Inférence de réseaux génétiques à partir de la littérature scientifique de þÿ C h l a m y d o m o n a s r e i n h a r d t i i : c o n c e p t i o n d u n p a c k a g e

Auteur : Lété, Jonathan

Promoteur(s) : Meyer, Patrick

Faculté : Faculté des Sciences

Diplôme : Master en biochimie et biologie moléculaire et cellulaire, à finalité spécialisée en bioinformatique et modélisation

Année académique : 2016-2017

URI/URL : http://hdl.handle.net/2268.2/3243

Inférence de réseaux génétiques à partir de la littérature scientifique de Chlamydomonas reinhardtii : conception d'un package R

LETE Jonathan

En vue de l'obtention du grade de Master de Biochimie et Biologie Moléculaire et Cellulaire à finalité Bioinformatique et Modélisation

Année académique 2016-2017

Université de Liège
Faculté des Sciences
PhytoSYSTEMS
Laboratoire de Biologie des Systèmes et Bioinformatique (BioSys)

Promoteur : Prof. Patrick E. Meyer

Remerciements

Tout d'abord, je souhaiterais remercier le Prof. Patrick E. Meyer pour m'avoir permis d'approfondir, durant ce mémoire, le sujet de mon stage de l'année dernière portant sur la *fouille de textes* (**text mining**). Je le remercie aussi pour son accueil, ses conseils et l'encadrement qu'il m'a apporté, qui m'ont permis de progresser dans ce domaine et dans la bioinformatique de manière générale.

Je remercie aussi les Doctorants Ngoc Pham et Manuel Noll pour leur soutien durant les différentes phases du développement de ce mémoire et pour la bonne ambiance qu'ils apportaient au service. Plus particulièrement, je remercie Manuel Noll pour sa supervision ainsi que sa colaboration dans l'écriture de l'article *"Gene Entity Recognition of Full Text Articles"*, présenté en juin dernier lors de la sixième conférence internationale de bioinformatique et sciences biomédicales à Singapour.

Ensuite, je remercie mes parents et ma famille qui m'ont toujours soutenu au long de mon cursus et pour leur participation dans la correction orthographique de ce mémoire.

Mes remerciements vont aussi à mes collègues et ami(e)s, Amandine, Arnaud, Julie, Raphaël, et tous les autres qui m'ont permis de me surpasser durant mon cursus universitaire.

Finalement, j'aimerais remercier l'ensemble des professeurs du Master de Biochimie et Biologie Moléculaire et Cellulaire et plus particulièrement ceux de la finalité Bioinformatique et Modélisation pour m'avoir enseigné les connaissances qui m'ont permis d'en arriver là aujourd'hui.

Résumé du Mémoire

La fouille de textes (text mining) comme outil de création automatique de réseaux génétiques

Avec l'amélioration constante de la vitesse et de l'accessibilité des techniques de génomique, transcriptomique et protéomique, la quantité de données disponible pour ces secteurs de recherche ne cesse d'augmenter. Malheureusement, la majorité de ces informations est encore stockée sous forme de documents textes non structurés et ce malgré la présence de certaines bases de données spécialisées.

Toutefois, une technique existe pour interpréter automatiquement ces documents textes et en retirer les informations utiles : la "fouille de textes" (text mining). Bien que généralement utilisée dans d'autres secteurs que la biologie, par exemple pour extraire les mots revenant le plus souvent sur twitter, elle est toutefois capable d'isoler des informations bien plus ciblées telle que la relation entre deux gènes.

Cette technique consiste à appliquer des analyses statistiques sur l'ensemble des mots issus d'un groupe de textes qui aura préalablement subi différents traitements tels que la séparation en phrases, la suppression de la ponctuation, etc …

Le but était ici de développer un package R capable de rechercher automatiquement, sur base d'un ou plusieurs mot clé spécifiés par l'utilisateur (*Chlamydomonas reinhardtii*, …), les documents disponibles sur une base de données ("PubMed", …), de les analyser et d'en extraire un réseau génétique d'interactions entre les différents gènes issus de ces documents.

Au final, malgré l'identification de relations entre gènes d'espèces différentes à cause de sa capacité à isoler les gènes de plusieurs espèces, les prédictions générées par notre modèle peuvent être considérées comme satisfaisantes (score F1 de 68% pour l'identification des gènes dans le corpus CRAFT) compte tenu du fait qu'il n'utilise aucune ressource externe spécifique au type d'organisme analysé.
La version finale de la librairie R "GeneMining" construite tout au long de ce mémoire est disponible sur le site http://www.biosys.ulg.ac.be/students/Lete/GeneMining_0.4.0.tar.gz.

Enfin, les résultats intermédiaires obtenus au cours de ce mémoire ont fait l'objet de l'écriture d'un article pour la sixième conférence internationale de bioinformatique et sciences biomédicales à Singapour. L'article en question peut être consulté à l'adresse suivante : http://www.biosys.ulg.ac.be/students/Lete/ICBBS2017.pdf.

Table des matières

1 Introduction

Durant les dernières années, les avancées technologiques en génétique et en biologie moléculaire ont permis le sequençage du génome d'un grand nombre d'espèces, entraînant de ce fait la découverte de nombreux éléments régulateurs (protéines, lipides, …).[1]
Dans le but d'intégrer toutes ces données, un nouveau domaine d'étude a vu le jour : la biologie des systèmes.

1.1 Biologie des systèmes

Liant la biologie et l'informatique, la biologie des systèmes est une discipline visant à modéliser des systèmes biologiques complexes en utilisant des outils mathématiques et statistiques.
Elle étudie tout particulièrement les interactions entre différents éléments d'un système biologique (cellules, organites, enzymes, …) en vue d'extraire un modèle représentant le fonctionnement du système dans son intégralité. Il s'agit donc d'une approche holistique basée sur l'idée qu'un tout est défini par plus que la somme de ses composants et s'opposant donc à la pensée réductionnelle.

> *"Identifying all the genes and proteins in an organism is like listing all the parts of an airplane. While such a list provides a catalog of the individual components, by itself it is not sufficient to understand the complexity underlying the engineered object. We need to know how these parts are assembled to form the structure of the airplane."*[2]

Ces modèles permettront par la suite de prédire l'évolution des systèmes concernés sous différentes conditions et d'ainsi développer des solutions aux divers problèmes de santé et d'environnement qui inquiètent notre société.

1.2 Réseaux de régulation génétique

Cette application de la biologie des systèmes s'intéresse en particulier à la modélisation de réseaux de relations entre gènes et autres molécules régulatrices (Figure 1). Ces réseaux ainsi générés permettent de mesurer l'impact de la modfication d'un gène sur la viabilité d'une cellule ou sur son efficacité. Un grand nombre d'expériences est généralement requis pour identifier les interactions spécifiques necessaires à la création de ce type de modèle.

