 3.13.

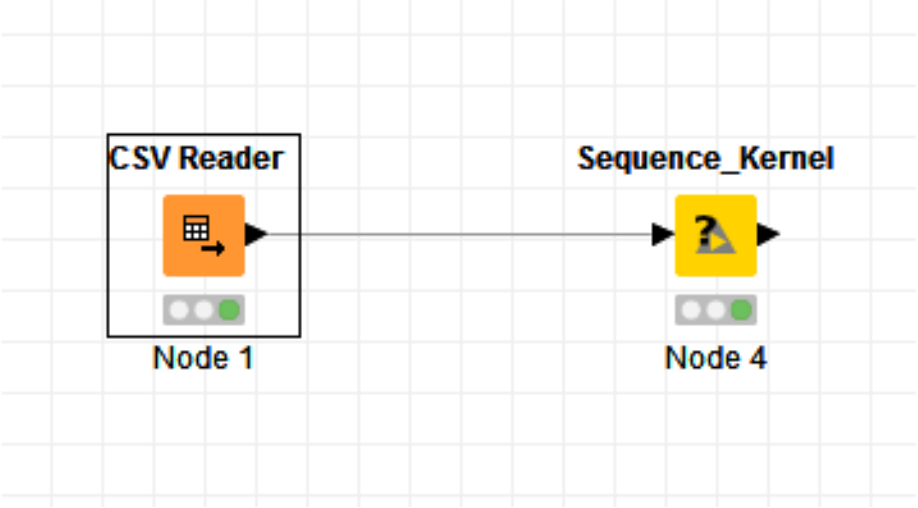


FIGURE 3.12 – Nœud Sequence-Kernel pour calculer les noyaux de séquences.

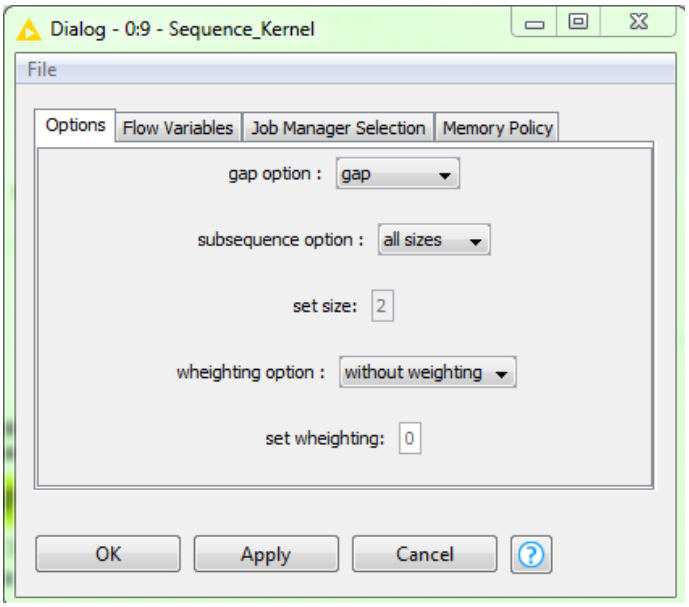
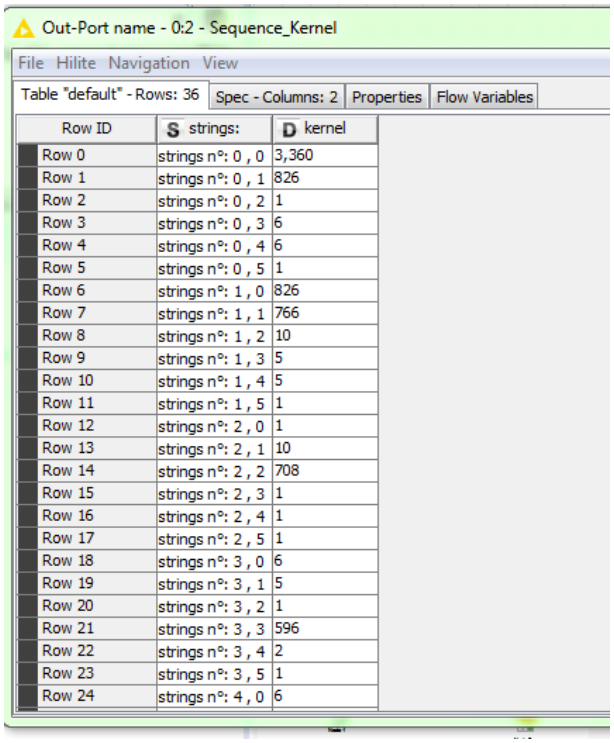


FIGURE 3.13 – Paramétrage du noyau toutes sous-séquences.

