RFreeStem : Une méthode de racinisation indépendante de la langue et sans règle

RFreeStem: A language and rule-free stemmer

Xavier Baril[1], Oihana Coustié[1,2], Josiane Mothe[2,3], Olivier Teste[2,4]

[1] Airbus SAS Operations, Toulouse, France, prenom.nom@airbus.com
[2] IRIT UMR5505 CNRS, Toulouse, France, prenom.nom@irit.fr
[3] ESPE, UT2J, Université de Toulouse, France
[4] IUT Blagnac, UT2J, Université de Toulouse, France

RÉSUMÉ. La racinisation est un pré-traitement essentiel dans de nombreuses tâches de fouille de texte. Les algorithmes les plus utilisés sont basés sur l'application successive de règles spécifiques à la langue. Cette construction les rend fortement dépendants de la langue d'application. Par opposition, les raciniseurs basés sur des corpus présentent souvent une importante complexité algorithmique, les rendant peu efficaces. Ils ne mettent pas non plus nécessairement à disposition les racines extraites, pourtant requises pour certaines tâches de traitement de texte. Nous proposons ici une nouvelle approche, appelée RFreeStem, qui se base sur l'étude d'un corpus et peut être appliquée à différentes langues. L'implémentation de notre méthode est flexible et efficace, car basée sur un unique parcours des n-grammes. Nous détaillons également une méthode d'extraction des racines. Nos expériences montrent que RFreeStem améliore les résultats des tâches de traitement de texte, plus encore que la référence de l'état de l'art, Porter, tout en proposant une racinisation sur des langues peu dotées, où aucune version de Porter n'est implémentée.

ABSTRACT. With the large expansion of available textual data, text mining has become of special interest. Due to their unstructured nature, such data require important preprocessing steps. Among them, stemming algorithms conflate the variants of words into their stems. However, the most popular algorithms are rule-based, and therefore highly language-dependent. In contrast, corpus-based stemmers often exhibit significant algorithmic complexity, making them inefficient. They do not necessarily provide the extracted stems either, which are required for certain text mining tasks. We propose a new approach, RFreeStem, that is corpus-based and can therefore be applied on many languages. The implementation of our method is flexible and efficient, since it relies on a single running through the words' n-grams. We also detail a method to extract the stems. Our experiments show that RFreeStem improves the results of text mining tasks, even more than the Porter reference, while providing a stemming solution on poorly endowed languages, which do not benefit from a version of Porter.

MOTS-CLÉS. Système d'information, fouille de texte, recherche d'information, analyse de sentiments, racinisation.
KEYWORDS. Information systems, text mining, information retrieval, sentiment analysis, stemmer, NLP.

1. Introduction

Au cours des dernières décennies, les progrès considérables des technologies et des recherches sur le Web ont conduit à une forte augmentation de la disponibilité de documents lisibles par machine [MdAIR14]. Ces données non structurées, souvent textuelles, représentent une intéressante opportunité d'extraction d'informations. Cependant, l'absence de structure rend la représentation de ces données difficiles, et implique souvent une quantité importante de pré-traitements nécessaires. Parmi les nombreuses tâches de pré-traitement pour l'analyse de texte, l'idée de regrouper les mots par leur racine syntaxique est apparue depuis la fin des années 60 [Lov68] et est maintenant quasi systématiquement utilisée [MPGP19].

La *racinisation* peut être décrite comme le processus de remplacement d'un mot par sa racine morphologique (aussi appelée *radical*). Un tel algorithme est appelé *raciniseur*. La racine peut être une sous-chaîne ou une concaténation de sous-chaînes du mot, ou même une sous-chaîne modifiée, comme dans la méthode de Porter [Por80], qui dérive le mot *happy* en *happi*. Plus que l'obtention de la racine

elle-même, le principal résultat d'un raciniseur se situe dans les groupes créés par le regroupement de mots issus de la même racine [FBY92]. Par exemple, l'algorithme de Porter regroupe sous la même racine, *happi*, les mots *happy* et *happier*. Le regroupement des mots réduit automatiquement la taille du vocabulaire puisqu'il supprime les variantes morphologiques des mots de l'ensemble étudié [J+11].

L'hypothèse faite ici est que les mots liés morphologiquement ont des significations similaires et représentent le même concept. Cette hypothèse est toutefois remise en question par les articles de recherche proposant des algorithmes de lemmatisation. Moral décrit la *lemmatisation* comme une approche linguistique basée sur une partie de la parole et le contexte [MdAIR14]. Ces méthodes sont moins dépendantes des variantes morphologiques des mots que les raciniseurs. Elles s'appuient souvent sur des dictionnaires ou des thésaurus (liste de mots associés à une notion). Même si l'intégration de notions sémantiques permet une représentation plus précise, ces algorithmes démontrent d'importantes complexités algorithmiques, associées à des temps de calcul et d'utilisation prohibitifs pour certaines applications [J+11]. De plus, la disponibilité de ces dictionnaires sémantiques de référence n'est pas nécessairement garantie, en particulier pour des langues peu outillées [YMA20b]. La lemmatisation ne sera donc pas détaillée dans cet article.

La racinisation est devenue une étape presque incontournable dans certaines tâches de fouille de texte. Parmi ces dernières, les algorithmes de *Recherche d'Information* (IR) analysent automatiquement les documents pour extraire des informations et répondre aux requêtes des utilisateurs [Hul96] : puisque la racine représente un concept plus large que le mot, plus de documents devraient correspondre à la requête. D'autres domaines d'extraction de texte utilisent largement le pré-traitement des racines. Jivani décrit la racinisation comme une exigence pour de nombreuses applications de *traitement automatique du langage naturel* (TALN) comme le regroupement et la catégorisation de documents ou l'analyse des sentiments (c'est-à-dire le processus pour décrire automatiquement les sentiments exprimés dans un message) [J+11].

Les raciniseurs les plus utilisés sont basés sur l'application de règles lexicales sur les mots afin d'en supprimer les parties non-pertinentes. Ces règles reposent sur les formes morphologiques des mots et sont donc nécessairement spécifiques à la langue. Toutefois, si les règles des langues connues et largement parlées sont abondamment spécifiées et établies, il est difficile de trouver une implémentation de ces algorithmes pour des langues ou dialectes peu dotés d'outils automatiques.

Par ailleurs, le pouvoir de la racinisation est hétérogène d'une langue à l'autre. En effet, le nombre de règles nécessaires, ainsi que la qualité et l'efficacité du raciniseur semblent fortement dépendre du langage sur lequel il est appliqué [KP96]. Pour être plus précis, les études linguistiques se distinguent communément en deux types structurels [MdAIR14] :

- Les langages analytiques (ou isolants), avec peu de structures morphologiques, où les variantes sont plus susceptibles d'être exprimées par des mots d'aide que par des variations de mots (par exemple chinois, vietnamien, anglais moderne) ;
- Les langages flexionnels, avec de fortes structures morphologiques. Parmi eux, on trouve :
  - Les langues fusionnelles, comme les langues latines (français, italien) ou germaniques (allemand, néerlandais, suédois, [BR04]). Elles utilisent de nombreux préfixes et suffixes pour modifier la nature grammaticale ou même le sens des mots ;
  - Les langues agglutinantes, comme le finnois, le japonais, la langue berbère ou le basque. Dans ces langues, les suffixes ne représentent pas seulement des inflexions, mais peuvent également

être la concaténation d'autres morphèmes. Par exemple, en basque, *etxe* signifie *maison* et *etxean* signifie *dans la maison*.

Au regard de ces différents types de langues, Jivani affirme que la racinisation s'est avérée plus efficace sur les langues à morphologies complexes, c'est-à-dire sur les langues flexionnelles [J+11]. Braschler [BR04] arrive à la même conclusion avec son étude sur les langues germaniques. Pourtant, sur de telles langues, un grand nombre de règles seraient nécessaires, afin de supprimer tous les affixes, ce qui rendrait le raciniseur peu flexible, difficile à interpréter et implémenter. Finalement, les raciniseurs ont plus d'intérêt à être appliqués sur les langues les plus flexibles, mais les raciniseurs basés sur des règles sont difficilement applicables à de telles langues.

La racinisation basée sur un corpus représente une alternative intéressante. Les mots sont traités au sein d'un corpus de textes, regroupés par similitude et une racine est extraite du groupe ainsi créé. Par opposition aux raciniseurs basés sur des règles, ces méthodes proposent un unique algorithme quelque soit la langue utilisée, et modulent leur comportement à l'aide de paramètres. Plus précisément, Frakes et Fox [FF03] définissent la *force* d'un raciniseur comme sa capacité à regrouper de nombreux mots sous la même racine (on parle de raciniseur fort ou léger, et de racinisation forte ou légère). Alors que les raciniseurs basés sur des règles ont une force fixe, dépendante des règles et de la langue, les raciniseurs basés sur des corpus peuvent moduler leur force à l'aide de leurs paramètres, pour s'adapter à la langue et au corpus étudiés.

Si le principal effort de tels algorithmes consiste à générer ces groupes, il reste toutefois nécessaire de fournir la racine, représentante du groupe de mots. En effet, sans faire nécessairement partie des mots de la langue étudiée [J+11], il peut être crucial que ces racines soient lisibles et interprétables par l'homme dans des tâches comme l'expansion de requêtes ou la recherche de dictionnaire [Hul96]. Outre l'algorithme de regroupement choisi, les raciniseurs basés sur des corpus intègrent donc généralement une seconde phase d'extraction de la racine des groupes de mots.

Par ailleurs, les raciniseurs basés sur des corpus de la littérature reposent sur des méthodes de regroupement avec d'importantes complexités en espace. Ils nécessitent en effet souvent des calculs de distances mot à mot, qui implique l'implémentation de matrices quadratiques (relativement au nombre de mots du corpus) [AB74, MMP+07]. Dans le cas du traitement de texte, ces matrices atteignent généralement de grandes dimensions, puisque la plupart des langues occidentales comptent une centaine de milliers de mots. Cette forte contrainte sur les ressources rend ces algorithmes peu adaptés aux corpus de grandes tailles, où une grande partie des mots de la langue est susceptible d'être utilisée.

En outre, l'évaluation des méthodes de racinisation est également une question difficile. Elle peut être effectuée en comparant ses résultats à un ensemble d'étiquettes validées. Dans ce cas, en plus des traditionnelles mesures que sont la précision et le rappel, Paice [Pai94] définit les erreurs de *sur-racinisation* et *sous-racinisation*. L'erreur de sur-racinisation compte le nombre de mots groupés ensemble alors qu'ils ne devraient pas l'être et la sous-racinisation décrit des erreurs où des mots de même racine n'ont pas été regroupés. Paice propose le calcul d'indicateurs sur la base de ces métriques. Cependant, ces calculs supposent l'accès à une vérité de terrain, c'est à dire aux racines labellisées des mots, qui sont difficiles à obtenir dans la pratique, surtout pour les langues peu dotées.

À la lumière de ces observations, notre contribution consiste en un raciniseur sans règles, appelé RFreeStem (pour Rule-Free Stemmer), qui peut être appliqué sur plusieurs langues. RFreeStem est basé

sur un corpus, ce qui le rend adaptable à n'importe quelle langue, notamment aux langues flexionnelles, et ce, sans apport de connaissance linguistique. RFreeStem est une méthode de regroupement hiérarchique par divisions successives de groupes. Il est reconnu que la représentation hiérarchique est particulièrement adapté aux données textuelles, hiérarchiques par nature. Ces divisions sont réalisées par parcours des n-*grammes* d'un groupe — séquences de n lettres consécutives. Ce choix présente deux avantages : (i) il évite de devoir calculer une matrice quadratique de distance, grâce à un parcours unique des n-grammes, contrairement aux autres méthodes basées sur des corpus, (ii) il correspond à une représentation flexible des données textuelles, sans hypothèses fortes sur la position ou l'unicité du radical, contrairement aux méthodes basées sur des règles. De plus, RFreeStem comporte deux hyper-paramètres qui permettent la gestion de la taille des n-grammes ainsi que de la profondeur de la structure hiérarchique créée. Comme cette profondeur est directement liée à la force du raciniseur, RFreeStem peut moduler sa force grâce à ce paramètre, et ainsi s'adapter à de nombreux jeux de données et langues. Enfin, RFreeStem inclut une méthode d'extraction des racines à partir des groupes créés, ce qui assure la disponibilité et l'interprétabilité des racines pour les tâches de fouille de données qui les nécessitent.

Pour faire face aux problèmes d'évaluation mentionnés précédemment, nous proposons d'évaluer RFreeStem en tant que méthode de pré-traitement pour la fouille de données. Nous évaluons également la capacité d'un raciniseur à compresser la taille d'un vocabulaire. Notre évaluation montre que RFreeStem se comporte globalement mieux que l'algorithme de Porter, qui est reconnu comme la référence de l'état de l'art [Por80]. En particulier, RFreeStem améliore de manière significative les résultats de classification de textes sur les langues flexionnelles testées, et est applicable aux langues peu dotées, pour lesquelles la variante de l'algorithme de Porter n'a pas été implémentée.

Dans cet article, nous proposons d'abord de recenser les différents types de méthodes de racinisation dans la littérature dans la section 2., avant de décrire notre méthode RFreeStem dans la section 3.. Nous décrivons dans la section 4. un cadre d'évaluation pour mesurer l'efficacité et la précision de notre raciniseur sur différentes tâches de fouille de texte et différentes langues. Nous y présentons et commentons également les résultats de notre évaluation. Enfin, nous concluons cet article et présentons nos futurs travaux dans la section 5..

2. Etat de l'art

La littérature offre une grande variété d'algorithmes de racinisation qui peuvent être classés principalement en deux types d'approche : (a) les méthodes basées sur des règles, (b) les méthodes basées sur des corpus de textes. Les sous-sections suivantes proposent de recenser certaines des études les plus importantes sur les algorithmes de dérivation dans ces deux catégories.

Méthodes basées sur des règles

La plupart des méthodes populaires de racinisation reposent sur l'application de règles. Ces règles visent généralement à supprimer les affixes (suffixes et préfixes) des mots individuellement, pour en obtenir le radical. En 1968, Lovins [Lov68] a proposé une première méthode d'*élimination des suffixes*. Simple et rapide, l'algorithme fait correspondre la fin du mot avec l'élément le plus long d'une liste de suffixes et applique des règles pour modifier la terminaison correspondante. L'algorithme de Lovins est

réputé peu fiable puisque sa liste de suffixes est basée sur un vocabulaire très technique. Les mots techniques et scientifiques ont pourtant tendance à dériver des langues latines et donc à être très flexionnels, alors que l'anglais moderne commun est plutôt isolant [1].