Dans un premier temps, ces expériences consistèrent à créer un profil d'expression de gènes en fonction d'une condition inhabituelle sur lequel une analyse par partitionement (clustering) était utilisée afin d'identifer les gènes co-exprimés avec d'autres gènes dont la fonction était connue.[3]

Bien que ce type d'analyse nous donne des informations sur la corrélation entre certains gènes et une fonction biologique, le lien de causalité reste inconnu. De nos jours, plusieurs méthodes ont depuis été proposées pour identifier ces relations régulatoires en utilisant le niveau d'abondance relative des ARN messagers et divers informations concernant certaines régulations post-transcriptionnelles et post-traductionnelles.[4] Ce type de construction de modèle basé sur des données expérimentales est aussi appelé "Rétro-ingénierie" ou "inférence de réseau".

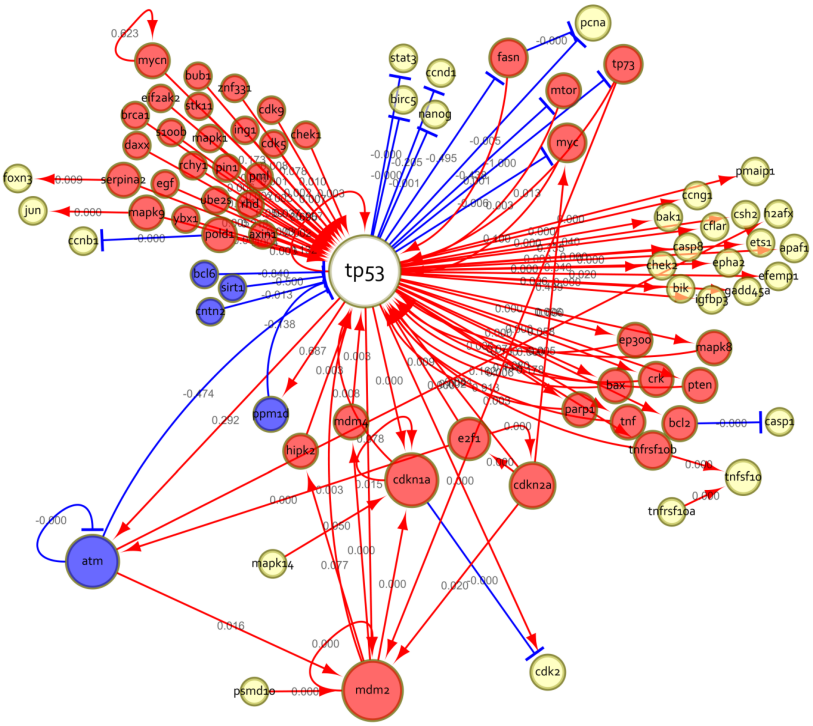


Figure 1: *Réseau de régulation génétique de la protéine p53, facteur de transcription régulant certaines fonctions cellulaires importantes comme la mitose ou la mort programmée chez l'homme. Les flèches de couleur rouge indiquent une induction/activation tandis que les bleues montrent une inhibition. La taille d'un noeud est proportionnelle au nombre de connections avec d'autres noeuds.[5]*

Ces méthodes d'inférence de réseau restent tout de même difficiles à mettre en place en raison de la nature combinatoire de la tâche (trouver la bonne combinaison de gènes régulateurs), du peu de données facilement accessibles et de l'imprécision de celles-ci.[4]
Il est, dès lors, bénéfique de coupler ces informations avec d'autres mesures de génomique, transcriptomique, protéomique et métabolomique ainsi qu'aux connaissances biologiques présentes dans la littérature en une unique procédure de modélisation.

1.3 Fouille de textes

Néanmoins, même si certaines des ces informations peuvent être facilement accessibles grâce à leur présence dans une base de donnée spécialisée, la majeure partie de ces données est contenue dans des publications scientifiques.
Par ailleurs, l'automatisation haut débit des différentes méthodes d'analyses biologiques a engendré une augmentation considérable du nombre de publications disponibles, rendant l'analyse manuelle de ces documents quasiment impossible.
De plus, ces informations sont sans cesse renouvelées et la quantité de documents disponibles s'accroît de jour en jour. Cette évolution s'observe notamment par l'augmentation considérable de la quantité d'articles disponibles sur MEDLINE, la base de données la plus importante pour les publications concernant les sciences de la vie et les sciences biomédicales. En 2016, celle-ci comptait près de 23,531,948 citations tandis qu'elle n'en disposait que de 14,103,589 en 2006 soit un accroissement moyen d'environ 940,000 articles par an.[6]

Des méthodes d'extraction automatique de données, appelées fouille de textes ou text mining en anglais, sont dès lors requises pour traiter cette énorme quantité de documents. Si les articles scientifiques sont écrits de façon à être agréable à lire et compréhensible par les initiés, ils le sont beaucoup moins par les ordinateurs à cause de leur manque de structure. Les informations essentielles sont généralement obtenues grâce à une hiérarchisation du texte initial permettant de définir les tendances et motifs d'entités spécifiques (date, localisation géographique, gène, …) via un apprentissage statistique (pattern recognition).

1.4 Stage 2016 : inférence de réseaux biologiques à partir de la littérature

Ce stage reposait sur l'utilisation de librairies "R" capables de réaliser de la fouille de texte afin d'isoler les gènes présents dans des documents texte issus d'une recherche sur la base de données PubMed. La reconnaissance des gènes se faisait ici via l'utilisation d'une base de données externe fournissant une liste des gènes d'un organisme donné contre laquelle chaque mot extrait était comparé.
En comparant les résultats obtenus pour *Escherichia coli* avec la base de données EcoliNet, il fut évident que cette méthode de recherche de gènes, en plus d'être uniquement limitée aux gènes fournis par l'utilisateur, ne permettait pas d'obtenir des résultats convaincants (18% de précision et 3% de rappel). D'autres méthodes, telle que la limitation de la taille de la fenêtre de recherche d'interactions, furent testées afin d'augmenter ces résultats sans toutefois obtenir un meilleur compromis.
Il était donc nécessaire d'investiguer d'autres méthodes capable d'identifier des noms de gènes dans un texte et ce quel que soit l'organisme concerné.

1.5 Reconnaissance d'entités nommées

La reconnaissance d'entités nommées (Named Entity Recognition, NER) est une sous-tâche de l'extraction de l'information (Information Extraction, IE) visant à classifier les mots issus d'un texte en différentes catégories pré-définies tels que des noms d'entreprises, de personnalités, de localisations ou même des dates.[7]
Ce genre de procédé comprend généralement une phase d'identification des entités, qui peuvent être composées de plusieurs éléments (tokens) contigus, suivie par une phase de classification dépendante des caractéristiques de ces entitées.
Ainsi, à partir d'une simple phrase non annotée[8] telle que :

```
"The University of Liège (ULg), in Liège, Wallonia, Belgium, is a major public
university in the French Community of Belgium."
```

Un algorithme de reconnaissance d'entités nommées, programmé dans l'optique d'identifier des noms d'entreprise et de localisation, générera l'annotation suivante :

```
"The <ORGANIZATION>University of Liège</ORGANIZATION> (ULg), in
<LOCATION>Liège</LOCATION>, <LOCATION>Wallonia</LOCATION>,
<LOCATION>Belgium</LOCATION>, is a major public university in the French
Community of <LOCATION>Belgium</LOCATION>."
```

Dans cet exemple, un nom d'entreprise à trois éléments et quatre noms de localisations formés d'un seul élément chacun ont été identifés et classifiés.