Pour voir le résultat du calcul du noyau, faites un clic droit et sélectionnez le bouton (out port name). La matrice noyau inhérent au fichier de données est affichées (Figure 3.14).



Row ID	strings:	kernel
Row 0	strings n°: 0 , 0	3,360
Row 1	strings n°: 0 , 1	826
Row 2	strings n°: 0 , 2	1
Row 3	strings n°: 0 , 3	6
Row 4	strings n°: 0 , 4	6
Row 5	strings n°: 0 , 5	1
Row 6	strings n°: 1 , 0	826
Row 7	strings n°: 1 , 1	766
Row 8	strings n°: 1 , 2	10
Row 9	strings n°: 1 , 3	5
Row 10	strings n°: 1 , 4	5
Row 11	strings n°: 1 , 5	1
Row 12	strings n°: 2 , 0	1
Row 13	strings n°: 2 , 1	10
Row 14	strings n°: 2 , 2	708
Row 15	strings n°: 2 , 3	1
Row 16	strings n°: 2 , 4	1
Row 17	strings n°: 2 , 5	1
Row 18	strings n°: 3 , 0	6
Row 19	strings n°: 3 , 1	5
Row 20	strings n°: 3 , 2	1
Row 21	strings n°: 3 , 3	596
Row 22	strings n°: 3 , 4	2
Row 23	strings n°: 3 , 5	1
Row 24	strings n°: 4 , 0	6

FIGURE 3.14 – Table de sortie (Matrice noyau) toutes sous-séquences.

Par contre, si nous voulons calculer le noyau p-spectre de taille 2 par exemple, nous spécifions les propriétés affichées dans la boîte de dialogue Figure 3.15.

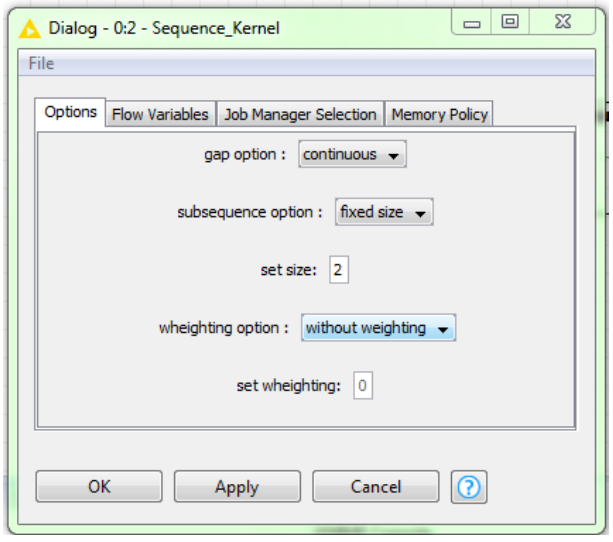
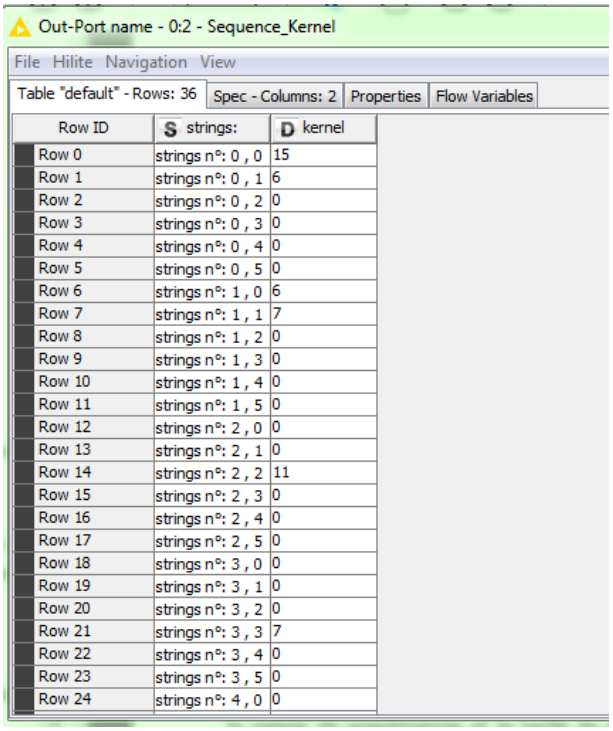