L'algorithme le plus populaire, fut proposé par Porter en 1980 [Por80]. Avec 5 étapes successives, il est plus lent que son prédécesseur en 2 étapes, mais il s'est avéré être empiriquement plus précis [Wil06], [Pai96], particulièrement pour les tâches de recherche d'information. Initialement, l'algorithme de Porter ne définissait des règles que pour la langue anglaise. Néanmoins, sa popularité a motivé l'implémentation de variantes dans différentes langues (projet `snowball` [Por01]). Cependant, il reste difficile de trouver de telles implémentations pour des langues peu dotées ou très flexionnelles (par exemple, le bahasa indonésien [Tal03], le néerlandais [KP94]). La racinisation de ces langues est toujours une question de recherche actuelle et difficile [YMA20a].

De nombreuses autres méthodes basées sur des règles ont été proposées par la suite. Parmi elles, le raciniseur nommé S-stemmer [Sav06] propose une racinisation légère, qui ne traite que la suppression des formes plurielles, et présente donc peu de puissance de compression. A l'inverse, Paice [Pai90] a proposé un algorithme de racinisation fort, qui applique successivement des règles de suppression et de remplacement. Jivani [J+11] estime que cette méthode a tendance a générer des erreurs de sur-racinisation. Dawson [Daw74] a proposé une adaptation de la méthode de Lovins avec beaucoup plus de suffixes et de règles, ce qui rend malheureusement l'algorithme peu flexible et interprétable [J+11]. Krovetz [Kro93] propose une approche basée sur les inflexions et les dérivations, avec son raciniseur *KSTEM*. La méthode s'appuie sur un lexique sans inflexion pour en déduire et retirer les inflexions, avant d'analyser les dérivations (les variantes qui changent la nature grammaticale des mots). L'auteur reconnaît que la performance dépend fortement du lexique fourni. Nous ajoutons même que trouver un lexique aussi exhaustif est difficile pour les langues peu dotées.

Les méthodes basées sur des règles sont indéniablement les plus utilisées dans la recherche d'informations, grâce à leur efficacité algorithmique et à la simplicité de leur mise en oeuvre [Har91]. Néanmoins, l'adaptation de ces méthodes à des langages très flexionnelles conduirait à un grand nombre de règles nécessaires. Par conséquent, d'autres articles de recherche sur la racinisation sont enclins à étudier d'autres approches que celles basées sur des règles.

Méthodes basées sur le corpus de textes

Afin de faire face à la forte dépendance linguistique des techniques de racine basées sur des règles, certaines méthodes proposent des raciniseurs basés sur des corpus de textes, qui sont donc adaptables à différentes langues, et différents jeux de données. La plupart de ces méthodes reposent sur des études statistiques du texte. Hafer [HW74] a développé un algorithme qui coupe les mots en deux parties : si la première partie appartient au corpus de textes, cette première partie devient le radical. L'algorithme de découpage est basé sur l'évolution, au sein des lettres du mot, de la variété du successeur : le nombre de caractères distincts qui suivent une chaîne dans tous les mots du corpus de textes. D'une part, les regroupements de mots qui en résultent dépendent fortement du corpus de textes. D'autre part, choisir

1. La langue anglaise est en fait légèrement flexionnelle, en raison de son héritage du vieil anglais. Ainsi, l'anglais moderne est moins isolant que des langues comme le mandarin ou l'indonésien [MdAIR14].

systématiquement la première partie du mot comme radical semble être une approche biaisée, dépendante de la langue. En effet, la méthode est initialement développée pour l'anglais, qui présente généralement plus de suffixes que de préfixes, ce qui est moins vrai pour les langues latines.

Une autre méthode, le raciniseur N-Gram proposée par Adamson [AB74], utilise une méthode de regroupement de chaînes communes pour créer des groupes de mots : un regroupement hiérarchique à liaison unique des mots. Pour évaluer la distance par paires de mots, les auteurs ont utilisé une distance de *Dice*, correspondant au nombre de bigrammes partagés distincts — séquence de deux lettres consécutives. L'utilisation d'une méthode de regroupement non supervisée rend le raciniseur dépendant du corpus de textes utilisé, mais offre également un moyen très flexible de gérer la force du raciniseur. Néanmoins, le regroupement à liaison unique est connu pour son importante complexité algorithmique en espace, en raison de la création d'une matrice de distance quadratique. De la même manière, *YASS* (Yet Another Stripping Stemmer), décrit dans [MMP+07], regroupe les mots avec des méthodes de regroupement hiérarchique. De plus, YASS définit des métriques de distance entre les mots qui encouragent la détection et suppression de suffixes longs. Selon [J+11], la recherche de suffixe long rend la méthode plus adaptée aux langues riches qui sont flexionnelles et riches en suffixes.

Enfin, Sorticut et Och [SO15] proposent une méthode hybride, puisqu'elle vise à apprendre automatiquement des règles, en se basant sur un corpus de textes. L'apprentissage non-supervisé sur le corpus rend cette méthode flexible et potentiellement adaptable à de nombreuses langues. Toutefois, les seules règles recherchées concernent la suppression d'affixes. Cette méthode se base donc sur l'hypothèse forte que les radicaux forment un bloc unique entouré, éventuellement, d'un préfixe et d'un suffixe. Les mots composés ne vérifient pourtant pas cette hypothèse, et sont fréquents dans des langues comme l'allemand.

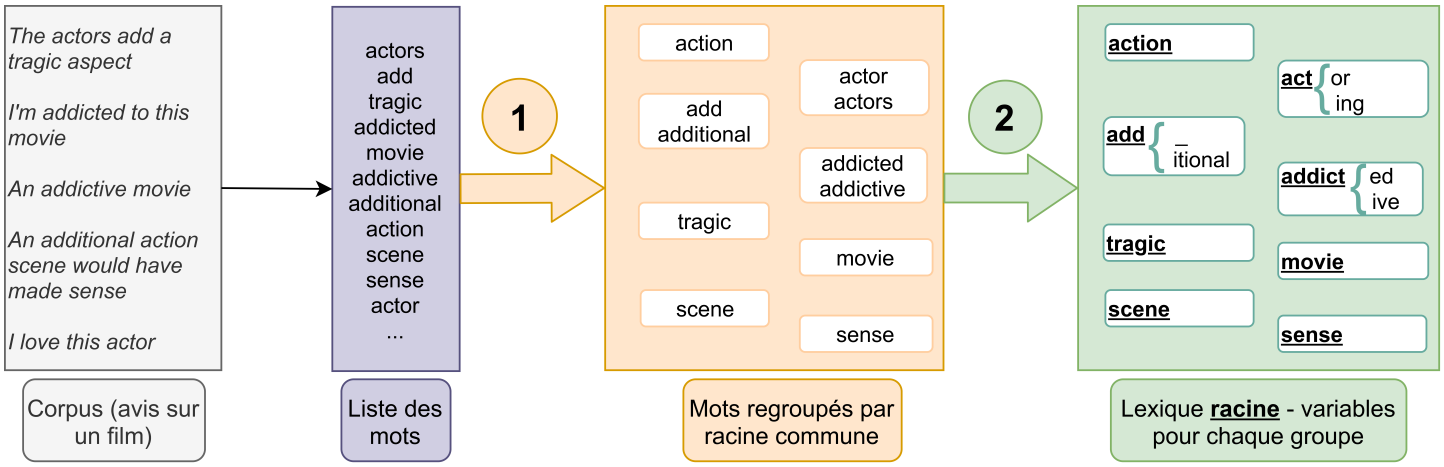
De notre lecture de la littérature, certaines caractéristiques apparaissent comme intéressantes et ont constitué la base de la méthode que nous proposons : (a) une méthode sans règle offre la possibilité d'appliquer le raciniseur à de nombreuses langues (b) les méthodes de regroupement non supervisées permettent d'adapter la racinisation au corpus (et à la langue) (c) les méthodes de regroupement basées sur des calculs de distance mot à mot sont trop coûteuses en ressources. Au regard de ces observations, nous détaillons les caractéristiques implémentées pour l'algorithme RFreeStem.

3. RFreeStemer : un nouveau raciniseur sans règle

Notre étude des méthodes existantes montrent des limitations communes que nous détaillons, et expliquons les choix d'implémentation de notre méthode pour y remédier.

Flexibilité du raciniseur : Les raciniseurs basés sur des corpus peuvent être adaptés à plusieurs langues, contrairement aux raciniseurs basés sur des règles. Ils sont en effet souvent basées sur des apprentissages non-supervisés. L'utilisation de paramètres peut notamment être un atout pour améliorer la modulation de la méthode, et en particulier, la force du raciniseur. D'une manière générale, l'adaptabilité d'un raciniseur à plusieurs langues nécessite de ne pas intégrer de fortes hypothèses sur la structure des mots (hypothèse sur la taille des suffixes, sur la position de la racine…). Pour répondre à ces exigences, nous proposons un raciniseur basé sur l'étude d'un corpus, qui adopte une représentation hiérarchique des mots basée sur l'étude des n-grammes. RFreeStem construit une organisation hiérarchique des groupes par division successives, grâce à un apprentissage non-supervisé. Notre raciniseur contient deux hyper-

Figure 1.: Représentation de l'approche générale des raciniseurs basés sur des corpus de textes dont RFreeStem. Après l'extraction des mots du corpus, l'étape (1) consiste à regrouper les mots de racine commune. Son implémentation pour RFreeStem est détaillée dans la section 3.2.. L'étape (2) consiste à extraire, pour chaque groupe créé, la racine commune des mots du groupe. Pour RFreeStem, nous présentons le résultat sous forme d'un lexique racine - parties variables, dont nous détaillons l'extraction dans la section 3.3.



.

paramètres : n, la taille des n-grammes étudiés, et h, un paramètre permettant la gestion de la profondeur de la représentation hiérarchique, directement liée à la force du raciniseur.

Efficacité algorithmique : La plupart des méthodes basées sur le regroupement de mots reposent sur le calcul de distances entre paires de mots, nécessitant la création d'une matrice quadratique, qui pose des problèmes de complexité en espace. Pour éviter ce calcul, RFreeStem utilise une heuristique. La génération de l'organisation hiérarchique des groupes est réalisée par divisions successives des groupes, comme nous le présenterons dans la sous-section 3.2. A chaque étape de division, les n-grammes des mots du groupe sont parcourus une unique fois, ce qui rend notre algorithme efficace et ne requiert pas une utilisation excessive des ressources informatiques.

Disponibilité des radicaux : Alors que les méthodes basées sur des règles génèrent directement les radicaux associés aux mots, les méthodes basées sur des corpus consistent plutôt à regrouper les mots. L'extraction de la racine doit donc être faite a posteriori. Ainsi, une seconde étape est ajoutée à notre algorithme de regroupement, représentée en vert (2) dans la Figure 1, et détaillée dans la sous-section 3.3.. Nous y proposons un protocole pour déduire la racine commune d'un groupe de mots généré par RFreeStem.

Finalement, le raciniseur RFreeStem comprend deux étapes, présentées dans la Figure 1 : après avoir extrait tous les mots du corpus de textes étudié, ces mots sont regroupés — au moyen d'une représentation hiérarchique, détaillée dans la sous-section 2. Chacun des groupes est ensuite traité pour en extraire les parties fixes, qui constituent la racine commune, et les parties variables. Nous proposons de générer une représentation sous forme de lexique, qui permet une identification claire de la racine et des différentes variantes présentes dans le groupe. Nous décrivons formellement dans cette section l'implémentation des deux étapes précédemment citées.

3.1. *Préliminaires*

Notations et définitions

Nous définissons une collection d'objets uniques, non ordonnés, et de même type comme un ensemble, décrit par des accolades et nommé par une lettre majuscule (par exemple $E = \{e, \forall e \in E\}$). Nous définissons aussi la séquence comme une collection d'objets de même type et ordonnés (non nécessairement uniques). Une séquence est notée par une lettre minuscule et décrite par la série de ses éléments (par exemple $s = (s_i)_{i<|s|}$). Nous adoptons la notation $[\![a; b]\!]$ pour désigner la séquence des entiers naturels compris entre a et b (bornes a et b incluses).

La séquence et l'ensemble sont munis de l'opération de taille, notée $|.|$, qui renvoie le nombre d'éléments de l'ensemble ou de la séquence.

En tant que collection ordonnée, la séquence est également munie des opérations d'ajout et de retrait `ajouter_tête`, `ajouter_queue`, qui insère un élément au début ou à la fin de la séquence, ainsi que `retirer_tête` et `retirer_queue`, qui supprime et retourne le premier ou dernier élément de la séquence. Nous ajoutons les notations d'accès aux valeurs de la séquence : la notation $s[i]$ peut être employée pour s_i, et par extension, $s[i, j]$ désigne la sous-séquence de s allant de l'indice i à j. Nous définissons la fonction `ordonner`, qui prend en entrée une séquence $s = s_1 s_2 \ldots$ dont les valeurs sont définies sur un alphabet $\mathcal{A}$ et une fonction f. La fonction f est définie sur $\mathcal{A}$, et attribue un score à a, $a \in \mathcal{A}$:

$$\begin{array}{rccc} f & : & \mathcal{A} & \rightarrow & \mathbb{R} \\ & & a & \mapsto & f(a) \in \mathbb{R} \end{array}$$

La fonction `ordonner` retourne une séquence s' qui correspond à la séquence s ordonnée selon les valeurs $f(s_1)$, $f(s_2)\ldots$ Cette fonction sera utilisée pour ordonner une séquence s selon une fonction score. Enfin, comme la séquence autorise les répétitions, nous identifions les occurrences d'une valeur de l'alphabet $\alpha \in \mathcal{A}$. Comme ces occurrences sont ordonnées, nous notons $\alpha_i(s)$ la i^{eme} occurrence du symbole $\alpha \in \mathcal{A}$ dans la séquence s.

Les ensembles quant à eux définissent les opérations d'union et d'intersection de plusieurs ensembles $\{E_i\}$, notée respectivement $\bigcup_i E_i$ et $\bigcap_i E_i$, et qui contiennent respectivement les objets présents dans au moins un des ensembles E_i, et présents dans tous les ensembles E_i. On définit également une partition $P = \{P_i, i < |P|\}$ d'un ensemble E comme un ensemble de sous-ensembles P_i de E tels que

— l'union des parties correspond à E : $\bigcup_i P_i = E$;

— les parties sont deux à deux disjointes : $\forall i, j \neq i < |P|, P_i \cap P_j = \emptyset$.