La plupart des ces algorithmes repose sur la création de règles spécifiques par des experts du domaine analysé et sur l'utilisation de techniques basées sur la grammaire linguistique pour former des modèles statistiques grâce à l'apprentissage automatique.

1.6 Apprentissage automatique

L'apprentissage automatique, plus souvent dénommé par son appellation anglaise "machine learning", a pour but l'étude des théories, des propriétés et des performances d'algorithmes capables d'apprendre et de faire des prédictions sur des données.
Il s'agit d'un champ d'étude interdisciplinaire basé sur l'utilisation de techniques issues de différents secteurs d'activités telles que l'intelligence artificielle, la théorie de l'optimisation, la théorie de l'information, les statistiques, les sciences cognitives et beaucoup d'autres disciplines scientifiques, mathématiques et d'ingénierie.[9]

Généralement, l'apprentissage automatique peut-être divisé en trois sous-catégories :

L'apprentissage supervisé, qui requiert l'entraînement d'un modèle sur base d'exemples dont la solution est fournie préalablement par un "professeur".

L'apprentissage non-supervisé, où les solutions ne sont pas mises à disposition de l'algorithme qui doit dès lors lui même grouper les exemples, sur base de leurs ressemblances, en un nombre défini de catégories.

L'apprentissage par renforcement, durant lequel l'environnement d'exemples est dynamique et l'algorithme est constament félicité ou puni en fonction de la justesse des prédictions générées.

L'objectif commun de ces différents apprentissages est de créer un modèle capable de généraliser un problème grâce à l'expérience acquise à partir des données fournies. On entend par "généralisation" la capacité du modèle à prédire de façon précise et fidèle la classe de nouvelles données.

De manière générale, les données d'entrainement suivent une distribution de probabilités inconnue mais toutefois représentative de l'espace des solutions. Ainsi, l'objectif de l'algorithme est de construire un modèle général suivant cette distribution permettant d'obtenir des prédictions suffisamment précises pour des données inconnues.
De ce fait, le programme ne doit pas uniquement se contenter de trouver un modèle capable de classifier les données d'entrée mais trouver la séparation optimale permettant de minimiser l'erreur des prédictions sur des données jamais rencontrées jusqu'alors (Figure 2).

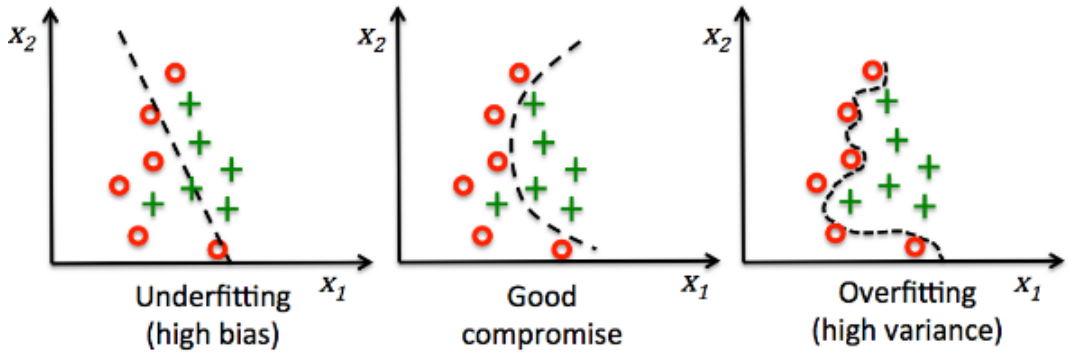


Figure 2: *Représentation graphique du compromis biais-variance. Sur l'image de gauche, le modèle généralise trop les informations des données initiales résultant en une faible variance accompagnée d'un biais important. À l'inverse, l'image de droite représente un modèle suivant le plus fidèlement possible les données d'entrainement produisant un faible biais mais une grande variance. L'image au centre correspond au modèle optimal, minimisant à la fois le biais et la variance.[10]*

Comme le jeu de données fourni au modèle est limité et que le futur est incertain, les solutions générées par ces modèles ne sont pas garanties et sont accompagnées le plus souvent d'une probabilité représentant la certitude de ces prédictions.

Grâce à leur grande flexibilité, l'utilisation des algorithmes d'apprentissage automatique s'est désormais étendu à une large gamme de domaines scientifiques afin de résoudre une variété de problèmes tels que la prédiction de catastrophes météorologiques majeures ou la détection d'accès frauduleux à la base de données de soin de santé d'un hopital.[11]

1.7 Traitement automatique du langage naturel

La sous-catégorie de l'apprentissage automatique visant à traiter un grand nombre de données issu du langage humain (naturel) est le Natural Language Processing (NLP), soit traitement automatique du langage naturel en français. On définit comme langage naturel tout langage ayant évolué au fil du temps et des répétitions sans préméditation. En cela, il s'oppose aux langages formels tels que le binaire ou les différents langages de programmation.

Le but principal de cette discipline est la compréhension du langage humain, laquelle se heurte toutefois à trois problèmes majeurs : le processus de la réflexion, le savoir extra-linguistique ainsi que la représentation et la compréhension de l'environnement linguistique.

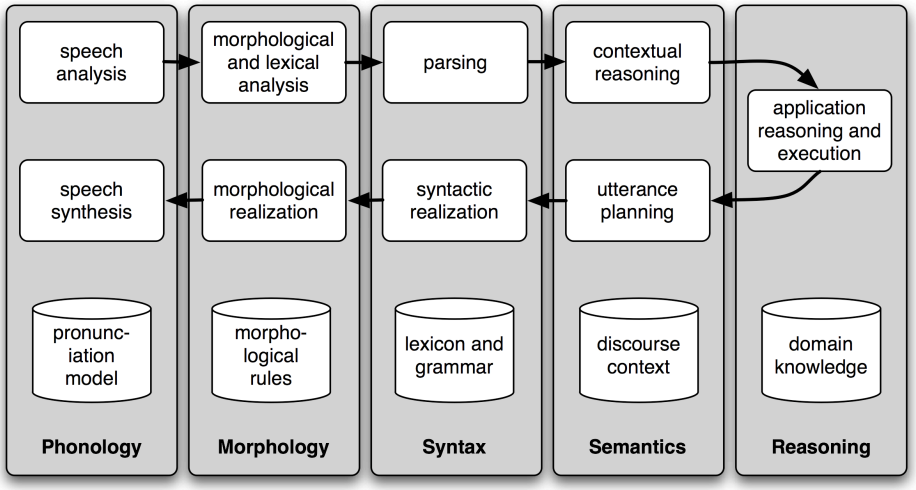


Figure 3: *Représentation des différents niveaux d'analyse utilisés dans le cadre de la construction d'un programme de dialogue vocal. L'entrée vocale (en haut à gauche) est analysée, les mots sont reconnus, les phrases sont décomposées et interprétées dans leur contexte durant l'étape de raisonnement. Une réponse est ensuite planifiée suivant une structure syntaxique, influençant la sélection de mots utilisés dans la sortie vocale.[12]*

Dès lors, les programmes NLP commencent généralement par déterminer la structure morphologique et la nature des mots avant de s'intéresser à leur agencement, suivant une grammaire particulière, afin de trouver la signification de la phrase qu'ils composent (Figure 3). Ces phrases sont alors analysées afin d'obtenir une représentation globale du contexte et du domaine dans lequel elles interviennent.[13] Ainsi, sept caractéristiques indépendantes sont utilisées afin de comprendre les langages naturels[14] :

- La phonétique qui définit la façon dont les mots sont prononcés à l'oral.
- La morphologie des mots qui regroupe les préfixes, les suffixes et les racines des mots.
- La nature des mots qui permet de les catégoriser en fonction de leurs propriétés lexicales.
- La syntaxe qui représente la façon dont les mots se combinent pour former des phrases.
- La sémantique, ou signification, des phrases dépendante du sens des mots qui les composent.
- Le discours qui nous renseigne sur la structure générale du texte.
- Et enfin, la pragmatique qui fournit la signification du texte dans le contexte de son emploi.