FIGURE 3.15 – Paramétrage de noyau du p-spectre de taille 2.

La Figure 3.16 montre les résultats de calcul du noyau de p-spectre du taille 2.



Row ID	strings:	kernel
Row 0	strings n°: 0 , 0	15
Row 1	strings n°: 0 , 1	6
Row 2	strings n°: 0 , 2	0
Row 3	strings n°: 0 , 3	0
Row 4	strings n°: 0 , 4	0
Row 5	strings n°: 0 , 5	0
Row 6	strings n°: 1 , 0	6
Row 7	strings n°: 1 , 1	7
Row 8	strings n°: 1 , 2	0
Row 9	strings n°: 1 , 3	0
Row 10	strings n°: 1 , 4	0
Row 11	strings n°: 1 , 5	0
Row 12	strings n°: 2 , 0	0
Row 13	strings n°: 2 , 1	0
Row 14	strings n°: 2 , 2	11
Row 15	strings n°: 2 , 3	0
Row 16	strings n°: 2 , 4	0
Row 17	strings n°: 2 , 5	0
Row 18	strings n°: 3 , 0	0
Row 19	strings n°: 3 , 1	0
Row 20	strings n°: 3 , 2	0
Row 21	strings n°: 3 , 3	7
Row 22	strings n°: 3 , 4	0
Row 23	strings n°: 3 , 5	0
Row 24	strings n°: 4 , 0	0

FIGURE 3.16 – Table de sortie (Matrice noyau) de p-spectre de taille 2.

Nous pouvons également calculer le noyau sous-séquences de mots (SSK). Ceci est fait en sélectionnant dans la boîte de dialogue les propriétés du noyau comme la valeur de pondération et la taille des sous-séquences (Figure 3.17).