Nous utilisons également la fonction valeur absolue d'un nombre $x \in \mathbb{R}$, que nous notons `abs` définie par

$$\texttt{abs}(x) = \begin{cases} x & \text{si } x \leq 0 \\ -x & \text{sinon.} \end{cases} \tag{1}$$

Formalisation du contexte

Notons W un ensemble de mots. Soit $w \in W$, un mot de l'ensemble, dont les lettres sont définies sur un alphabet $\mathcal{A}$. Ainsi, le mot w est défini comme une séquence de lettres $w = a_1 a_2 .. a_k$, où $k = |w|$ est la longueur du mot w et $\forall i \in [\![1; k]\!], a_i \in \mathcal{A}$. Quel que soit n, un entier inférieur ou égal à k, un n-gramme

de w est défini comme une sous-séquence de n lettres consécutives de w. Le mot *acting*, de longueur 6, a par exemple pour 2-grammes 'ac', 'ct', 'ti'..., et pour 3-grammes 'act', 'cti', 'tin'... On notera que w a exactement $k - n + 1$ n-grammes, $\forall n \leq k$. Nous notons $\mathcal{N}(w)$ l'ensemble des n-grammes du mot w et $\mathcal{N}(W) = \bigcup_{w \in W} \mathcal{N}(w)$, l'ensemble des n-grammes de tous les mots de W. Enfin, pour un n-gramme $g \in \mathcal{N}(\mathcal{W})$, nous notons W_g le sous-ensemble de mots de W qui contiennent ce n-gramme.

Nous utilisons dans cet article le coefficient *Dice*, défini dans [Kon05]. Ce coefficient est une mesure de distance entre deux mots, qui s'appuie sur la comparaison de leurs n-grammes. Nous utilisons cette mesure pour évaluer l'homogénéité d'un groupe de mots. Pour deux mots w_1 et w_2, le coefficient de *Dice* est défini par :

$$Dice(w_1, w_2) = 2 \times \frac{|\mathcal{N}(w_1) \cap \mathcal{N}(w_2)|}{|\mathcal{N}(w_1)| + |\mathcal{N}(w_2)|}$$

Pour généraliser ce calcul à un groupe de mots W, nous proposons l'extension :

$$Dice(W) = |W| \times \frac{|\bigcap_{w \in W} \mathcal{N}(w)|}{\sum_{w \in W} |\mathcal{N}(w)|} \tag{2}$$

Comme mentionné précédemment, RFreeStem regroupe les mots d'un corpus selon une organisation hiérarchique. Pour décrire cette représentation, nous utilisons le concept de *dendrogramme*, qui est un arbre représentant la génération itérative par divisions successives des groupes de mots. Le dendrogramme est fréquemment utilisé comme diagramme représentant l'organisation hiérarchique des groupes générés. Un dendrogramme D est composé de *noeuds*, dont le premier est la *racine*, notée r_D. Pour un noeud d du dendrogramme D, et un groupe de nouveaux éléments $G = (g_1, g_2 \ldots)$, nous définissons l'opération `ajouter_descendants`(d, G) qui ajoute au noeud d l'ensemble des descendants présents dans G (ajout des nouveaux noeuds et création des liens de parenté).

Énoncé du problème

Dans le cadre notre étude, la racinsation consiste à obtenir un double résultat : (i) un regroupement des mots selon leur racine et (ii) la racine de chaque mot. Alors que les raciniseurs basés sur des règles définissent une racine pour chaque mot, ce qui permet une déduction directe des groupes de mots, les raciniseurs basés sur des corpus créent des groupes basés sur la similarité des mots, puis déduisent la racine correspondante. Il s'agit, formellement de répartir les mots de W en plusieurs groupes, c'est-à-dire générer une partition des mots de W, $P = \{P_i, i \leq |P|\}$. Chacune de ces parties forment un groupe de mots P_i, $i \leq |P|$, représenté par une racine commune.

L'identification de la racine commune consiste à trouver les sous-séquences communes à tous les mots du groupe. Notons que notre approche permet de définir plusieurs séquences successives qui forment la racine, ce qui n'est pas le cas de tous les raciniseurs. Néanmoins, il est reconnu que des langues comme l'allemand qui concatènent plusieurs morphismes, contiennent de fait plusieurs racines grammaticales. Dans ce cas, la racine F_i du groupe P_i est une séquence de sous-séquences de lettres de $\mathcal{A}$, $F_i = b_1 b_2 \ldots$ où $\forall j < |F_i|, b_j = a_{j_1} a_{j_2} \ldots$. Chaque sous-séquence qui compose la racine est un bloc de lettres consécutives que l'on retrouve dans chaque mot de P_i. Les blocs doivent également apparaître, pour chaque mot, dans l'ordre indiqué par la séquence F_i. Finalement, F_i doit respecter les contraintes suivantes :

— Quelque soit $w \in P_i$, et quelque soit $b_j \in F_i$, le bloc doit être présent dans chaque mot ;

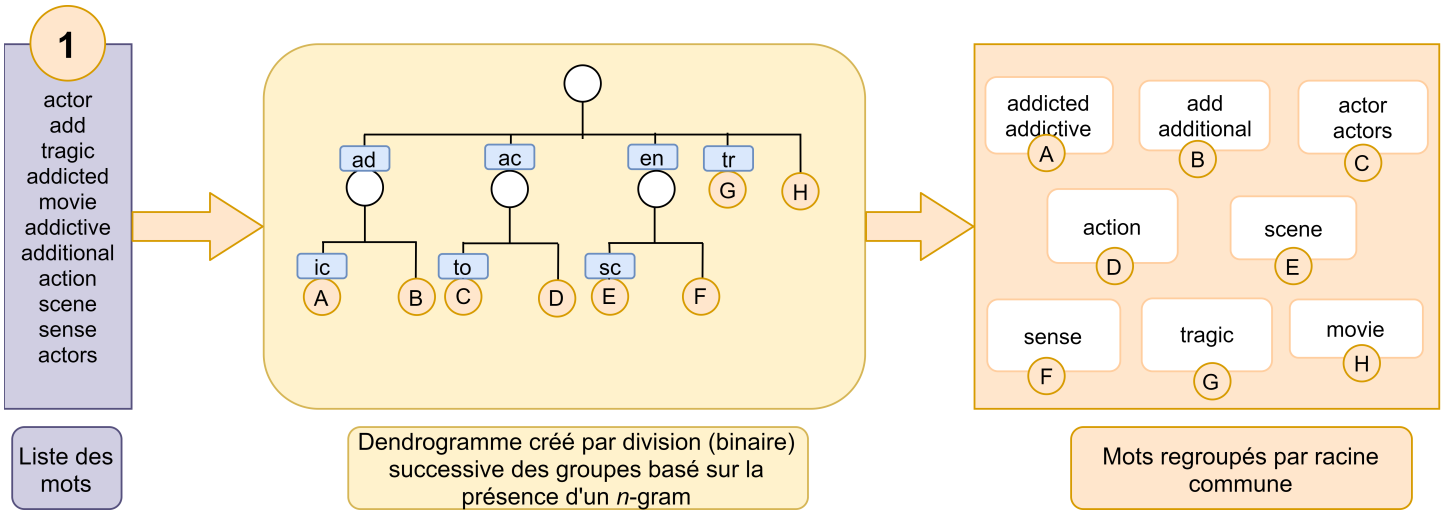


Figure 2.: La méthode de regroupement de RFreeStem (étape (1)), basée sur des divisions successives des groupes de mots en plusieurs sous-groupes, et représentée par un dendrogramme, un diagramme représentant l'organisation hiérarchique des groupes générés. Chaque branche du dendrogramme correspond à une division d'un groupe de mots, basée sur la présence de n-grammes (rectangles bleus) dans les mots. Les noeuds non étiquetés regroupent le reste des mots.

— $\forall w \in P_i$, les blocs doivent apparaître dans le même ordre que dans F_i, sans intersection. Autrement dit, si un bloc $b_j \in F_i$ est trouvé dans w, les blocs suivants dans F_i doivent être trouvés dans la suite du mot.

La formalisation suivante résume les deux exigences :

$$\forall w \in P_i,\ \forall b_i \in F_i,\ \exists (p,q) < |w| \textbf{ tq. } b_j = w[p,q] \textbf{ et } b_{j+1} \subset w[q+1, |w|] \tag{3}$$

La tâche de découverte de la racine consiste donc à trouver des blocs b_j qui respectent ces règles pour former la racine F_i.

3.2. *Etape 1 : Regroupement des mots*

La première étape de notre processus consiste à regrouper les mots du corpus de textes en groupes. Nous utilisons pour cela une méthode de regroupement hiérarchique (généralement plus adaptée pour la modélisation de données textuelles [LSB97]) avec une approche de division : à partir d'un groupe unique contenant tous les mots du corpus, nous le divisons de manière itérative en sous-groupes, comme illustré dans la figure **2**. Nous appelons ces étapes itératives des *étapes de division*. Les divisions sont interrompues lorsqu'un critère d'arrêt est validé, que nous explicitons par la suite. L'algorithme 1 décrit le processus de création du dendrogramme ainsi que l'extraction finale des groupes de mots.

Comme le montre l'algorithme 1, le dendrogramme est initialisé par un noeud racine regroupant l'ensemble des mots. A chaque étape de division, des sous-groupes sont extraits : s'ils valident la condition `continuer()`, liée au critère d'arrêt, ils sont ajoutés au dendrogramme et seront à leur tour parcourus et divisés — la liste **parcours_noeuds** peut être vue comme une file et permet de s'assurer du parcours de l'ensemble des noeuds ajoutés au dendrogramme. Si la condition `continuer()` n'est pas validée, alors les sous-groupes sont rejetés, l'itération s'arrête pour cette branche, et le groupe initial W est désigné comme groupe final.

Algorithm 1: Algorithme de création du dendrogramme et extraction des groupes finaux par divisions successives

input : $\mathcal{C}$: l'ensemble des mots d'un corpus
output: groupes : Un ensemble de groupes de mots

```
D ← Dendrogramme(racine=C);
parcours_noeuds = liste();
parcours_noeuds.ajouter_tête(r_D);
G = ensemble();
while |parcours_noeuds|>0 do
    W ← parcours_noeuds.retirer_queue();
    G ← division(W);
    if continuer(W, G) then
        D.ajouter_descendants(W, G);
        for G_i ∈ G do
            parcours_noeuds.ajouter_tête(G_i);
        end
    else
        groupes ← groupes ∪{W};
    end
end
return groupes;
```

Nous détaillons dans cette sous-section deux implémentations importantes : (i) la méthode `division`, basée sur les n-grammes, qui définit la répartition des mots de W en sous-groupes, et (ii) le critère d'arrêt `continuer` qui permet de gérer la profondeur du dendrogramme.

Description d'une étape de division

À chaque étape de division, un ensemble de mots est divisé en plusieurs sous-groupes. Notre proposition consiste à se baser sur la simple présence d'un n-gramme pour regrouper les mots, dans chaque sous groupe proposé, où n est un hyper-paramètre de notre méthode. Il s'agit donc de choisir un n-gramme susceptible d'être une partie de la racine. Ainsi les mots regroupés auront une partie de leur racine en commun. Nous détaillons les critères retenus pour établir la pertinence d'un n-gramme dans la section suivante, et nous nous contentons pour l'instant de désigner par s la fonction qui attribue aux n-grammes un critère de pertinence. L'algorithme 2 décrit formellement ce processus.

Avec ce nouvel algorithme, illustrons maintenant le fonctionnement de notre méthode sur un exemple complet, correspondant au corpus de mots présenté en Figure **2**, où nous fixons $n = 2$. Le dendrogramme présenté est initialisé avec une racine contenant l'ensemble des mots du corpus. Une étape de division (Algorithme 2) est alors appliquée sur le groupe formé par l'ensemble des mots. On extrait d'abord les n-grammes présents dans les mots du corpus : *ac*, *ct*, *to*, *or*, *ad*, *dd*, *tr*... Ces n-grammes sont ensuite ordonnés selon une fonction de score (décrite dans cette section). Les n-grammes sont parcourus dans l'ordre, et pour chaque n-gramme, les mots qui contiennent ce n-gramme sont regroupés, et exclus de la suite du parcours. Ainsi, si le premier n-gramme est *ad* selon l'ordre issu de la fonction de score, les

Algorithm 2: Etape de division (fonction `division`)

input : W : un ensemble de mots W, n=un entier naturel, s : une fonction de score de n-grammes, lim : un nombre limite de n-grammes à parcourir

output: G : sous-ensembles de mots de W générés par la division

1 n-grammes $\leftarrow \mathcal{N}(W)$;
2 n-grammes $\leftarrow$ `trier`(n-grammes, s);
3 $G \leftarrow$ `ensemble`();
4 **while** $|n\text{-}grammes| > lim$ ***ET*** $|W|>0$ **do**
5 $\quad g \leftarrow n$-grammes.`retirer_tête`();
6 $\quad G \leftarrow G \cup \{W_g\}$; `// ` W_g`: les mots de` W `contenant le` n`-gramme` g
7 $\quad W \leftarrow W \backslash W_g$; `// les mots de` W `sans` g
8 **end**
9 **return** G;

mots *add*, *addictive*, *additional* et *addicted* seront regroupés. Les n-grammes sont parcourus et génèrent des groupes de mots jusqu'à ce qu'une limite de mots traités soit atteinte (lim, ligne **4** de l'algorithme). La mise en place de cette limite nous permet de ne pas traiter des n-grammes dont les scores seraient trop faibles, qui pourraient créer des groupes peu réalistes. Les mots restants sont regroupés dans un dernier groupe, le noeud sans étiquette de la Figure **2**. L'étape de division s'achève. Conformément à la condition de la ligne **8** de l'algorithme 1, si les groupes créés lors de cette étape satisfont une condition (détaillée dans la suite de cette section), l'étape de division est validée. Les groupes créés sont alors ajoutés comme descendants du noeud étudié, et ajouté au dendrogramme. Ils seront parcourus à leur tour pour une étape de division. Dans le cas contraire, les nouveaux groupes sont rejetés, et le noeud étudié devient une feuille du dendrogramme, qui donnera lieu à la création d'un groupe final.