Un algorithme de traitement automatique du langage naturel est programmé pour tenir compte de l'entièreté ou d'une partie de ces différents paramètres. Les différentes méthodes développées pour arriver à cette fin peuvent être classées en trois catégories :

L'approche symbolique basée sur l'utilisation de règles et de lexiques générés par la réflexion humaine. Autrement dit, l'idée derrière cette approche est de fournir au programme des règles pré-établies par des experts linguistiques et considérées comme valides de manière générale pour un langage précis.

L'approche statistique résultant de l'analyse d'exemples récurrents de phénomènes linguistiques. Les modèles générés par cette approche isolent des schémas présents dans plusieurs observations via des analyses mathématiques effectuées sur de grands jeux de données. En identifiant certaines tendances parmi un grand nombre d'exemples, le système peut développer ses propres règles linguistiques qui permettent l'analyse d'autres jeux de données et/ou la génération de nouvelles phrases.

L'approche connexionniste où les approches symbolique et statistiques sont mêlées. Cette approche est généralement formée d'un ensembles de règles de langages connectées ou non en fonction du type de donnée d'entrée, identifié par une méthode d'inférence statistique.

Avec l'augmentation du nombre de données disponibles, ce dernier type d'analyse hybride est celui le plus prévilégié de nos jours grâce à sa robutesse, sa flexibilité et sa capacité de modéliser des phénomènes où le comportement linguistique est difficillement identifiable.

2 Objectifs

Malgré les progrès considérables de ces dernière années dans les domaines du traitement automatique du langage naturel et de la reconnaissance d'entités nommées, certaines recherches[15] ont démontré que ces algorithmes, souvent développés dans le cadre de la résolution d'un problème spécifique, subissent une perte de performances lorsqu'ils sont appliqués sur des problèmes similaires issus d'un autre domaine. Pour chaque nouveau domaine d'application, il est, dès lors, nécessaire de générer un nouveau jeu de règles et de statistiques afin de produire des résultats satisfaisants.

Le but recherché lors de ce mémoire était de construire une libraire pour le langage de programmation "R" contenant les outils nécessaires à la création de réseaux d'interactions entre gènes à partir de documents texte issus de la littérature scientifique de n'importe quel organisme.

Pour ce faire, plusieurs étapes ont été nécessaires :

1. Construction d'un modèle permettant d'identifier automatiquement la présence de gènes dans des textes non structurés.
2. Identification et classement des interactions entre les gènes identifés.
3. Création d'outils de collecte automatique de documents scientifiques correspondant à un organisme spécifique tel que *Chlamydomonas reinhardtii*.

Ainsi, plusieurs approches et algorithmes ont été testés afin d'obtenir des prédictions à la fois rapides et performantes. À chaque étape, ces performances furent déterminées par comparaison avec des références (gold standard) externes.

3 Matériel et méthodes

3.1 Hardware

L'entièreté des résultats présentés dans ce mémoire ont été obtenus à l'aide du serveur dédié au service de "Biologie des Systèmes" (BioSys) du groupe "PhytoSYSTEMS" de l'Université de Liège dirigé par le Professeur Patrick E. Meyer. Ce serveur tourne sous le système d'exploitation Linux Ubuntu 14.04.4 LTS x86_64 et possède un processeur Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz à 12 coeurs ainsi que 64 Gigas de Ram.
L'accès au langage de programmation "R" s'est fait grâce à la version serveur de RStudio via une session limitée à l'utilisation de 8 des 12 coeurs et à 20 Gigas de Ram. Toutefois, "R" n'utilisant pas de base les capacités de multithreading sans l'utilisation de librairies spécialisées, la totalité des calculs ont été résolus par un seul coeur.

3.2 Environnement et langages de programmation

3.2.1 BioLinux 8

BioLinux 8 est un environnement *Open Source* de type Unix pouvant être installé sur n'importe quel laptop, serveur ou machine virtuelle. Cette distribution ajoute plus de 250 librairies bioinformatiques en plus des programmes de base de Linux Ubuntu 14.04 LTS.[16]

3.2.2 Unix

Unix est un système d'exploitation multitâche et multi-utilisateur développé en 1969 par l'informaticien Kenneth Thompson dans les laboratoires Bell. Celui-ci fut rédigé en langage d'assemblage avant d'être réécrit en C en 1972 afin de créer un logiciel le plus "portable" possible, ne nécessitant que peu de modification du code spécifique à la machine lors de l'installation.
Il a ensuite donné naissance à plusieurs variantes dont les plus populaires à ce jour sont les BSD (notamment FreeBSD, NetBSD et OpenBSD), GNU/Linux, iOS et macOS. Ces systèmes d'exploitation de type Unix sont caractérisés par plusieurs concepts telles que la sauvegarde de données sous forme de documents texte, un système de gestion de fichiers hiérarchique, l'utilisation d'appareils et de certaines communications inter-processus (IPC) en tant que fichier ou encore l'utilisation d'un grand nombre de petits programmes pouvant être reliés ensemble via un interpréteur en ligne de commande.[17]

3.2.3 R

R est un langage de programmation *Open Source* implémenté à partir du langage de programmation S et spécialisé dans le calcul statistique, le traitement de données et l'analyse graphique.
De plus, sa portée d'analyse est facilement extensible via l'utilisation de librairies spécialisées.
Habituellement utilisé via un terminal à l'aide de lignes de commande, il existe toutefois sous forme d'une interface graphique plus accessible grâce à RStudio.[18]

3.3 Librairies R

3.3.1 CoreNLP

Librairie fournissant l'interface minimum nécessaire à l'utilisation des annotateurs de la librairie java "Stanford CoreNLP". Celle-ci fournit les méthodes couvrant les tâches de tokenisation, d'étiquetage morpho-syntaxique, de lemmatisation, de reconnaissance d'entités nommées, de détection par co-référence et d'analyse des sentiments.[19]

3.3.2 e1071

Librairie regroupant un ensemble de fonctions permettant l'utilisation de l'analyse de classes latentes, la transformée de Fourier à court terme, le partitionnement de données, des séparateurs à vaste marge, la résolution du problème de plus court chemin, la classification naïve bayésienne et bien d'autres.[20]

3.3.3 RandomForest

Implémentation R de l'algorithme de forêt d'arbres décisionnels de Breiman (basée sur le code Fortran original de Breiman et Cutler) pour la classification et la regression.[21]