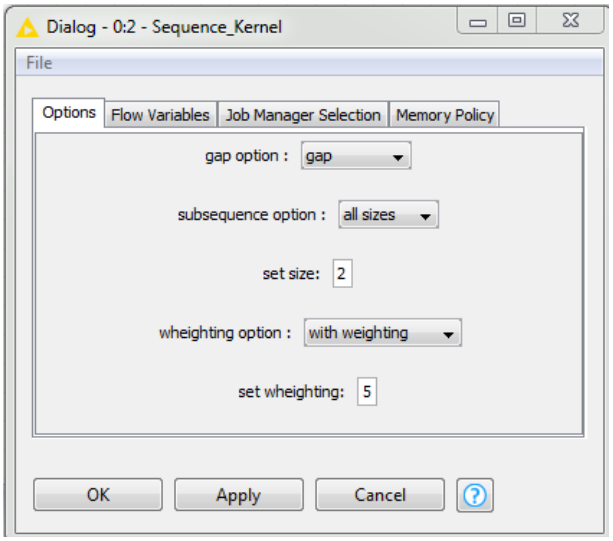
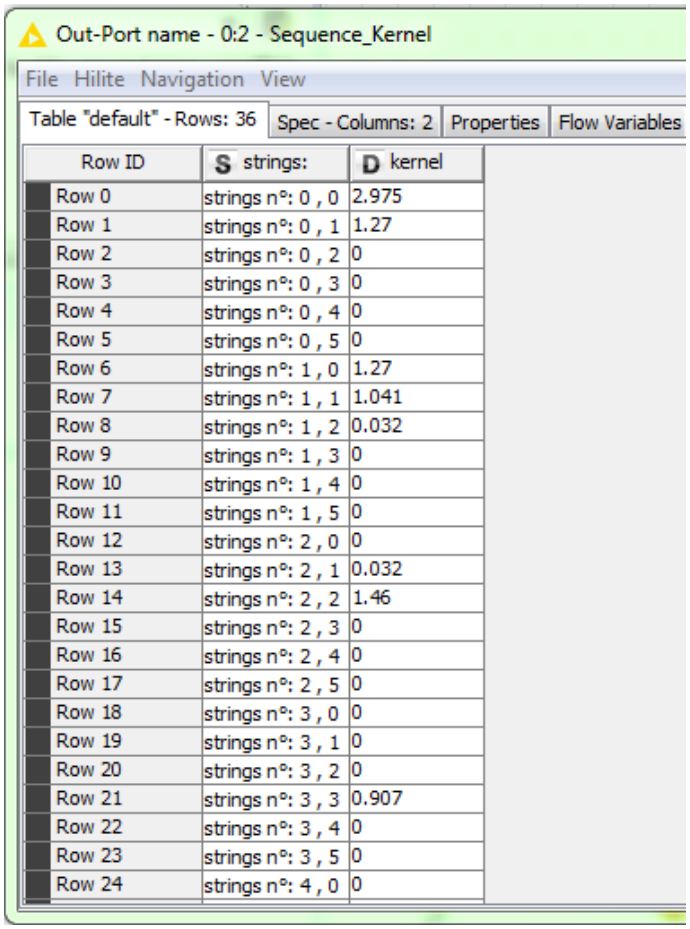


FIGURE 3.17 – Paramétrage de noyau sous-séquences de mots.

La Figure 3.18 montre la matrice noyau des sous-séquences de mots.



Out-Port name - 0:2 - Sequence_Kernel

File Hilite Navigation View

Table "default" - Rows: 36 | Spec - Columns: 2 | Properties | Flow Variables

Row ID	strings:	kernel
Row 0	strings n°: 0 , 0	2.975
Row 1	strings n°: 0 , 1	1.27
Row 2	strings n°: 0 , 2	0
Row 3	strings n°: 0 , 3	0
Row 4	strings n°: 0 , 4	0
Row 5	strings n°: 0 , 5	0
Row 6	strings n°: 1 , 0	1.27
Row 7	strings n°: 1 , 1	1.041
Row 8	strings n°: 1 , 2	0.032
Row 9	strings n°: 1 , 3	0
Row 10	strings n°: 1 , 4	0
Row 11	strings n°: 1 , 5	0
Row 12	strings n°: 2 , 0	0
Row 13	strings n°: 2 , 1	0.032
Row 14	strings n°: 2 , 2	1.46
Row 15	strings n°: 2 , 3	0
Row 16	strings n°: 2 , 4	0
Row 17	strings n°: 2 , 5	0
Row 18	strings n°: 3 , 0	0
Row 19	strings n°: 3 , 1	0
Row 20	strings n°: 3 , 2	0
Row 21	strings n°: 3 , 3	0.907
Row 22	strings n°: 3 , 4	0
Row 23	strings n°: 3 , 5	0
Row 24	strings n°: 4 , 0	0

FIGURE 3.18 – Table de sortie (Matrice noyau) sous-séquence de mots.

Dans le cas où nous disposons les mots de tailles différentes, la table inData émet une erreurs car elle n'accepte pas les valeurs nulles dues à la différence des tailles de mots.

Pour faire face à cette situation nous avons combiné l'utilisation des nœuds Missing Value et Missing Value (Apply) qui servent à remplacer les valeurs nulles par une valeur spécifique.

La figure 3.19 illustre le choix de la valeur manquante pour le nœud Missing Value. Par la suite, le nœud Sequence-Kernel ignore toutes les cellules contenant la valeur manquante spécifiée.

Le workflow (Figure 3.20) sert à calculer le noyaux de séquences lorsque les mots sont de tailles différentes.