Détermination de la fonction de score des n-grammes

Il ressort de l'algorithme décrit précédemment que la définition de la fonction qui trie les n-grammes est cruciale. En effet, elle attribue un score à chaque n-gramme, qui servira à ordonner les n-grammes et définir un ordre de parcours. Un n-gramme avec un score élevé sera responsable du regroupement de tous les mots qui le contiennent. Rappelons que, pour un n-gramme $g \in \mathcal{N}(W)$, W_g désigne l'ensemble des mots qui contiennent g. Si g est le premier n-gramme (celui à qui on a attribué le meilleur score) alors les mots W_g seront regroupés ensemble. En revanche, si g a un score plus faible, il est probable que, lorsque vient son tour de regrouper les mots correspondants, certains mots de W_g soient déjà attribués à d'autres groupes. Le groupe généré à partir de g ne sera alors qu'un sous-ensemble de W_g. Par conséquent, si le n-gramme *tion* a un meilleur score que *clas*, alors *classification* sera regroupé avec les mots de suffixe *tion* (comme *imitation*, *attention*) au lieu d'être réuni avec des mots avec lesquels il partage une racine grammaticale, comme *class* ou *classify*. Pour éviter cette situation, nous discutons des caractéristiques qui peuvent définir un "bon" n-gramme :

1. Le nombre de mots contenant ce n-gramme, c'est-à-dire la taille du groupe généré : $|W_g|$. Un n-gramme g associé à une faible quantité $|W_g|$ ne regroupe que quelques mots, ce qui entre en conflit avec l'approche de division de notre regroupement hiérarchique. Nous nous intéressons donc aux n-grammes regroupant le plus de mots. Nous observons cependant que les quelques n-grammes les

plus fréquents sont souvent des affixes courants (par exemple *tion*). Pour éliminer ces n-grammes, nous proposons une taille de référence l_{ref} qui correspond à la taille maximum des groupes W_g, après suppression des $q\%$ les plus élevés (ce qui peut être vu comme l'élimination des valeurs extrêmes) ;

2. L'homogénéité du groupe généré W_g, qui peut être évaluée avec :
   (a) La moyenne des distances des mots deux à deux. Ces distances peuvent être estimées avec une métrique adaptée aux données textuelles, comme la distance de Levenshtein ;
   (b) Le nombre de n-grammes distincts dans le groupe généré W_g. Un groupe homogène devrait avoir de nombreux n-grammes en commun, et donc peu de n-grammes différents. Il s'agit donc de favoriser des n-grammes g avec de faibles valeurs $|\mathcal{N}(W_g)|$, où $\mathcal{N}(W_g) \subset \mathcal{N}$ est le sous-ensemble des n-grammes de W que l'on retrouve dans les mots de W_g. En d'autres termes, il s'agit des n-grammes qui partagent au moins un mot en commun avec le n-gramme g ;
   (c) Une distance basée sur les n-grammes, comme le coefficient de *Dice*, qui calcule le taux de n-grammes communs.
   (d) Le nombre de lettres communes dans les mots du groupe généré W_g. Cette idée se rapporte à la création ultérieure de lexique que nous proposons : un nombre élevé de lettres communes signifie une méthode plus longue.
3. L'hétérogénéité du groupe généré. Une mesure peut être le nombre d'autres groupes générés contenant le n-gramme. En effet, nous imaginons facilement que *tion* est susceptible d'apparaître dans presque tous les groupes. Il est donc préférable d'éviter de regrouper des mots à partir de n-grammes qui sont présents dans de nombreux groupes générés. Toutefois, observons que la relation "g_1 est présent dans l'ensemble des n-grammes du groupe généré par g_2" est réciproque : $g_1 \in \mathcal{N}(W_{g_2}) \iff g_2 \in \mathcal{N}(W_{g_1})$. Par conséquent, le nombre de groupes dans lesquels g_1 est présent n'est autre que le nombre de n-grammes présents dans $\mathcal{N}(W_{g_1})$. Or, cette quantité correspond à $\left|\mathcal{N}(W_{g_1})\right|$, qui est déjà identifiée comme un critère à minimiser (point 2) ;
4. Une autre intuition en affinant cet algorithme sur des jeux de données anglais est de pénaliser les n-grammes qui sont susceptibles d'être des affixes (n-grammes dont les positions dans les mots sont souvent soit au début soit à la fin). Nous estimons cependant que ce critère est fortement lié à la langue et donc en contradiction avec notre volonté de proposer une méthode indépendante de la langue.

Ces discussions nous portent à nous intéresser aux critères 1 et 2. Le coefficient de *Dice*, défini par l'équation 2 semble particulièrement intéressant puisqu'il adopte une mesure de distance en accord avec notre étude des n-grammes. Ce coefficient pénalise également l'hétérogénéité intra-groupe telle que précédemment définie : les groupes avec de nombreux n-grammes non-communs présentent un écart important entre le numérateur et dénominateur. Il aura donc tendance à favoriser des groupes extrêmement homogènes, notamment, il attribue le meilleur score de $Dice = 1$ aux groupes ne contenant qu'un seul mot. Pour équilibrer notre score, nous introduisons une notion de taille-objectif, véhiculée par le critère 1. Finalement, nous proposons de noter les n-grammes avec la fonction :

$$s(g) = \frac{1}{2} \times \Big(1 - Dice(W_g) + \texttt{abs}\Big(\frac{|\mathcal{N}(W_g)| - \ell_{ref}}{\ell_{ref}}\Big)\Big) \tag{4}$$

Dans cette formule, le premier membre représente l'homogénéité du groupe : une faible distance de *Dice* entre les n-grammes d'un groupe généré reflète un groupe avec des mots similaires. Le deuxième

membre pénalise l'écart entre la taille de référence et la taille réelle d'un groupe généré. Notons que le score $s(g)$ doit être le plus faible possible pour qu'un n-gramme soit considéré comme une racine. Sur la base de plusieurs expériences, nous montrons que cette fonction sera discriminante pour la notation des n-grammes.

Critère d'arrêt de division

Comme présenté dans l'algorithme 1, les divisions successives sont interrompues lorsqu'un critère d'exigence d'homogénéité est atteint : la fonction `continuer` retourne VRAI si ce critère n'est pas encore atteint (il faut donc continuer le processus divisif pour obtenir des groupes plus homogènes) et FAUX sinon (le groupe d'origine est suffisamment homogène, il peut former un groupe). Il apparaît donc que la définition de ce critère d'arrêt contrôle l'exécution de notre méthode ainsi que son résultat final. Si ce critère est exigeant, l'itération se poursuivra jusqu'à ce que des groupes extrêmement homogènes soient créés, le dendrogramme sera donc plus grand, et plus long à créer. En revanche, un critère d'homogénéité faible interrompra rapidement l'exécution, générant un plus petit dendrogramme avec des groupes moins homogènes et plus grands.

Nous définissons un critère d'homogénéité relatif, pour ne pas être dépendant de l'ordre de grandeur de la mesure d'homogénéité choisie. En l'occurrence, nous exigeons une amélioration relative d'un score d'homogénéité. En effet, les étapes divisives créent des groupes de plus en plus homogènes, puisqu'ils partagent de plus en plus de n-grammes communs. Par souci de cohérence, nous proposons d'utiliser comme mesure de cette homogénéité le coefficient $Dice$. Pour faciliter les calculs, nous nous intéressons au premier groupe, généré par le n-gramme ayant le score le plus élevé. Notons ce groupe W_1, qui a nécessairement un coefficient de $Dice$ plus élevé que celui du groupe W. L'amélioration de ce critère d'homogénéité est donc assurée lors du processus de division. Nous exigeons un pourcentage d'amélioration h pour valider la division. Nous proposons finalement le critère d'arrêt suivant :

$$\texttt{continuer}(W, sous\_groupes) \iff \textbf{\textit{Dice}}(W) < h \cdot \textbf{\textit{Dice}}(W_1) \tag{5}$$

Nous précisons que h est un hyper-paramètre de la méthode. A défaut de choisir une valeur a priori pour h, nous proposons de déterminer expérimentalement la meilleure valeur, comme nous le faisons pour l'hyper-paramètre n.

3.3. *Etape 2 : Génération de la racine et du lexique*

La seconde étape de notre méthode vise à générer un dictionnaire à partir des groupes précédemment créés. Dans chaque groupe homogène W, cette étape consiste à trouver les parties fixes et variables dans les mots du groupe. Une partie fixe est définie comme une séquence de lettres consécutives qui peuvent être trouvées dans chaque mot du groupe. Par exemple, le groupe formé par les deux mots *classification* et *unclarified* a deux parties fixes : *cla* et *ifi*. Les parties variables sont simplement les parties qui ne sont pas fixes. Dans l'exemple, nous trouverions respectivement {*ss*, *cation* } et {*un*, *r*, *ed* }. Le but est d'obtenir un lexique recensant les racines du groupe (parties fixes) et toutes les valeurs possibles prises par les parties variables, comme le montre la dernière partie de la figure **1**. Dans l'exemple des mots *classification* et *unclarified*, nous voulons également aligner les parties variables correspondantes, pour

obtenir :

$$\left\{ \begin{array}{l} \text{un} \\ \_ \end{array} \right. \textbf{cla} \left\{ \begin{array}{l} \text{r} \\ \text{ss} \end{array} \right. \textbf{ifi} \left\{ \begin{array}{l} \text{ed} \\ \text{cation} \end{array} \right.$$

Cependant, le nombre de variables n'est pas constant. Afin de pouvoir les aligner, nous affinons notre définition : soit $w \in W$ un mot du groupe, et F la séquence des parties fixes du groupe W, quelque soit $i < |F|$, la partie variable $v_i(w)$ est le n-gramme situé entre les parties fixes $F[i]$ et $F[i+1]$ dans w. Par exemple ici, nous avons $v_1(\textit{unclarified}) = \textit{r}$ et $v_1(\textit{classification}) = \textit{ss}$ alors que $v_0(\textit{unclarified}) = \textit{un}$ et $v_0(\textit{classification}) =$ « _ ». Quelque soit $i < |F|$, nous définissons l'ensemble $v_i = \{v_i(w), w \in W\}$, qui contient les valeurs prises par les parties variables qui suivent la partie fixe $F[i]$.

De cette définition, il apparaît que les parties variables sont déduites de l'identification des parties fixes. Il est donc d'abord nécessaire d'extraire les parties fixes. Pour illustrer la difficulté de cette tâche, nous étendons notre exemple à $W = \{$*clarity*, *claimed*, *cleansing*, *classical*, *unclarified*, *declaration*, *classification* $\}$. Bien entendu, l'obtention d'un tel groupe hétérogène n'est pas souhaitable. Cependant, au cas où la fonction de notation ne se comporterait pas comme prévu, la méthode de génération de lexique doit être capable de gérer cette hétérogénéité. Pour trouver des parties fixes, nous recherchons d'abord les lettres qui sont communes à tous les mots de W, ici, *c*, *l*, *a* et *i*. En définissant ces lettres comme unique source de parties fixes, le mot *clarity* se diviserait naturellement avec les parties fixes $F_1 = \{\textit{cla}, i\}$ et les parties variable $v_1(\textit{clarity}) =$*r* et $v_2(\textit{clarity}) =$*ty*. Néanmoins, cette répartition est incompatible avec le mot *cleansing*, qui ne contient pas la partie fixe $F_1[1] = \textit{cla}$. Nous concluons qu'il est nécessaire de trouver un ensemble de parties fixes suffisamment générique pour inclure tous les mots. Ici, $F_2 = \{\textit{cl}, a, i\}$ représente une meilleure alternative.

Par ailleurs, les mots *classification* et *unclarified* contiennent deux occurrences de *i*, qui pourraient toutes deux être la partie fixe. Concrètement, la proposition de F_1 (nous utilisons F_1 pour la simplicité de l'illustration, le même problème est observable pour F_2), pour le mot *classification*, donnera les parties variables $v_1(\textit{classification}) =$*ass*, $v_2(\textit{classification}) =$ *fication* ou bien $v_1(\textit{classification}) =$ *ssif*, $v_2(\textit{classification}) =$ *cation*, selon le *i* qui sera interprété comme partie fixe (***clas****si****f****ication* ou ***clas****sifi****c****ation*). Plus généralement, lorsque plusieurs occurrences des lettres communes apparaissent (*c*, *l*, *i* et *a*), nous devons savoir laquelle est une vraie partie fixe. Le tableau suivant recense les fréquences des lettres communes dans chaque mot, pour identifier les mots qui posent problème :

	clarity	*claimed*	*cleansing*	*classical*	*unclarified*	*declaration*	*classification*	min.
c	1	1	1	1	1	1	2	1
l	1	1	1	1	1	1	1	1
a	1	1	1	1	1	2	2	1
i	1	1	1	1	2	1	3	1

Nous identifions comme référence les mots avec les fréquences minimales des lettres communes, c'est-à-dire les colonnes qui sont parfaitement égales à la colonne « min. » du tableau (les quatre premiers mots du tableau). Nous cherchons à identifier, pour les mots *unclarified*, *declaration* et *classification*, les occurrences des lettres communes qui sont des parties fixes, puisque (i) *declaration* et *classification* ont deux occurrences de la lettre commune *a*, (ii) *unclarified* a deux occurrences de la lettre commune *i*, et *classification* en a trois, (iii) *classification* a deux occurrences de la lettre commune *c*. Notons $\alpha_i(w)$ la i^{eme} occurrence de la lettre α dans le mot w : si $w =$ *unclarified*, $i_1(w)$ est le *i* situé entre le *r* et le *f*.