3.3.4 SnowballC

Interface R pour la librairie C libstemmer implémentant l'algorithme de racinisation de Porter visant à réduire un mot à sa racine commune afin d'aider à la comparaison de vocabulaires. Les langages actuellement supportés sont le Danois, le Néerlandais, l'Anglais, le Finnois, le Français, l'Allemand, le Hongrois, l'Italien, le Norvégien, le Portugais, le Roumain, le Russe, l'Espagnol, le Suèdois et le Turc.[22]

3.4 Données

3.4.1 CRAFT

Le corpus de textes complets richement annotés (Colorado Richly Annotated Full Text Corpus, CRAFT) est un corpus manuellement annoté composé de 67 articles complets issus de journaux biomédicaux.[23] Chaque article fait partie de la collection d'articles en libre accès de PubMed Central (PMC OA). En tout, plus de 21.000 phrases contenant près de 560.000 mots ont été manuellement analysées afin d'extraire environ 100.000 entités correspondant à, au moins, une de ces 7 catégories :

- Entité chimique d'intéret biologique
- Ontologie cellulaire
- Ontologie des protéines
- Ontologie des séquences biologiques
- Ontologie des gènes
- Gène Entrez
- Taxonomie du NCBI

Ce corpus a permis, dans le cadre de ce mémoire, de construire les différents modèles de classification permettant de différencier les noms de gènes. Ainsi, seules les annotations du type "Gène Entrez" furent sélectionnées pour la classification. 70% de ce corpus fut utilisé pour l'entraînement des modèles et les 30% restant servirent à valider la performance des différents modèles.

3.4.2 PubMed

PubMed est un moteur de recherche libre d'accès permettant d'accéder à la base de données d'articles scientifiques MEDLINE.[24] Il peut être utilisé via navigateur par protocole HTTP ou via l'utilisation de plusieurs programmes "E-Utilities" permettant la recherche et le téléchargement automatique d'articles via le protocole FTP.

Ces programmes furent utilisés afin de télécharger 131.605 résumés d'articles dont le sujet correspondait à *Escherichia coli* en tant que sujet MeSH majeur (Major MeSH Topic). Il faut toutefois spécifier que 29.189 de ces résumés ne contiennent uniquement que le nom de l'article. Ces résumés servirent à la fois à l'évaluation des modèles de classification en situation réelle d'utilisation et à la mesure de l'impact du changement d'environnement dans le cadre de la construction de modèles sur des textes entiers pour l'analyse de résumés. Par la suite, les résumés disponibles pour *Caenorhabditis elegans* et *Chlamydomonas reinhardtii* furent téléchargés afin de valider nos algorithmes sur plusieurs organismes.

3.5 Collecte des données

Le corpus CRAFT 2.0 fut téléchargé manuellement sur le site bionlp-corpora. Parmi les différentes annotations disponibles, seules certaines fournissaient des informations concernant les gènes de la base de donnée “Entrez”.

En raison de sa facilité de lecture, le format GENIA XML (craft-2.0/genia-xml/term/entrezgene) fut retenu. Celui-ci a l’avantage de fournir les annotations à l’intérieur même des articles (inline XML) où les gènes sont encadrés à l’aide des balises `<term sem="EG:XXXXX">` et `</term>`. Le code “EG:” suivi de cinq chiffres est dépendant du gène en question et fait référence à son code d’accès dans la base de donnée Entrez.

Les résumés d’articles PubMed concernant *Escherichia coli*, *Caenorhabditis elegans* et *Chlamydomonas reinhardtii* ont quant à eux été téléchargés automatiquement grâce à la fonction “search-Nfetch” présente dans notre nouvelle librairie. Celle-ci utilise deux E-Utilities fournis directement par le site du NCBI : “ESearch”, permettant de rechercher les identifiants d’articles correspondant à un sujet particulier; et “EFetch”, permettant de télécharger le résumé de ces articles sur base des identifiants.

3.6 Traitement des données

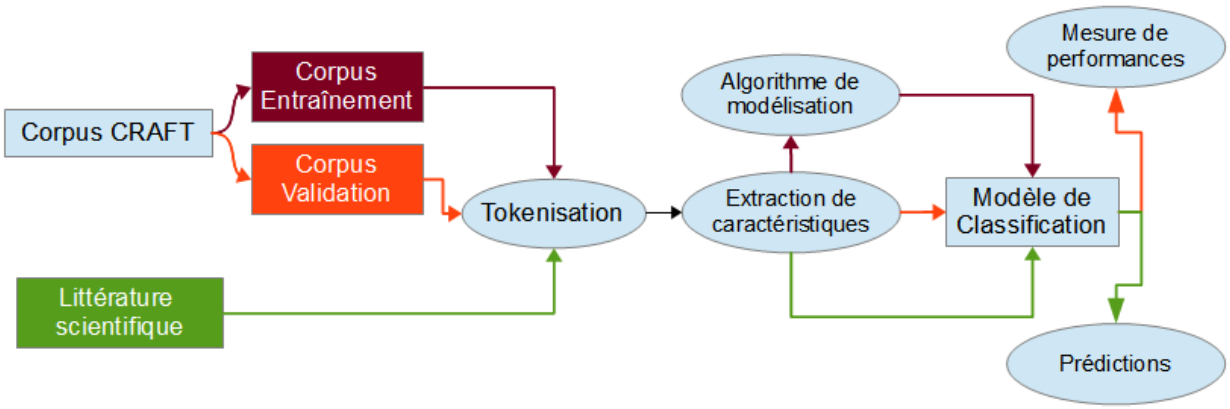


Figure 4: *Représentation schématique des différentes étapes effectuées par la librairie R “GeneMining”. Les documents passent d’abord par une étape de tokenisation afin d’en extraire les différents mots. Les caractéristiques de chaque mot sont ensuite identifiées. Les mots issus du corpus d’entraînement sont utilisés par un algorithme de modélisation afin de construire un classificateur gène/non gène. Les performances de ce modèle sont calculé à l’aide des mots extraits du corpus de validation. Enfin, la classe des mots issus de la littérature scientifique est déterminée grâce à ce même modèle.*

3.6.1 Prétraitement : tokenisation

La tokenisation est l'étape durant laquelle un document est découpé en éléments appellés *tokens*. Ces tokens sont le plus souvent les mots composant les différentes phrases du document. Toutefois, dans certains cas, un token peut être un ensemble de mots consécutifs formant une même entité.[25]

Dans ce mémoire, les documents ont tout d'abord été découpés en phrases sur base de la présence d'un point suivi d'un espace puis d'une lettre majuscule et ce afin d'éviter de couper une phrase en deux lorsqu'un nom raccourci d'espèce tel que "E. coli" est présent. Ensuite, les mots de ces différentes phrases sont isolés en découpant au niveau des espaces entre ces mots. Toujours dans le but de ne pas séparer les noms raccourcis d'espèces, les espaces précédés par un point n'ont pas été pris en compte lors de ces séparation.

Dans le cas des articles annotés du corpus CRAFT, les entitées mono- ou multi-tokens représentant les gènes sont d'abord isolées grâce aux balises avant l'étape de tokenisation des phrases en mots. Tous les autres mots non-balisés sont classifiés en tant que "non-gène". Dans le but de faciliter la comparaison entre tokens, les symboles de ponctuation éventuellement présents au début ou à la fin de chaque mot furent supprimés.