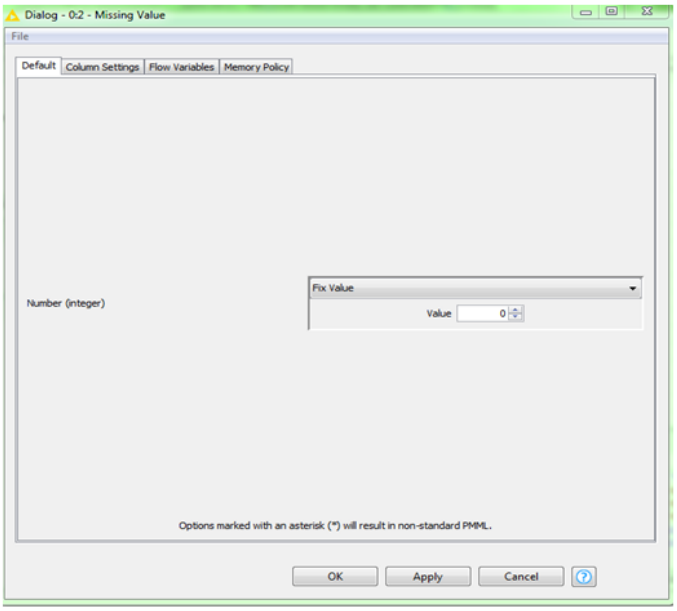


FIGURE 3.19 – Boite de dialogue missing value.

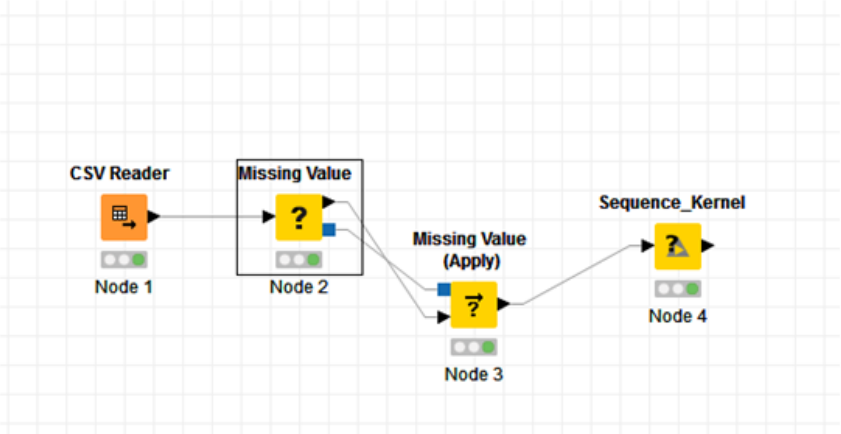


FIGURE 3.20 – Workflow pour le calcul des noyaux de séquences.

De ce qui précède, nous pouvons concluons que notre nœud (Sequence-Kernel) est capable de calculer les noyaux de séquences rencontrés dans la littérature. De plus, il est clair, qu'il est possible de considérer plusieurs autre noyaux de séquences en combinant les différents paramètres.

Par ailleurs, vue l'aspect modulaire des noyaux de séquences, nous pouvons étendre les workflows des noyaux des séquences pour réaliser d'autre tâches d'apprentissage automatique.

Conclusion

Dans la littérature, il existe une variété de noyaux de séquences chacun est centré sur un problème spécifique. Il est judicieux donc de penser à un outil d'unification de ces noyaux de séquences.
Dans cette perspective, nous avons investigué une famille de noyaux de séquences, à savoir : le noyau p-spectre, le noyau de toutes sous- séquences, le noyau sous-séquences de longueur fixe et le noyau sous-séquences de mots (SSK).
Par la suite, nous avons présenté et implémenté une plate-forme générale à base d'automate pondéré pour calculer les noyaux de séquences.
Notre implémentation se manifestée comme un nouveau nœud au sein de la plate-forme KNIME. Par conséquente, il devient facile d'utiliser les noyaux de séquences dans des tâches divers d'apprentissage automatique.
Au terme de notre travail, nous envisageons intégrer officiellement notre nœud dans KNIME. Par ailleurs, il semble intéressant d'explorer toutes les possibilités de notre nœud (nouveaux noyaux de séquences) dans des scénarios pratiques.