Pour décider quelles occurrences des lettres communes conserver, nous examinons d'abord l'ordre dans lequel les lettres communes apparaissent dans les mots de référence : *c*, *l*, *a*, *i*. On peut alors déduire, pour le mot *classification*, que l'occurrence c_1 correspond à une partie fixe, plutôt que c_2. Cependant, cet ordre ne promeut aucune des occurrences de *a* et de *i* dans les trois mots problématiques. Nous ajoutons donc une autre caractéristique à notre protocole : l'espace entre les parties fixes pour chaque mot de référence. Cette valeur ne peut pas être calculée pour les autres mots, car nous ne saurions pas quelle occurrence choisir. Le tableau suivant recense le nombre de lettres séparant deux lettres communes consécutives dans les 4 mots de référence :

	clarity	*claimed*	*cleansing*	*classical*	moyenne	écart interquartile
c - *l*	1	1	1	1	1	0
l - *a*	1	1	2	1	1.25	0.25
a - *i*	2	1	3	2	2	0.5

Pour chaque couple de lettres communes consécutives, la moyenne des espacements observés sur les mots de référence peut être considérée comme l'espace de référence. Ainsi, les lettres *a* et *i* sont espacés de 2, 1, 3 et 2 lettres dans les 4 mots de référence. En moyenne, cet espace est de 2, et nous pouvons estimer qu'il s'agit de l'écart de référence. Pour le mot *unclarified*, l'espace entre a_1 et i_1 est de 2, et entre a_1 et i_2 est de 4. L'occurrence i_1 correspond donc aisément à la partie fixe. De plus, pour *classification*, nous observons les espaces suivants entre les différentes occurrences de *a* et *i* (éligibles selon l'ordre précédemment établi, *a* avant *i*) :

a-occurrence	*i*-occurrence	occurences gap	expected gap	difference with expected
a_1	i_1	3	2	1
a_1	i_2	5	2	3
a_1	i_3	9	2	7
a_2	i_3	2	2	0

De cette étude, la relation qui correspond le mieux à l'espace de référence est le couple $a_2 - i_3$. Cependant, il faut également considérer l'espace avec les autres lettres communes. En l'occurrence, la différence entre l'espace de référence pour la relation $l - a$ et l'espace observé entre *l* et a_2 est importante. En calculant la moyenne de ces différences sur l'ensemble des relations exigées, nous choisirons plutôt $a_1 - i_2$. Appliquons cette méthode au mot *declaration* :

a-occurrence	observed		difference with expected		mean difference
	l - *a*	*a* - *i*	*l* - *a*	*a* - *i*	
a_1	1	4	0	2	1
a_2	3	2	2	0	1

Ici, en moyenne, les deux occurrences de *a* se voient attribuer le même score : a_1 est favorisé par la relation $l - a$ alors que a_2 correspond mieux à l'espace de référence de la relation $a - i$. Nous ajoutons donc une dernière caractéristique à notre méthode de décision : une mesure de la stabilité des espacements. Dans l'exemple précédent, nous observons que l'espace entre les lettres *l* et *a* a des valeurs plus stables que l'écart entre *a* et *i*. Par conséquent, la différence entre les valeurs réelles et de référence de

l'espace devrait tenir compte de la fiabilité de cet espace de référence. Ainsi, nous proposons de calculer l'intervalle interquartile de chaque espace (dernière colonne du tableau 2). Cet intervalle interquartile est inversement proportionnel au coefficient attribué aux écarts. Dans notre exemple, comme la relation l - a est assez stable, a_2 sera fortement pénalisé de ne pas respecter l'espace de référence. L'occurrence a_1 sera finalement choisie comme partie fixe.

Nous obtenons finalement les parties fixes correspondant à F_2, et pouvons maintenant déduire simplement les parties variables. On obtient, dans notre exemple, le résultat recherché, présenté sous forme de lexique :

$$\left\{\begin{array}{l} \text{un} \\ \text{de} \\ \_ \end{array}\right. \textbf{cl} \left\{\begin{array}{l} \text{e} \\ \_ \end{array}\right. \textbf{a} \left\{\begin{array}{l} \text{r} \\ \text{ns} \\ \text{ss} \\ \text{arat} \\ \_ \end{array}\right. \textbf{i} \left\{\begin{array}{l} \text{ty} \\ \text{med} \\ \text{ng} \\ \text{cal} \\ \text{fied} \\ \text{on} \\ \text{fication} \end{array}\right.$$

La racine correspondante de ce groupe pourrait alors être *_cl_a_i_*. En pratique, nous espérons ne pas obtenir un groupe aussi hétérogène. Dans un groupe bien formé, nous pouvons espérer avoir des racines plus intelligibles comme *_classif_* ou *_clar_*. Cette nouvelle représentation du corpus racinisé pourrait améliorer les résultats de comparaison de textes, qui pourraient être nécessaires dans les tâches de TALN. De plus, les recherches dans le lexique seront plus faciles pour l'utilisateur.

4. Evaluation

Nous évaluons dans cette section notre raciniseur RFreeStem. Notre objectif est de montrer que notre méthode offre de meilleurs résultats que la méthode de Porter, référence actuelle de l'état de l'art, tout en proposant une meilleure puissance de compression. Pour ce faire, nous proposons d'exécuter notre raciniseur et celui de Porter sur des données textuelles brutes avant d'y appliquer une tâche de catégorisation automatique de textes.

4.1. *Environnement d'expérimentation*

Notre évaluation est divisée en deux études expérimentales : (i) premièrement, nous évaluons l'efficacité de notre raciniseur sur la langue anglaise en comparant deux tâches : la catégorisation automatique de textes (**Tâche 1**) et l'analyse de sentiments (**Tâche 2**) ; (ii) nous étudions ensuite l'amélioration produite par la racinisation pour différentes langues, sur la tâche d'analyse de sentiments.

Pour chaque étude expérimentale, nous comparerons les différentes versions de notre algorithme (en raison des valeurs possibles de ces deux paramètres n et h) avec (i) la version de Porter (lorsqu'elle existe) et (ii) la version sans raciniseur. Par souci d'équité de comparaison de résultats, nous sélectionnons aléatoirement 2000 documents pour chaque jeu de données, en respectant les proportions initiales des différentes classes.

Tâche 1 : La catégorisation de texte est une tâche populaire qui vise à déterminer automatiquement la classe d'un document dans un corpus. Pour tester l'efficacité de notre raciniseur en tant qu'étape de prétraitement pour la catégorisation de texte, nous avons appliqué notre méthode au corpus de rapports de cas AustLII [2]. Nous étudions un ensemble de 2000 citations provenant de fichiers XML différents. Ces fichiers contiennent à la fois le texte de la citation légale et une catégorie étiquetée manuellement, parmi les suivantes :

classe	cited	referred to	applied	followed	considered	discussed	Autres(13)
#documents	938	351	228	169	129	88	97
#termes	16 609						

L'ensemble de données est fortement déséquilibré puisque 46,9 % des citations portent l'étiquette *cited*, alors que les deux-tiers des classes (regroupées dans *Autres*) ne représentent que 4,9 % des données. Les jeux de données déséquilibrés sont réputés plus difficiles à traiter, il sera donc intéressant de voir si notre raciniseur simplifie cette tâche.

Tâche 2 : L'analyse de sentiments est devenue une tâche très populaire, en particulier avec la forte expansion des réseaux sociaux et de leurs données textuelles. Le but est d'évaluer les sentiments dans les commentaires ou critiques écrits par des humains. Pour évaluer la capacité de RFreeStem à améliorer les résultats de cette tâche complexe, nous avons choisi un jeu de données de critiques de films fréquemment utilisé [3]. Initialement mentionné dans [PL04], le corpus est encore couramment utilisé aujourd'hui [BKH16], [SZL+18].

Comme la littérature semble s'accorder sur le fait que la racinisation est plus adaptée aux langues flexionnelles, nous proposons de comparer notre raciniseur sur des ensembles de données similaires de différentes langues. Nous avons donc étudié les avis Amazon en français et en allemand [4] pour la tâche d'analyse des sentiments (**Tâche 2**). Puisque les produits sont évalués avec une note de 1 à 5, nous avons extrait des étiquettes catégorielles avec la règle simple de transformation suivante : si la note est strictement supérieure à 3, attribuer une classe positive, sinon, si elle est strictement inférieure à 3, attribuer une classe négative, sinon, attribuer à la classe neutre.

Enfin, puisque notre méthode est basée sur un corpus, elle n'est pas dépendante de la langue et elle peut être appliquée à n'importe quelle langue, en particulier celles qui sont peu dotées en ressources et outils. Nous proposons d'étudier un ensemble de données en ourdou romain [5] pour une analyse des sentiments à 3 classes (**Tâche 2**).

La Table **4.1** présente la répartition des classes des documents pour chaque ensemble de données de la **Tâche 2**, ainsi que le nombre de termes initialement présents. Alors que le corpus anglais correspond à une classification binaire et équilibrée, les trois autres jeux de données contiennent en plus des sentiments positif et négatif le sentiment neutre. Les jeux de données d'Amazon (allemand et français) sont

2. `http://www.austlii.edu.au/` [GCH12] l'utilise pour la catégorisation
3. `http://www.cs.cornell.edu/people/pabo/movie-review-data/`
4. `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`
5. http ://archive.ics.uci.edu/ml/datasets/Roman+Urdu+Data+Set, [SR18]

Langue	Source	Positif		Neutre		Négatif		#termes
Anglais	Revues de films	1000	50%			1000	50%	7 736
Français	Commentaires Amazon	1634	81.7%	177	8.9%	189	9.4%	13 656
Allemand	Commentaires Amazon	1600	80.0%	244	12.2%	156	7.8%	30 986
Ourdou	Tweets	595	29.7%	883	44.2%	582	29.1%	7886

Tableau 4.1.: Présentation des données pour la **Tâche 2** - 2000 documents dans chaque cas.

fortement polarisés puisque plus de 80% des commentaires sont positifs. A l'inverse, le jeu de données en ourdou s'apparente davantage à une distribution gaussienne, avec des classes positive et négative comparables, et une plus grande proportion de commentaires neutres. Enfin, on observe une grande disparité entre le nombre de termes pour chacune des langues. Nous attribuons cette diversité en partie au caractère flexionnel du français et de l'allemand, alors que l'anglais et l'ourdou sont des langues isolantes. De plus, l'allemand compose des nouveaux mots par concaténation de différents morphismes, ce qui a tendance à accroître la taille du vocabulaire employé. Il est donc fort probable que la racinisation ait un impact positif sur la représentation de ces deux langues.

Métriques d'évaluation

Nous rappelons que notre évaluation consiste à observer l'amélioration des tâches de traitement de texte étudiées induite par la racinisation. Pour mesurer cette amélioration, nous calculons différentes mesures d'évaluation. Tout d'abord, la F1-mesure s'exprime grâce aux calculs des vrais positifs (TP), faux positifs (FP) et faux négatifs (FN). Pour un document d dont l'étiquette réelle est l, et catégorisé par l'algorithme dans la classe c, les vrais positifs sont les documents de l qui sont aussi classés dans c. Les faux négatifs sont les documents de l qui ne sont pas classés dans c. Enfin, les faux positifs sont les documents qui ne proviennent pas de la classe l mais qui sont associés à la classe c. Finalement, la F1-mesure s'exprime comme :

$$F1 - mesure = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

.

Nous étudions également la précision de la méthode, définie comme le taux de documents correctement associés à leur classe d'origine. Enfin, nous nous intéressons au pouvoir de compression du raciniseur. Pour le mesurer, nous utilisons l'ICF (Index Compression Factor) [FF03]. L'ICF s'exprime comme $\frac{m-s}{m}$ où m est le nombre total de mots du corpus, et s le nombre de racines différentes proposées (le nombre de groupes créés).

Description du protocole expérimental

Le processus que nous appliquons est le suivant : pour chaque jeu de données, et pour chaque méthode de racinisation évaluée, nous appliquons la méthode de racinisation à l'ensemble des données brutes et générons donc un lexique où chaque entrée est un couple {mot : racine}.

Nous exécutons ensuite un algorithme de classification supervisé sur les données en remplaçant chaque mot par sa racine (ou en gardant les mots d'origine, pour la version sans raciniseur). Nous avons choisi d'implanter un classificateur bayésien naïf, entraîné sur 70 % des données, en respectant la proportion des étiquettes. Les 30 % restants sont prédits avec le classificateur entraîné, et les résultats sont comparés à leurs étiquettes de classe. Les documents sont représentés par une matrice termes / documents, sur laquelle le classificateur est appliqué. Nous sélectionnons uniquement les termes dont la fréquence dans les documents est supérieure à f_{min}, où f_{min} est une variable dont nous étudierons l'impact sur les résultats. Une méthode qui présente des résultats satisfaisants sur de petites valeurs de f_{min} présente l'avantage d'adopter une représentation précise et efficace. Par ailleurs, nous exécutons plusieurs fois chaque calcul de classification (20 itérations), puisque le classificateur bayésien est basé sur des calculs stochastiques.

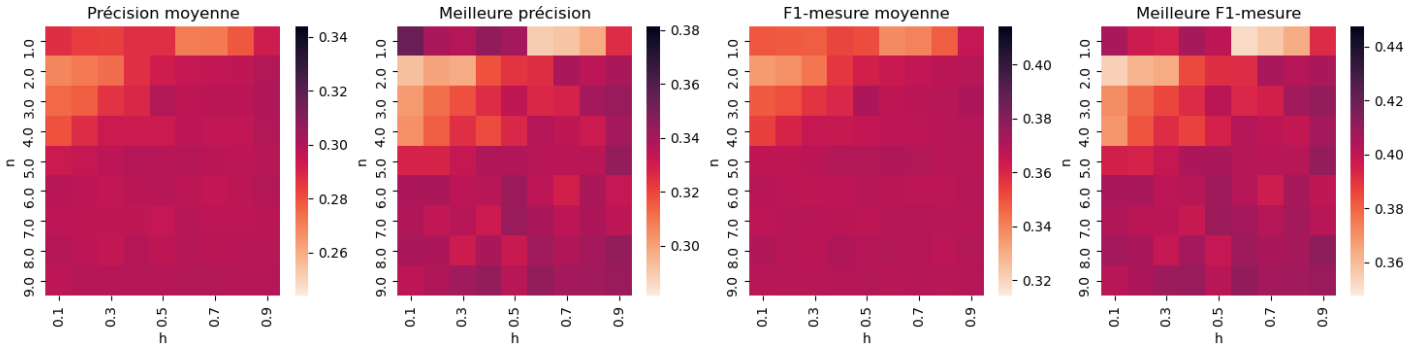
Comme mentionné précédemment, nous voulons observer l'impact des valeurs des hyper-paramètres n (la longueur de n-grammes) et h (le taux d'amélioration du coefficient de *Dice* requis pour continuer le processus de division) de notre méthode sur ces résultats. Par conséquent, nous générons autant de lexiques que de couples d'hyper-paramètres candidats. Après analyse de la sensibilité de notre méthode à ces paramètres, nous sélectionnons les configurations de notre méthode les plus intéressantes et les comparons aux méthodes concurrentes. Pour sélectionner les configurations de notre méthode, nous nous intéressons à plusieurs critères :

— La moyenne des F1-mesures sur les différentes valeurs f_{min} ;
— La meilleure F1-mesure sur les différentes valeurs de f_{min} ;
— La moyenne des précisions sur les différentes valeurs f_{min} ;
— La meilleure précision sur les différentes valeurs de f_{min}.