3.6.2 Étiquetage morpho-syntaxique

Un étiqueteur morpho-syntaxique est un programme visant à assigner une étiquette morpho-syntaxique (nom, verbe, adjectif, etc…) à chaque token sur base de leur définition et de leur contexte.[26] Il s'agit là d'une tâche ardue étant donné qu'un même mot peut être associé à plusieurs étiquettes en fonction de son contexte.

On distingue deux grands types d'étiqueteur :

Rule-based utilisant des règles générées manuellement afin de résoudre le problème de l'ambiguité d'étiquetage.

Stochastique soit construits sur base d'un modèle de markov caché (HMM), sélectionnant la séquence d'étiquettes maximisant le produit de la vraisemblance de chaque mot; soit basés sur l'utilisation d'arbres décisionnels ou des modèles d'entropie maximum afin de combiner des caractéritiques probabilistiques.

Durant ce mémoire, l'étiqueteur morpho-syntaxique log-linéaire de Stanford fut utilisé via le package R "CoreNLP" afin d'utiliser les informations grammaticales des mots lors de la classfication. Une fois toutes les étiquettes identifiées, celles-ci servirent aussi afin de regrouper les mots utilisés dans un contexte proche.

3.6.3 Partitionnement par contexte

Bien évidemment, la grammaire d'un mot n'est pas suffisante en elle-même pour classifier ce mot en tant que gène. Un autre élément important pouvant aider à cette classification est le contexte grammatical dans lequel le mot apparaît.
Ce contexte fut défini comme la séquence d'étiquettes morpho-syntaxiques du mot courant et des autres mots l'entourant de part et d'autre dans une fenêtre de trois mots (sept positions). Ces séquences ont ensuite permis de regrouper les mots possédant le même contexte.
Cependant, un simple regroupement sur base de ces séquences mènerait à la création d'une multitude de groupes eux-mêmes proches les uns des autres. Il est en effet logique, par exemple, de regrouper les mots possédant une séquence formée de sept noms avec ceux possédant une séquence de six noms et d'un adjectif. Ce type de regroupement est permis car, grammaticalement, un adjectif est "proche" d'un nom.

Il était donc nécessaire d'évaluer les distances entre les différentes étiquettes morpho-syntaxiques pour réaliser ce type de rapprochement. Disposant d'un large corpus, celles-ci furent calculées statistiquement sur base de la distance physique médiane (nombre de mots séparant deux étiquettes spécifiques) entre chaque couple d'étiquettes pour l'entièreté du corpus CRAFT.

Une fois ces distances obtenues, les différents mots ont pu être regroupés en plusieurs ensembles grâce à l'algorithme de partionnement en k-moyennes de Lloyd.[27][28] Ce type de partionnement fonctionne en plusieurs étapes (Figure 5) :

1. Sélection, généralement aléatoire, de k "centres" initiaux autour desquels les différentes données devront être regroupées.
2. Identification, pour chaque donnée, du centre le plus proche et regroupement des données assignées à un même centre.
3. Mise à jour des k centres sur base de la moyenne des données assignées à chaque groupe.
4. Application itérative des étapes 2 et 3 durant un nombre défini d'étapes ou jusqu'à convergence afin de minimiser la somme totale des distances entre les données et les différents centres.

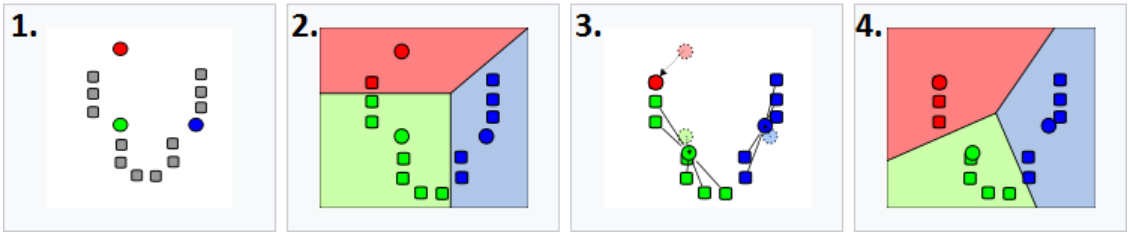
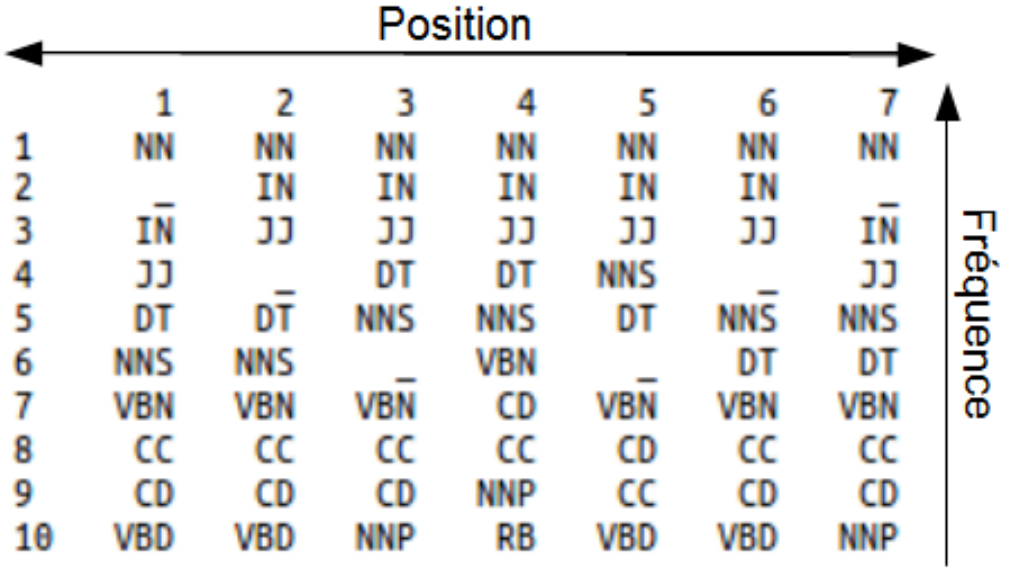


Figure 5: *Illustration des quatre étapes principales des algorithmes de partionnement en k-moyennes (modifié d'après Weston Pace, 26 July 2007).*

Dans notre cas, cet algorithme fut utilisé afin de regrouper les différentes séquences de sept étiquettes morpho-syntaxiques en 37 groupes. En effet, au lieu de générer aléatoirement les centres initiaux, ceux-ci furent générés sur base du classement des différentes étiquettes pour chacune des sept positions sur base de leur fréquence d'apparition à cette même position (Figure 6).[29]

Comme 37 étiquettes morpho-syntaxiques peuvent être assignées, 37 centres différents ont pu être générés. Ainsi, on remarquera que pour chacune des sept positions l'étiquette la plus commune est "NN" correspondant à la présence d'un nom commun singulier.