Annexe

Annex A : Représentation des graphes par liste d'adjacence

class Graph{ // structure de graphe par liste d'adjacence.
ArrayList<Edge>[] G ; // les adjacences d'un état.
boolean[] finales ;//tableau booléen pour stocker l'information si que l'état soit finale ou non.
//int V ; nombre des états de Graphe.
public int[] sourcea ;//Variable globale pour stocker les états sources de Graphe A.
public int[] sourceb ;// Variable globale pour stocker les états sources de Graphe B.
public Graph(int n)//constructeur surchargé.
void addEdge(int u,int v,int w,double lambda,int puissance) //pour rajouter un nouvel transition.
public boolean estFinale(Integer e)// pour reconnaître si l'état e est finale ou non.
public Graph initialiser2(Graph g,int[] mot,double lm,int epsilon1,int epsilon2 ,boolean trous).// l'augmentation de graphe et le boolean pour avoir si 'il existe des trous.
public Graph initialiserFiltre(Graph g,int sigma,double lm) // l'automate filtre.
public Graph initialiserPgram(Graph g,int p,int sigma,double lm) //l'automate p-gram.
public Graph intrsection(Graph A, Graph B,double lm, String s) //
l'intersection entre les graphes A et B, et le String s est le chemin de fichier pour stocker la structure de graphe d'intersection.}
public class Edge{
int v ;// l'état de destination.
int w ; //l'alphabet.
double lambda ; // la pondération lambda.

int puissance ; // la puissance de pondération.
public Edge(int v,int w,double lambda,int puissance)//constructeur de la class Edge surchargé.}

Annexe B : Classe NodeDialog

final SettingsModelString selectionTaille = Sequence-KernelNodeModel.createSettingsModelSelectionTaille() ;// crée un instance de setting models pour paramètrer la taille de sous-séquences.
SettingsModelInteger taille = new SettingsModelInteger(Sequence-KernelNodeModel.TAILLE, 2) ; () ;// crée un instance de setting models pour définir la valeur de la taille de sous-séquences.
//Nous utilisons ces settings models avec le composant StringSelection et le composant NumberEdit.
addDialogComponent(new DialogComponentStringSelection(selectionTaille,"subsequence option : ",Sequence-KernelNodeModel.TAILLE-BIN)) ;//le composant de dialogue par défaut DialogComponentStringSelection pour paramètrer la taille de sous-séquences.
addDialogComponent(new DialogComponentNumberEdit(taille, "set size : ", 1)) ;// le composant de dialogue par défaut DialogComponentNumberEdit pour définir les valeurs de taille de sous-séquences.
//Si l'utilisateur choisi l'option " all sizes", nous désactivons DialogComponentNumberEdit associés au set size.
selectionTaille.addChangeListener(new ChangeListener() { @Override public void stateChanged(ChangeEvent arg0) { taille.setEnabled(false) ; if(selectionTaille.getStringValu sizes")){ taille.setEnabled(true) ; } } }) ;

Annexe C : Classe NodeModel

public static final String TAILLE = "taille" // la clé de configuration de la valeur de taille : utilisée par NodeDialog et NodeModel comme un clé pour stocker les paramètres.
public static final String[] TAILLE-BIN = new String[] { "all sizes","fixed size"} ;
// la clé de configuration de l'ensemble des valeurs possibles pour selectioner la taille de sous-séquences : utilisée par NodeDialog et NodeModel comme un clé pour