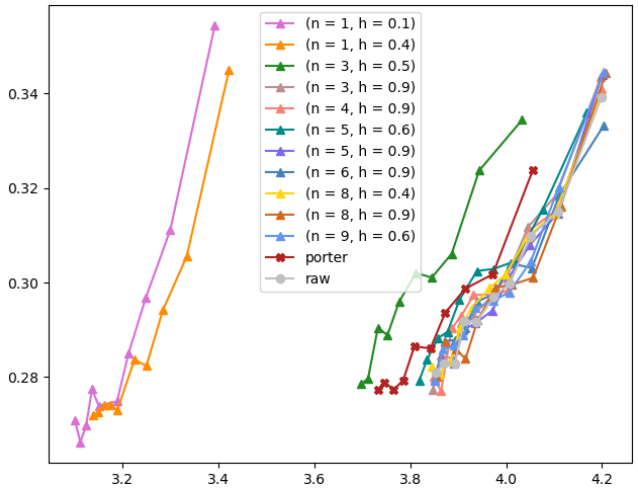
Pour les versions sélectionnées, pour la méthode de Porter ainsi que pour la version sans raciniseur, nous étudions précisément l'évolution des deux mesures en fonction de f_{min}.

Pour les besoins de notre algorithme, nous fixons par observation expérimentale les deux valeurs de lim et q, utilisées pour moduler respectivement l'arrêt de parcours des n-grammes et la taille idéale des groupes $\mathcal{N}_g$, afin d'exclure respectivement les n-grammes ayant un score trop faible, et les n-grammes représentant des affixes courants. Nous prenons lim de sorte de parcourir 75 % des n-grammes à chaque itération, et $q = 99$ (la taille idéale de $|W_g|$ est celle du quantile 99% des différents $|W_g|, g \in \mathcal{N}_W$).

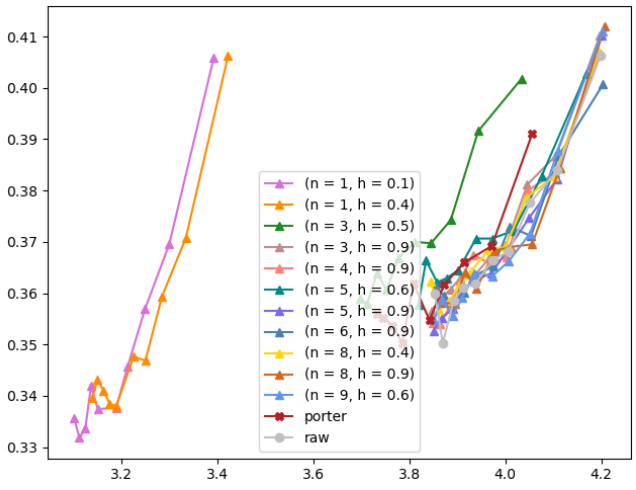
Nous exécutons 20 fois chaque configuration de notre algorithme pour chaque f_{min} étudié, et faisons varier f_{min} de 1 (tous les termes sont conservés) à 10 (seuls les termes apparaissant au moins 10 fois sont conservés). Chaque configuration de RFreeStem correspond à une valeur du couple de paramètres n et h, et contient donc 200 résultats. Il en va de même pour l'évaluation de Porter, et celle des données brutes. Nous proposons de faire varier les hyper-paramètres de notre méthode de la manière suivante : (i) h est compris entre 0 et 1 avec un pas de 0.1, (ii) n est un entier naturel variant de 1 à 9. Nous obtenons donc 81 configurations différentes. Nous souhaitons afficher l'évolution des métriques étudiées en fonction de f_{min}. Comme il semble peu réaliste d'afficher les 81 courbes, nous procédons d'abord à une sélection de nos méthodes. Pour ce faire, nous étudions les 4 indicateurs précédemment évoqués, les présentons sous forme de carte de chaleur, et sélectionnons, pour chacun, les 4 meilleures configurations. Nous obtenons donc, au plus, 16 configurations, que nous comparons aux résultats du raciniseur de Porter et à ceux de la version sans raciniseur.



(a) Cartes de chaleur des 4 indicateurs pour les 81 configurations de RFreeStem



(b) Précision



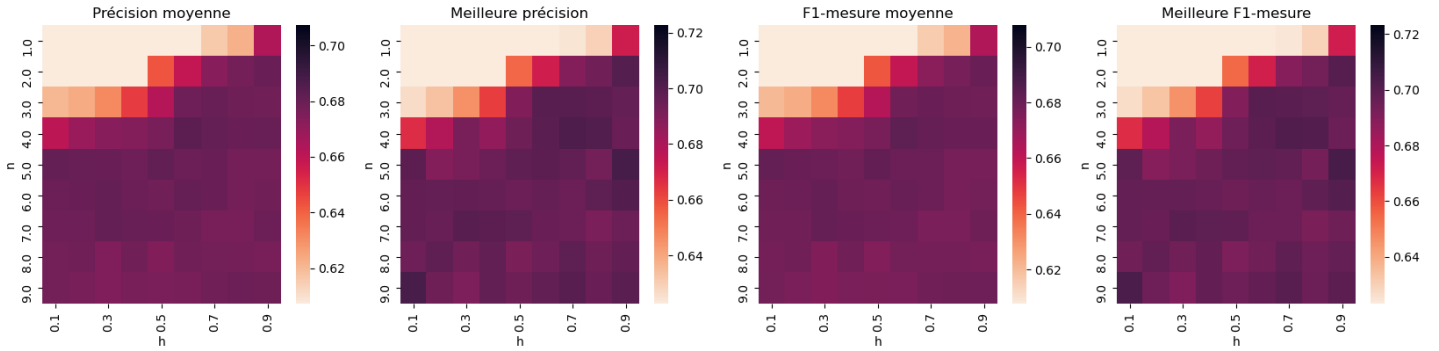
(c) F1-mesure

Figure 3.: Etude de la précision et F1-mesure pour sur le classificateur multinomial (**Tâche 1**) en anglais (AustLII) avec (a) les cartes de chaleur permettant de sélectionner les meilleures configurations pour RFreeStem (b) évolution de la précision et (c) évolution de la F1-mesure pour différents nombres de termes (liée aux valeurs de f_{min}) pour la version de Porter, nos versions sélectionnées, et la version sans racinisation.

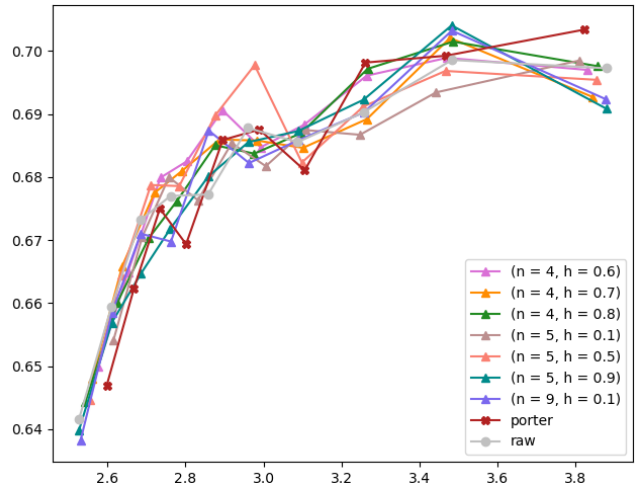
4.2. *Résultats et discussions*

Comparaison des deux tâches pour la langue anglaise

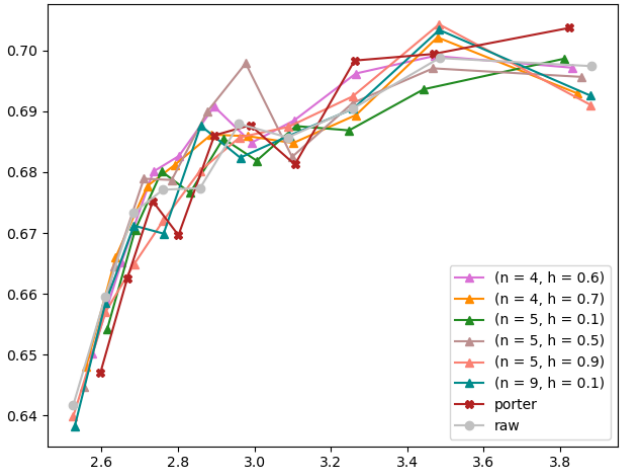
Nous comparons d'abord les résultats obtenus pour les deux tâches **Tâche 1** et **Tâche 2** pour l'anglais. Les données de la **Tâche 1** présentent de nombreuses classes, dont certaines ne contiennent que très peu d'exemples. Il est donc particulièrement difficile d'entraîner le classifieur à reconnaître ces classes, ce qui explique les résultats faibles observés en Figure **3**. Les cartes de chaleur (Figure **3a**) montrent un comportement lissé de notre raciniseur vis-à-vis de l'évolution de ces paramètres : la carte est globalement homogène et les transitions de couleurs sont douces. Ceci signifie que notre raciniseur est peu sensible à l'évolution de ces paramètres, et donc peu dépendant de leur optimisation. Les transitions douces de couleurs définissent des zones intéressantes, avec de bonnes précisions et F1-mesures pour des zones communes. Pour cette raison, les configurations sélectionnées pour leurs bonnes précisions présentent également de bonnes F1-mesures, et inversement. Nous observons notamment sur les Figures **3b** et **3c** que les différentes versions présentent des profils similaires entre leur précision et F1-mesure. La plupart de nos versions améliorent la version sans raciniseur, alors que la méthode de Porter présente des résultats décevants, atteignant seulement une F1-mesure de 0.391 et une précision de 0.324. Nous notons deux configurations particulièrement intéressantes, *(n=1, h=0.1)* ainsi que *(n=1, h=0.4)*, qui proposent



(a) Cartes de chaleur des 4 indicateurs pour les 81 configurations de RFreeStem



(b) Précision

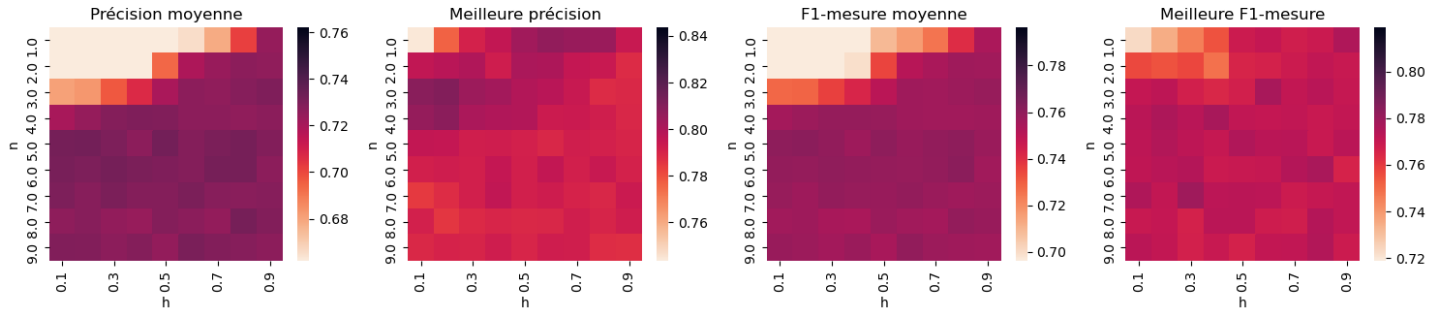


(c) F1-mesure

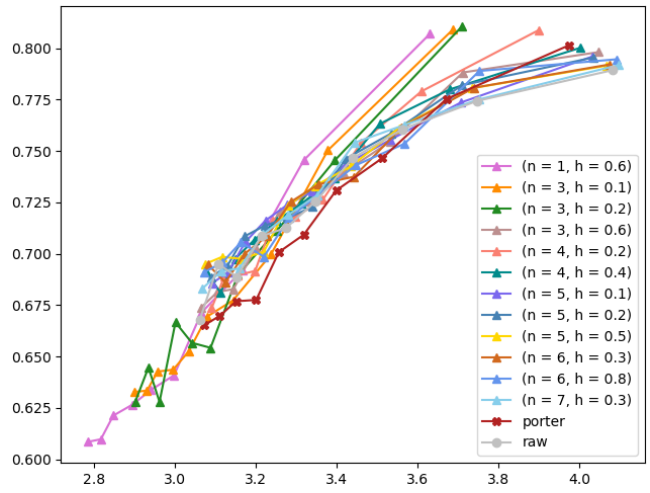
Figure 4.: Etude de la précision et F1-mesure pour sur le classificateur binaire (**Tâche 2**) en anglais (revues de films) avec (a) les cartes de chaleur permettant de sélectionner les meilleures configurations pour RFreeStem (b) évolution de la précision et (c) évolution de la F1-mesure pour différents nombres de termes (liée aux valeurs de f_{min}) pour la version de Porter, nos versions sélectionnées, et la version sans racinisation.

une racinisation forte et sont capables d'atteindre les meilleures précisions (jusqu'à 0.354) et de bonnes F1-mesures (jusqu'à 0.412) en divisant par 6 le nombre de termes nécessaires (environ 2 500 termes au lieu de 16 000). Au-delà de la sélection liée au filtre de f_{min}, les raciniseurs les plus forts ont tendance à réduire le vocabulaire de la langue, ce qui s'observe graphiquement par la limite à droite des courbes selon l'axe du nombre de termes.