	1	2	3	4	5	6	7
1	NN	NN	NN	NN	NN	NN	NN
2	_	IN	IN	IN	IN	IN	_
3	IN	JJ	JJ	JJ	JJ	JJ	IN
4	JJ	_	DT	DT	NNS	_	JJ
5	DT	DT	NNS	NNS	DT	NNS	NNS
6	NNS	NNS	_	VBN	_	DT	DT
7	VBN	VBN	VBN	CD	VBN	VBN	VBN
8	CC	CC	CC	CC	CD	CC	CC
9	CD	CD	CD	NNP	CC	CD	CD
10	VBD	VBD	NNP	RB	VBD	VBD	NNP

Figure 6: *Visualisation des 10 premiers centres générés sur base de la fréquence d'apparition des différentes étiquettes morpho-syntaxiques à chaque position. Plus une étiquette est fréquente, plus elle sera utilisée tôt dans la formation des centres. Les étiquettes utilisées sont celles du "Penn Treebank POS Tags". Les "_" représentent l'absence de mots (début et fin de phrase).*

Les données d'entrainement du corpus CRAFT ont ensuite été groupées autour de ces centres initiaux en fonction de leur distance avec ceux-ci. Ces distances étaient calculées sur base de la somme des distances entre étiquettes pour chaque position et le centre possédant la distance totale la plus faible fut déterminé pour chaque donnée. Une fois les groupes obtenus, le nouveau centre de ceux-ci fut calculé en déterminant, pour chaque position, l'étiquette minimisant la distance avec l'ensemble des données appartenant à ce groupe. Ces étapes de regroupement et de mise à jour des centres ont été répétées jusqu'à convergence.

Une fois la convergence atteinte, les groupes ne possèdant aucun point furent supprimés. Le numéro du groupe assigné à chaque donnée d'entraînement sera considéré comme une caractéristique à part entière lors de l'étape de création du modèle.

Les dernier centres calculés seront sauvegardés et serviront à assigner un groupe aux nouvelles données (validation et autres) lors de l'étape de classification.

L'ajout de cette caractéristique permit de remplacer à elle seule un grand nombre d'autres caractéristiques. En effet, les informations concernant le contexte dans lequel un mot intervenait était fournies, au départ, en regardant les caractéristiques des mots situés de part et d'autre du mot courant dans une fenêtre de trois mots. Elle permit donc de diviser par sept le nombre total de caractéristiques, réduisant fortement la complexité de construction du modèle.

3.6.4 Caractéristiques basées sur l'utilisation de règles

Bien qu'une nomenclature de gène unique et commune à tous les organismes n'existe pas, certaines caractéristiques sont plus présentes chez ceux-ci, comparées aux autres mots communs de l'anglais scientifique.

Ainsi, une série de règles[30] permettant de capturer ces spécificités furent développées. Ces règles, définies manuellement, sont généralement arbitraires et dépendantes de la compréhension ainsi que de l'expérience du programmeur dans le domaine concerné.

Celles-ci sont portées principalement sur l'étude de la morphologie des mots et exigent beaucoup de temps pour être identifiées et validées. De plus, ces règles ne sont généralement pas mutuellement exclusives et peuvent donc mener à des interférences.

Parmi l'ensemble de règles developpées et testées, seuls les suivantes furent sélectionnées dans la version finale du programme :

NB_Upper Nombre de lettres majuscules présentes dans le mot.
NB_Lower Nombre de lettres minuscules présentes dans le mot.
NB_Letter Nombre total de lettres minuscules et majuscules présentes dans le mot.
NB_Digit Nombre de chiffres présents dans le mot.
NB_Special Nombre de caractères spéciaux (ponctuation, …) présents dans le mot.
NB_Vowel Nombre de voyelles présentes dans le mot.
NB_Consonant Nombre de consonnes présentes dans le mot.
Max_cons_consonant Nombre maximum de consonnes consécutives présentes dans le mot.
NB_Greek Nombre de lettres grecques présentes dans le mot.

Chacune des ces caractéristiques fut aussi transformée en ratio en la divisant par le nombre total de caractères présents dans le mot, portant le nombre de caractéristiques de cette catégorie à 18. L'utilisation conjointe de ces nombres et ratios s'est montrée plus adaptée pour la classification que l'un ou l'autre seul.

Une 19ème règle, binaire cette fois-ci, fut utilisée, indiquant la présence ou l'absence de chiffre romain à la fin des mots.

3.6.5 Caractéristiques statistiques

Trois autres caractéristiques des mots, plus statistique cette fois, furent aussi développées.[30] Celles-ci consistèrent à réduire l'ensemble des mots du corpus d'entrainement CRAFT à leurs motifs. Pour ce faire, l'entièreté des lettres majuscules a été transformée en "A" majuscule, les minuscules en "a" minuscule, les chiffres en "0" et les caractères spéciaux en "-".

Ensuite, le nombre d'occurences de chaque motif fut calculé et chaque mot fut associé à l'occurence de son motif. En effet, selon l'hypothèse qu'un mot possèdant un motif peu courant a plus de chance d'être un gène, il existe une valeur d'occurence limite optimale en dessous de laquelle une majorité de mots sont des gènes.

De plus, si un mot possède un motif jamais rencontré jusqu'alors dans le corpus d'entraînement celui-ci aura une valeur d'occurence de 0 et aura plus tendance à être considéré comme un gène lors de la phase de classification. Sur ce même principe, les motifs raccourcis, où les symboles consécutifs des motifs ont été abrégés (Aaaa-00 devient Aa-0), et les racines (stem) de ces mots furent collectés et organisés en fonction de leur occurence.

3.6.6 Caractéristiques basées sur l'utilisation de dictionnaires

Finalement, une dernière série de caractéristiques fut déterminée sur base de l'utilisation de dictionnaires externes regroupant les différents préfixes et suffixes couramment rencontrés dans la langue anglaise.[30] La présence d'un de ces préfixe/suffixe sur les 1, 2, 3, 4 et 5 premières/dernières lettres d'un mot se traduit par la présence d'un "1" sur la caractéristique binaire correspondante, soit un total de 10 caractéristiques pour cette catégorie.

3.6.7 Détection des limites des entités

Un problème majeur rencontré lors de la reconnaissance d'entités nommées pouvant être composées de plusieurs tokens est de savoir faire la différence entre une suite d'entités mono-token consécutives et une entité multi-tokens.

Dans ce but, la simple classification en deux classe "gène" ou "non-gène" fut remplacée par une autre à cinq classes : B, I, L, O, U. Ces cinq classes permettent d'identifier les non-gènes, considérés comme "en-dehors" (**O**utside, "O") des entités d'intérêt, les gènes mono-token constitués d'une seule

"unité" (**U**nit, "U") et les gènes multi-tokens constitués au moins d'un token de début (**B**eginning, "B"), d'un token de fin (**L**ast, "L") et éventuellement de tokens situés entre les deux (**I**nside, "I").

Bien sur, il ne s'agit là que d'un schéma d'encodage parmi d'autres mais il a toutefois été démontré par Ratinov et Roth[31] comme étant plus facile à apprendre lors de la modélisation que le schéma populaire "BIO", produisant donc de meilleurs prédictions.