stocker les paramètres.
//Parce que vous avez besoin d'instancier des SettingsModels identiques à deux endroits (dans le constructeur NodeModel et le constructeur NodeDialog), nous implémentons des méthodes statiques qui créant ces instances et l'appelées ces méthodes aux deux endroits.
static SettingsModelString createSettingsModelSelectionTaille() {return new SettingsModel
BIN", "") ;//
//Définition des SettingsModels
private final SettingsModelString m-taille-sel =
createSettingsModelSelectionTaille() ;
private SettingsModelInteger m-taille = new
SettingsModelInteger(Sequence-KernelNodeModel.TAILLE, 2) ;
//Méthode d'exécution de Sequence-Kernel
protected BufferedDataTable[] execute(final BufferedDataTable[] inData,
final ExecutionContext exec).
//Les méthodes saveSettings, validateSettings et loadValidatedSettings pour vérifier la validation et le chargement de SettingsModels.
protected void saveSettingsTo(final NodeSettingsWO settings) {
m-taille-sel.saveSettingsTo(settings) ;
m-taille.saveSettingsTo(settings) ;
//Méthode validateSettings
protected void validateSettings(final NodeSettingsRO settings)
throws InvalidSettingsException {
m-taille-sel.validateSettings(settings) ;
m-taille.validateSettings(settings) ;
//Méthode loadValidatedSettings
protected void loadValidatedSettingsFrom(final NodeSettingsRO
settings)
throws InvalidSettingsException {
m-taille-sel.loadSettingsFrom(settings) ;
m-taille.loadSettingsFrom(settings) ;

Bibliographie

[kni,] https://www.knime.org.

[Agrawal et al., 1994] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[Aseervatham, 2007] Aseervatham, a. (2007). *Apprentissage à base de Noyaux Sémantiques pour le traitement de données textuelles*. PhD thesis, Université Paris-Nord-Paris XIII.

[BELLAOUAR, 2018] BELLAOUAR, S. (2018). *Noyaux de mots et d'arbres : efficaité et unification*. PhD thesis, UNIVERSITE AMAR TELEDJI-LAGHOUAT.

[Bellaouar et al., 2017a] Bellaouar, S., Cherroun, H., Nehar, A., and Ziadi, D. (2017a). 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. ACM.

[Bellaouar et al., 2017b] Bellaouar, S., Cherroun, H., and Ziadi, D. (2017b). Efficient geometric-based computation of the string subsequence kernel. *Data Min Knowl Disc*.

[Christopher, 2016] Christopher, M. B. (2016). *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York.

[Gupta et al., 2011] Gupta, V. et al. (2011). Recent trends in text classification techniques. *International Journal of Computer Applications*, 35(6).

[Hechenbichler and Schliep, 2004] Hechenbichler, K. and Schliep, K. (2004). Weighted k-nearest-neighbor techniques and ordinal classification.

[Hoare, 1996] Hoare, T. (1996). *Unification of theories : A challenge for computing science*, pages 49–57. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Hopcroft and Ullman, 1979] Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

[Huang, 2009] Huang, T.-M. (2009). *Kernel Based Algorithms for Mining Huge Data Sets : Supervised, Semi-supervised, and Unsupervised Learning.* Springer-Verlag, Berlin, Heidelberg.

[Kuich and Salomaa, 1986] Kuich, W. and Salomaa, A. (1986). *Semirings, automata, languages.* EATCS monographs on theoretical computer science. Springer-Verlag, Berlin, New York, Heidelberg.

[Kuksa et al., 2009] Kuksa, P. P., hsi Huang, P., and Pavlovic, V. (2009). Scalable algorithms for string kernels with inexact matching. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 881–888. Curran Associates, Inc.

[Leslie et al., 2002] Leslie, C., Eskin, E., and Noble, W. S. (2002). The spectrum kernel : a string kernel for svm protein classification. *Pac Symp Biocomput*, pages 564–575.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *J. Mach. Learn. Res.*, 2 :419–444.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning.* McGraw Hill series in computer science. McGraw-Hill.

[Mohri, 2009] Mohri, M. (2009). Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer.

[Mohri et al., 1996] Mohri, M., Pereira, F. C. N., and Riley, M. (1996). 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.*

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1) :81–106.

[Salzberg, 1994] Salzberg, S. L. (1994). C4. 5 : Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3) :235–240.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3) :210–229.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.

[Turing, 1995] Turing, A. M. (1995). Computers &amp ; thought. chapter Computing Machinery and Intelligence, pages 11–35. MIT Press, Cambridge, MA, USA.

[Yang and Wu, 2006] Yang, Q. and Wu, X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology and amp ; Decision Making*, 05(04) :597–604.