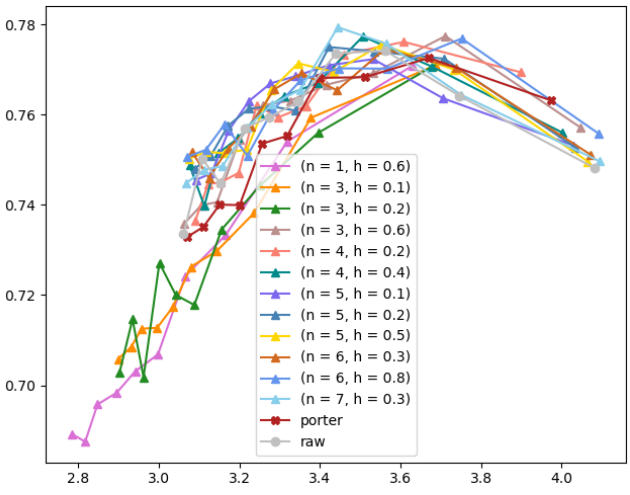
La **Tâche 2**, illustrée ici par le jeu de données des revues de film semble quant à elle beaucoup plus aisée. Il n'y a que deux classes, qui sont représentées de manière équitable. Ainsi, les résultats présentés dans la Figure **4** sont bien plus élevés que ceux de l'étude précédente. En outre, on remarquera le comportement quasi-identique de la précision et de la F1-mesure (lié à l'équilibre parfait des classes). Les cartes de chaleur (Figure **4a**) mettent en évidence des zones démarquées : la zone supérieure gauche correspond à des raciniseurs forts, qui regroupent de nombreux mots avec peu de racines. Il semble que ces solutions ne soient pas avantageuses pour ce jeu de données, avec des F1-mesures et précisions moyennes n'excédant pas 0.42. Nous l'expliquons par l'aspect informel des commentaires : l'anglais est généralement peu flexionnel, et en particulier ses termes modernes et informel. On aura ainsi plutôt tendance à dire *not happy* que *unhappy*. A l'inverse, le corpus AustLII contient des citations légales, dans un discours plus formel, avec des mots d'éthymologie latine, et donc plus flexionnel. Nous concluons que les corpus informels ont besoin de plus de mots pour incorporer le sens initial des documents, et ont



(a) Cartes de chaleur des 4 indicateurs pour les 81 configurations de RFreeStem



(b) Précision



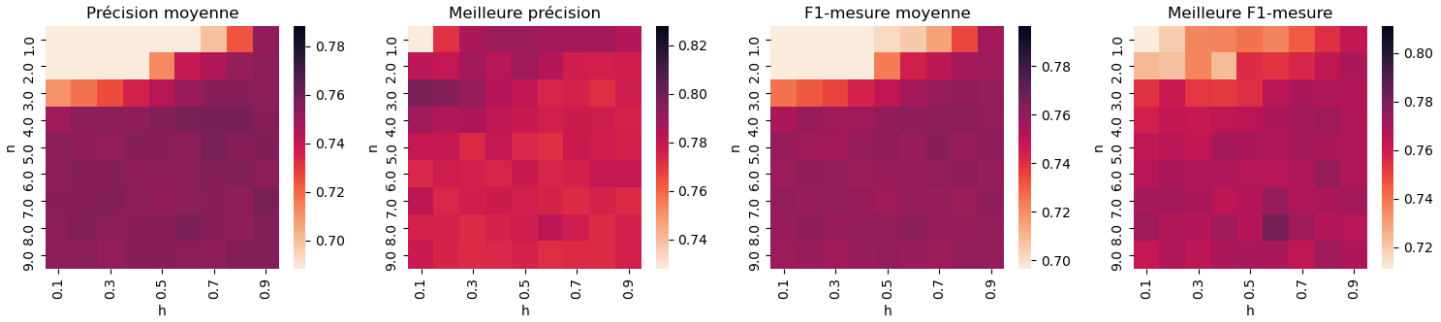
(c) F1-mesure

Figure 5.: Etude de la précision et F1-mesure pour sur le classificateur binaire (**Tâche 2**) en français (commentaires de produits Amazon) avec (a) les cartes de chaleur pour sélection des meilleures configurations de RFreeStem (b) évolution de la précision et (c) évolution de la F1-mesure pour différents nombres de termes (liée aux valeurs de f_{min}) pour less versions sélectionnées, celle de Porter, et celle sans racinisation.

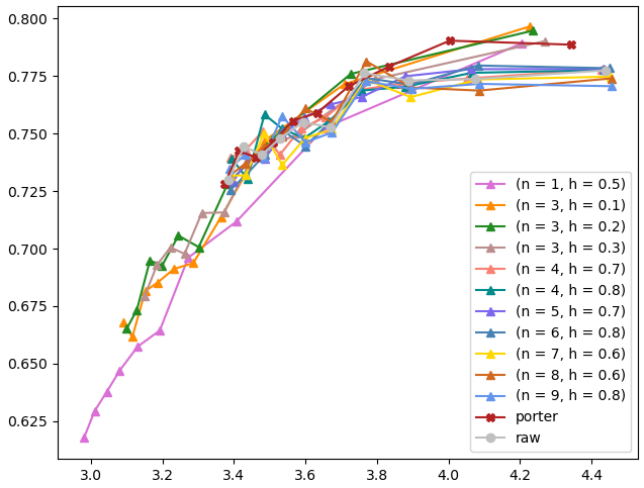
également moins de variantes flexionnelles susceptibles d'être supprimées lors de la racinisation.

Ces observations sont confirmées par la faible diminution du nombre de termes utilisés pour la plupart des méthodes, par rapport à la version originale, sans racinisation : dans les Figures **4b** et **4c**, il apparaît que la version proposant la racinisation la plus forte réduit le nombre de termes à environ 6 700 mots, pour un nombre de termes initial à 770 mots. Toutefois, la plupart de nos méthodes améliorent la précision et la F1-mesure de la version originale, et certaines atteignent des résultats équivalents à celui de Porter (environ 0.70), nécessitant toutefois moins de termes pour atteindre ces scores (6 000 termes pour Porter, 3 000 termes pour RFreeStem).

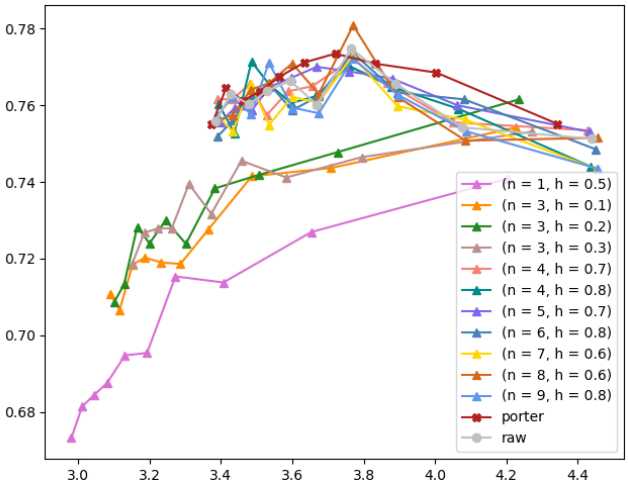
Dans cette première étude, nous observons donc que la **Tâche 1** de classification multinomiale est difficile, et que l'algorithme de Porter dégrade les résultats obtenus avec la version sans racinisation. Les données étant plutôt flexionnelles, puisqu'issues d'un registre soutenu, RFreeStem parvient à améliorer significativement les résultats initiaux, en proposant notamment deux versions qui réduisent considérablement la dimension des données. La **Tâche 2** est quant à elle plus facile, et les données d'origine présentent déjà des résultats satisfaisants. Les racinisations légères, comme celle de Porter et certaines de nos versions, sont les plus adaptées car elles permettent de conserver de nombreux termes intacts, et ainsi de maintenir le sens des commentaires. La racinisation a donc moins d'impact sur ce deuxième exemple. Nous pensons qu'au-delà de la tâche, ceci s'explique par le choix de la langue. Nous nous attendons à ce que la racinisation ait plus d'effet sur les langues flexionnelles, et le montrons dans l'étude



(a) Cartes de chaleur des 4 indicateurs pour les 81 configurations de RFreeStem



(b) Précision



(c) F1-mesure

Figure 6.: Etude de la précision et F1-mesure pour sur le classificateur binaire (**Tâche 2**) en allemand (commentaires de produits Amazon) avec (a) les cartes de chaleur pour sélection des meilleures configurations de RFreeStem (b) et (c) (b) évolution de la précision et (c) évolution de la F1-mesure pour différents nombres de termes (liée aux valeurs de f_{min}) pour nos versions sélectionnées, celle de Porter, et celle sans racinisation.

suivante.

Comparaison entre différentes langues sur la **Tâche 2**

Nous étudions maintenant les commentaires de produits Amazon en français et allemand. Ces deux langues sont flexionnelles : chaque racine possède de nombreuses variantes et ainsi de nombreux mots peuvent être groupés ensemble tout en respectant leur signification. Cependant, comme pour les revues de films, ces corpus regroupent des discours informels humains (ce qui est commun en analyse de sentiments). Dans un discours informel, le français, comme l'anglais, a tendance à être moins flexible : nous préférons dire *Je ne suis pas satisfait* que *Je suis insatisfait*. Les résultats présentés dans la Figure **5** sont toutefois encourageants. Les cartes de chaleur (Figure **5a**) sont encore une fois lisses avec des zones distinguées d'intérêt. Comme pour les revues de films, la zone supérieure gauche semble être à éviter au regard des indicateurs de moyenne, qui indiquent des valeurs inférieures à 0.70. Les indicateurs de meilleures précision et F1-mesure sont toutefois moins formels à ce sujet ; notamment, on observe dans cette zone des meilleures précisions avoisinant 0.80. Il semblerait donc que ces méthodes présentent de bons résultats pour certaines valeurs de f_{min}, mais des résultats globalement hétérogènes, qui font chuter la moyenne.

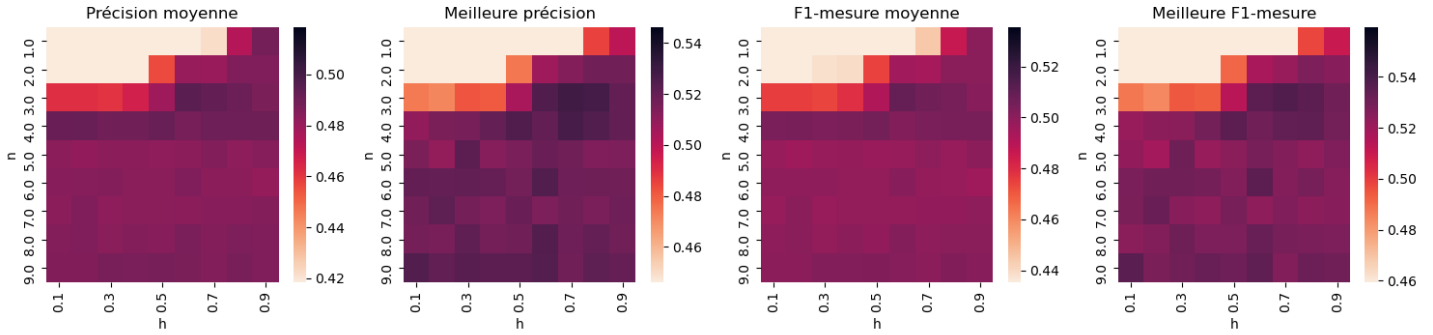
Cette observation est confirmée par l'étude des courbes de précision (figure **5b**) et de F1-mesure (fi-

gure **5c**), où des courbes de cette zone, *(n=1, h=0.6)*, *(n=3, h=0.1)* et *(n=3, h=0.2)*, présentent des valeurs élevées (jusqu'à 0.811 de précision), mais aussi des valeurs faibles (précisions inférieures à 0.63), liées à la taille très réduite du vocabulaire pour des valeurs de f_{min} élevées (à gauche de l'axe du nombre de terme, seulement 630 mots pour la première version de *(n=1, h=0.6)*). Ces trois méthodes réussissent toutefois à proposer des valeurs bien plus élevées que le raciniseur de Porter en terme de précision (0.811 contre 0.802 pour Porter), et des résultats satisfaisants pour la F1-mesure (environ 0.77 pour ces trois versions et Porter). On remarque de manière générale que toutes nos configurations ainsi que la version de Porter améliorent les résultats par rapport aux données brutes, qui recensent, au mieux, une précision de 0.789. Ceci confirme l'intérêt de la racinisation pour une langue flexionnelle telle que le français. En l'occurrence, notre raciniseur propose des solutions plus précises et plus efficaces (en terme de réduction de dimension) que la méthode de Porter.

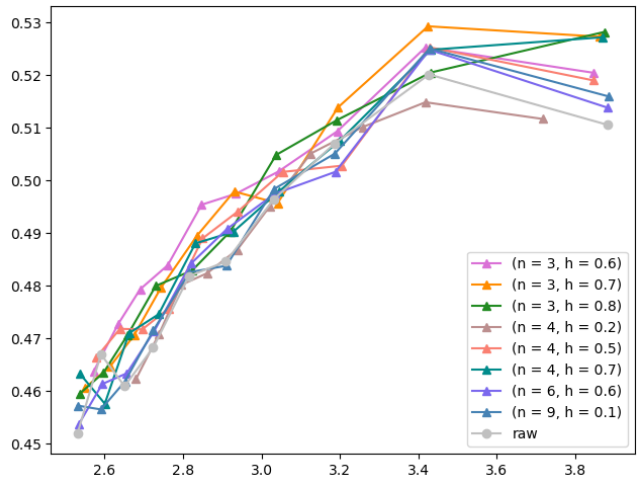
Des résultats similaires sont observés avec les commentaires allemands, notamment pour la description des cartes de chaleur en Figure **6a**. De nombreuses versions améliorent la précision de la classification comparée aux données brutes, et 2 de nos méthodes, avec des racinisations fortes, égalent les performances de Porter (Figure **6b**). En ce qui concerne la F1-mesure, les données originales présentent déjà des résultats satisfaisants, avec un meilleur score à 0.775 ; la méthode de Porter atteint des performances similaires (0.773). Seule une version de RFreeStem, correspondant à une racinisation légère, améliore les résultats initiaux, atteignant 0.781 de F1-mesure. Nous estimons d'une manière générale que RFreeStem est plus adapté à la langue allemande que l'algorithme de Porter. En effet, le processus de racinisation de Porter n'inclut aucun traitement pour les mots composés complexes, alors que [BR04] soutient que, pour la langue allemande, la tâche de décomposition de ces parties complexes est critique, plus encore que la découverte du radical. En opposition, notre algorithme est théoriquement capable de récupérer de telles structures linguistiques complexes, ce qui peut expliquer ces résultats encourageants.

Enfin, nous proposons d'étudier le comportement de notre méthode sur une langue pour laquelle peu de ressources de traitement automatique sont disponibles, comme l'ourdou. Pour cette langue, aucune version de Porter n'a été implémentée à notre connaissance. Nous comparons donc seulement nos résultats à ceux obtenus avec les données brutes dans la Figure **7**. La répartition équitable des données entre les différentes classes a pour conséquence un comportement similaire entre la précision et la F1-mesure. Les configurations de racinisation forte (zone supérieure gauche) semblent générer de mauvais résultats (inférieurs à 0.43), alors qu'une zone plus intéressante se dessine au centre à gauche. Ces méthodes, détaillées dans les Figures **7b** et **7c**, montrent une amélioration par rapport aux données initiales : on observe une meilleure F1-mesure à 0.538 et une meilleure précision à 0.529 contre respectivement 0.531 et 0.520 pour la version sans racinisation. Notre méthode améliore la tâche d'analyse de sentiments pour cette langue peu dotée.

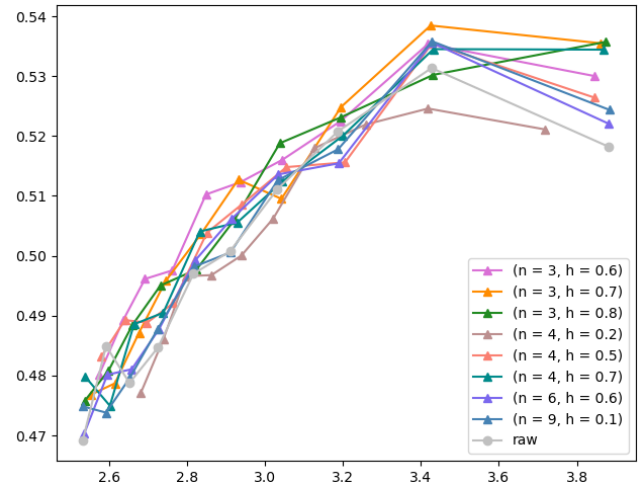
Pour conclure, le tableau **4.2** résume nos expérimentations. Nous y sélectionnons, pour chaque méthode, les meilleures versions, l'une par rapport à la précision, l'autre par rapport à la F1-mesure (mais il arrive qu'il s'agisse de la même), parmi les configurations des paramètres n, h (pour RFreeStem seulement) et f_{min}. Nous recensons la précision, la F1-mesure, mais aussi l'ICF, qui s'exprime, selon la formule précédemment fournie, au moyen de la colonne #termes initial — le nombre de mots dans le corpus d'origine — et la colonne #racines — le nombre de groupes et donc de racines uniques. L'ICF rend compte de la capacité du raciniseur à compresser la taille du vocabulaire, et dans le cadre de notre étude, la dimension d'entrée des données pour l'algorithme de classification. La colonne #termes restants définit le nombre de termes après pré-traitement (suppression des mots vides, et sélection des mots



(a) Cartes de chaleur des 4 indicateurs pour les 81 configurations de RFreeStem



(b) Précision



(c) F1-mesure

Figure 7.: Etude de la précision et F1-mesure pour sur le classificateur binaire (**Tâche 2**) en ourdou (Tweets) avec (a) les cartes de chaleur permettant de sélectionner les meilleures configurations pour RFreeStem (b) évolution de la précision et (c) évolution de la F1-mesure pour différents nombres de termes (liée aux valeurs de f_{min}) pour nos versions sélectionnées et la version sans racinisation.

présents au moins f_{min} fois).

Nous observons d'abord que les meilleurs résultats sont systématiquement obtenus par des versions de RFreeStem pour la précision et la F1-mesure. On constate par ailleurs que certaines de nos propositions présentent des résultats très satisfaisants sur la métrique pour laquelle elles n'ont pas été sélectionnées. Par exemple, pour les données d'Amazon en allemand, la première version de notre méthode est sélectionnée pour son excellente précision, mais sa F1-mesure est également tout à fait satisfaisante, et comparable aux meilleurs résultats. Notons par ailleurs la corrélation précédemment évoquée entre les valeurs des paramètres de RFreeStem et la force de la racinisation, exprimée par l'ICF : de faibles valeurs de n et h correspondent à des racinisations fortes, qui génèrent peu de groupes et sont donc capables de réduire fortement la taille du vocabulaire, entraînant des valeurs élevées pour l'ICF. On constate également que pour les jeux de données où deux de nos versions sont proposées, ces deux configurations sont radicalement opposées : celle provenant de la précision est souvent une racinisation très forte, et celle provenant de l'étude de la F1-mesure est très légère. Nous concluons que notre raciniseur est très modulable, et peut proposer une grande diversité de type de racinisation.

En analysant les lexiques proposés par les meilleures méthodes du tableau **4.2**, nous repérons deux difficultés notables et antagonistes, liées à l'optimisation des hyper-paramètres. Parmi les meilleures configurations retenues, celles qui présentent de faibles valeurs de h et n sont peu divisives et ont tendance à

Données	#termes initial	Méthode	Paramètres n	h	f_{min}	#racines	#termes restants	Précision	F1	ICF
AustLII	16609	RFreeStem	1	0.1	1	2748	2468	**0.354**	0.406	**0.835**
			8	0.9	1	16553	16020	0.344	**0.412**	0.003
(Anglais)		Porter			1	12000	11374	0.324	0.391	0.278
Tâche 1		Aucune			1	16609	15795	0.339	0.406	0
Revue films	7736	RFreeStem	5	0.9	2	7704	3042	**0.704**	**0.704**	0.004
(Anglais)		Porter			1	6757	6666	0.703	0.703	**0.127**
Tâche 2		Aucune			2	7736	3050	0.699	0.699	0
Amazon	13656	RFreeStem	3	0.2	1	5984	5120	**0.811**	0.772	**0.562**
			7	0.3	3	13410	3679	0.763	**0.776**	0.018
		Porter			1	10718	9432	0.802	0.763	0.215
					2		4687	0.775	0.773	
(Français)		Aucune			1	13656	12068	0.789	0.748	0
Tâche 2					3		3653	0.761	0.774	
Amazon	30986	RFreeStem	3	0.1	1	18757	16907	**0.797**	0.754	**0.395**
			8	0.6	4	30841	5978	0.781	**0.781**	0.005
		Porter			2	24624	10335	0.790	0.769	0.205
					4			0.771	0.773	
(Allemand)		Aucune			1	30986	27843	0.778	0.751	0
Tâche 2					4	30986	5879	0.776	0.775	
Tweets	7886	RFreeStem	3	0.7	2	7856	2681	**0.529**	**0.538**	**0.004**
(Ourdou)										
Tâche 2		Aucune			2	7886	2686	0.520	0.531	0

Tableau 4.2.: Résultats expérimentaux des meilleures configurations de RFreeStem, de Porter et de la méthode sans racinisation (meilleure précision et meilleure F1-mesure). La colonne #termes initial désigne le nombre de mots dans le corpus, la colonne #racines désigne le nombre de racines, ou de groupes, créés par la méthode et la colonne #termes restants désigne le nombre de termes retenus pour la classification (après pré-traitement, et sélection des mots présents au moins f_{min} fois).

regrouper de nombreux mots ensemble. Pour le français par exemple, la configuration $(n = 3,\ h = 0.2)$ regroupe tous les mots contenant le 3-gram *ver*, à savoir, des mots aussi variés que *traverser*, *divers* ou *univers*. Si ces mots ont des étymologies différentes, ils n'ont parfois que peu d'importances dans la tâche d'analyse de sentiments. En revanche, des antonymes, par exemple *réaliste* et *irréaliste* sont regroupés, ce qui peut masquer la polarisation des sentiments décrits. Les scores pourtant élevés de ces configurations, qui génèrent une racinisation forte, s'expliquent par l'importante réduction de la dimension. En effet, la tâche de classification bénéficie grandement de cette réduction (5 984 mots contre 13 656), qui évite la situation de la malédiction de la grande dimension [AHK01].

L'importance de la réduction de la dimension peut se déduire des résultats des méthodes peu divisives. En français, la configuration $(n = 7,\ h = 0.3)$ construit une racinisation légère, contenant peu d'erreurs de sur-racinisation. Elle génère donc de nombreux groupes (13 410 racines), laissant le jeu de données dans une situation de grande dimension. On remarquera donc que pour cette configuration, les meilleurs résultats sont obtenus avec $f_{min} = 3$, une paramétrisation qui fait chuter le nombre de termes à 3 679.

Or, la suppression des termes est peu flexible : avec $f_{min} = 2$, nous conservons 5 630 racines parmis les 13 410 racines ; la configuration $f_{min} = 3$ amène directement à 3 679 termes. La faible flexibilité de cette paramétrisation a posteriori empêche une sélection efficace des termes. Ainsi, l'optimisation des paramètres de RFreeStem consiste à trouver un équilibre entre une racinisation forte qui réduit efficacement la dimension mais crée des groupes parfois peu pertinents, et une racinisation faible qui fait peu d'erreurs de sur-racinisation, mais ne permet pas une représentation efficace des données.

5. Conclusion

La racinisation est généralement utilisée comme une étape de prétraitement pour différentes tâches de fouille de texte. Étant donné que la sensibilité de la racinisation dépend fortement de la langue des textes analysés, nous avons proposé un raciniseur sans règle, basée sur des données de corpus. Il peut donc être appliqué quelle que soit la langue. Nous avons montré que notre algorithme égale ou surpasse le célèbre algorithme de Porter dans différentes tâches, et que ses performances sont encore plus importantes sur les langages flexionnels.

Une amélioration de notre méthode pourrait consister à fournir une méthode automatique de détermination des valeurs des hyper-paramètres n et h. Cela éviterait la tâche d'optimisation de paramètres. Une autre extension de nos travaux pourrait être de proposer un cadre d'évaluation plus large en (i) analysant les performances de RFreeStem sur les langues agglutinatives (ii) en évaluant l'amélioration de la précision du raciniseur sur des tâches de recherche d'information, voire sur des tâche de RI croisée ou multi-lingue, (iii) en confrontant notre raciniseur à d'autres méthodes basées sur des corpus, comme [SO15]. Nous laissons ces questions intéressantes ouvertes pour une contribution future. Enfin, la méthodologie proposée dans ce papier peut être généralisée à de nombreux cas de structuration de données textuelles, comme l'inférence de structure pour des données semi-structurées [CMTB20, NAM97], ou la compression de texte [Wit04].

Bibliographie

George W Adamson and Jillian Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information storage and retrieval*, 10(7-8) :253–260, 1974.

Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv :1607.06450*, 2016.

Martin Braschler and Bärbel Ripplinger. How effective is stemming and decompounding for german text retrieval ? *Information Retrieval*, 7(3-4) :291–316, 2004.

Oihana Coustié, Josiane Mothe, Olivier Teste, and Xavier Baril. Meting : A robust log parser based on frequent n-gram mining. *IEEE The International Conference on Web Services (ICWS)*, 2020.

John Dawson. Suffix removal and word conflation. *ALLC bulletin*, 2(3) :33–46, 1974.

William Bruce Frakes and Ricardo Baeza-Yates. *Information retrieval : Data structures & algorithms*, volume 331. Prentice Hall Englewood Cliffs, NJ, 1992.

William B Frakes and Christopher J Fox. Strength and similarity of affix removal stemming algorithms. In *ACM SIGIR Forum*, volume 37, pages 26–30. ACM New York, NY, USA, 2003.

Filippo Galgani, Paul Compton, and Achim Hoffmann. Knowledge acquisition for categorization of legal case reports. In *Pacific Rim Knowledge Acquisition Workshop*, pages 118–132. Springer, 2012.

Donna Harman. How effective is suffixing ? *Journal of the american society for information science*, 42(1) :7–15, 1991.

David A Hull. Stemming algorithms : A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1) :70–84, 1996.

Margaret A Hafer and Stephen F Weiss. Word segmentation by letter successor varieties. *Information storage and retrieval*, 10(11-12) :371–385, 1974.

Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6) :1930–1938, 2011.

Grzegorz Kondrak. N-gram similarity and distance. In *International symposium on string processing and information retrieval*, pages 115–126. Springer, 2005.

Wessel Kraaij and Renée Pohlmann. Porter's stemming algorithm for dutch. *Informatiewetenschap*, pages 167–180, 1994.

Wessel Kraaij and Renée Pohlmann. Viewing stemming as recall enhancement. In *SIGIR*, volume 96, pages 40–48, 1996.

Robert Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–202. ACM, 1993.

Julie Beth Lovins. Development of a stemming algorithm. *Mech. Translat. & Comp. Linguistics*, 11(1-2) :22–31, 1968.

Ludovic Lebart, André Salem, and Lisette Berry. *Exploring textual data*, volume 4. Springer Science & Business Media, 1997.

Cristian Moral, Angélica de Antonio, Ricardo Imbert, and Jaime Ramírez. A survey of stemming algorithms in information retrieval. *Information Research : An International Electronic Journal*, 19(1) :n1, 2014.

Prasenjit Majumder, Mandar Mitra, Swapan K Parui, Gobinda Kole, Pabitra Mitra, and Kalyankumar Datta. Yass : Yet another suffix stripper. *ACM transactions on information systems (TOIS)*, 25(4) :18, 2007.

Samuel Marcos-Pablos and Francisco J García-Peñalvo. Information retrieval methodology for aiding scientific database search. *Soft Computing*, pages 1–10, 2019.

Svetlozer Nestorov, Serge Abiteboul, and Rajeev Motwani. Inferring structure in semistructured data. *ACM SIGMOD Record*, 26(4) :39–43, 1997.

Chris D. Paice. Another stemmer. *SIGIR Forum*, 24(3) :56–61, November 1990.

Chris D Paice. An evaluation method for stemming algorithms. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–50. Springer-Verlag New York, Inc., 1994.

Chris D Paice. Method for evaluation of stemming algorithms based on error counting. *Journal of the American Society for Information Science*, 47(8) :632–649, 1996.

Bo Pang and Lillian Lee. A sentimental education : Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3) :130–137, 1980.

Martin F Porter. Snowball : A language for stemming algorithms, 2001.

Jacques Savoy. Light stemming approaches for the french, portuguese, german and hungarian languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 1031–1035, 2006.

Radu Soricut and Franz Och. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 1627–1637, 2015.

Zareen Sharf and Saif Ur Rahman. Performing natural language processing on roman urdu datasets. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, 18(1) :141–148, 2018.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan : Directional self-attention network for rnn/cnn-free language understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Fadillah Z Tala. A study of stemming effects on information retrieval in bahasa indonesia. *Institute for Logic, Language and Computation, Universiteit van Amsterdam, The Netherlands*, 2003.

Peter Willett. The porter stemming algorithm : then and now. *Program*, 40(3) :219–223, 2006.

Ian H Witten. Adaptive text mining : inferring structure from sequences. *Journal of Discrete Algorithms*, 2(2) :137–159, 2004.

Tilahun Yeshambel, Josiane Mothe, and Yaregal Assabie. 2airtc : The amharic adhoc information retrieval test collection. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 55–66, 2020.

Tilahun Yeshambel, Josiane Mothe, and Yaregal Assabie. Amharic document representation for adhoc retrieval. In *Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2020, Volume 1 : KDIR*, pages 124–134, 2020.