3.7 Algorithmes de modélisation

Contrairement à l'utilisation d'une règle unique de nomenclature pour classifier les gènes, nous nous retrouvons ici avec une multitude de caractéristiques, chacune plus ou moins informative pour l'identification des gènes, qu'il va falloir combiner afin d'obtenir le meilleur modèle de classification possible. C'est dans ce contexte qu'interviennent les algorithmes d'apprentissage automatique supervisés.

Certains de ces algorithmes ont déjà été utilisés dans les tâches de reconnaissance d'entités nommées biomédicales mais nous discuterons principalement de l'algorithme de forêt d'arbres décisionnels qui n'a pas encore été utilisé, à notre connaissance, pour la classification de mots en tant que gènes dans la littérature biomédicale.

3.7.1 Classification naïve bayésienne

Les classifieurs naïfs bayésiens sont des classifieurs probabilistiques simples, basés sur l'hypothèse (dite naïve) que les différentes caractéristiques à combiner sont indépendantes entre elles. Grâce à cette commodité mathématique, le calcul des différentes probabilités est fortement simplifié, rendant cette technique facile à comprendre et à implémenter. En effet, admettons une tâche de classification entre k classes possibles $C_1, C_2, ..., C_k$. Chaque mot, représenté par son vecteur de n caractéristiques $x = (x_1, x_2, ..., x_n)$, sera assigné à une classe selon la formule de maximum à posteriori suivante :

$$argmax_{K=1,2,...,k}\ p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Malgré ces simplifications extrêmes, cette technique s'est montrée compétitive à de nombreuses reprises avec d'autres algorithmes d'apprentissage automatique et sert souvent de ligne de base pour comparer les performances d'autres méthodes de catégorisation de textes.[32] C'est pour ce dernier rôle que cette technique fut utilisée dans le cadre de ce mémoire.

3.7.2 Séparateurs à vaste marge

Les séparateurs à vaste marge, Support Vector Machine (SVM) en anglais, sont des algorithmes d'apprentissage automatique supervisés pouvant être utilisés à la fois pour des problèmes de classification ou de regression. Ceux-ci représentent chaque exemple sous la forme d'un point dans un espace à n dimensions (où n représente le nombre de caractéristiques assignées à chaque exemple).

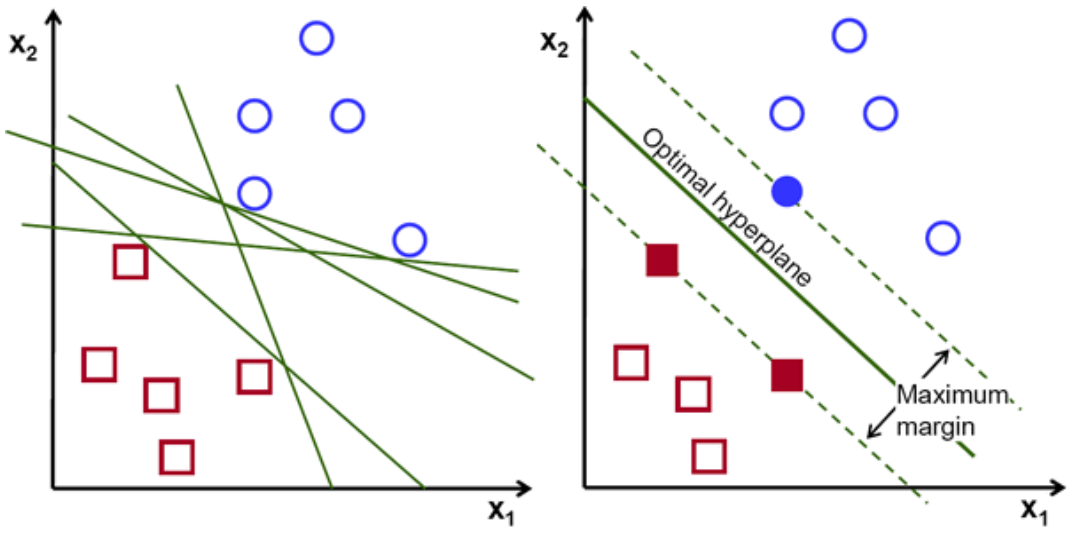


Figure 7: *Application d'un séparateur à vaste marge pour la classification de données en deux classes. Une multitude d'hyper-plans capables de séparer ces données existent (gauche) mais seul un de ces hyper-plans maximise la marge de séparation (droite) (OpenCV Dev Team).*

Dans le cas de la classification entre deux classes, chaque point est alors assigné à une de ces classes et le but de l'algorithme est de trouver le meilleur hyper-plan à *n-1* dimensions capable de séparer les deux classes de points avec la plus large marge possible.[33] En effet, bien qu'il existe plusieurs hyper-plans capables de séparer les données, seul celui maximisant sa distance avec les points les plus proches de part et d'autre (les vecteurs de support) sera choisi car au plus la marge est large, au plus l'erreur de généralisation sera faible (Figure 7).

En pratique, les données fournies ne sont généralement pas linéairement séparables. Les algorithmes SVM surmontent alors ce problème en ayant recours au "kernel trick" qui va permettre de représenter le problème dans une dimension d'ordre supérieur où une séparation linéaire est alors possible. Toutefois, il se peut que les données soient toujours difficilement séparables après cette manipulation à cause, par exemple, de la présence d'échantillons mal classés. Pour résoudre ce genre de problème, un séparateur à marge souple, minimisant les erreurs de classement via l'introduction d'une constante de pénalité, sera utilisé. Enfin, si les données doivent être séparées en plus de deux classes, plusieurs stratégies existent.

Les deux plus connues sont le “seul contre tous”, où un classifieur est construit pour chaque classe en séparant les points assignés à cette classe de tous les autres, et le “un contre un”, où les classes sont confrontées entre elles une par une.

De par sa popularité[34], cette technique fut utilisée durant les premières étapes de ce mémoire avant d'être remplacée par l'algorithme de forêt d'arbres décisionnels, moins gourmand en ressources et plus adapté au traitement de caractéristiques non numériques.

3.7.3 Forêt d'arbres décisionnels

La forêt d'arbres décisionnels est une technique d'apprentissage automatique visant à combiner plusieurs ensembles de prédicteurs afin d'obtenir une meilleure performance. Chaque prédicteur consiste en un arbre de décisions dont le résultat sera comparé à ceux des autres arbres formant la forêt.

Un arbre de décisions est un diagramme en forme d'arbre illustrant tous les résultats possibles issus d'une prise de décision (Figure 8). Chaque branche de cet arbre représente une décision amenant soit à une autre décision, soit à une valeur de la variable cible (discrète lors d'une classification; continue pour une régression).
Ce type de modèle est construit sur base de la segmentation récursive des données d'entraînement à l'aide des différentes décisions appliquées jusqu'à l'obtention de plusieurs groupes de données possédant majoritairement la même valeur de variable cible.[35] L'ordre de sélection de ces décisions se base sur un critère d'homogénéité des sous-ensembles produits à chaque étape de décision, lui-même dépendant du type d'algorithme utilisé (principalement l'entropie de Shannon ou le coefficient de Gini).

Dans le cas de l'utilisation d'une forêt d'arbres décisionnels pour une classification, chaque arbre votera pour une classe et celle la plus populaire parmi l'ensemble de la forêt sera choisie. Pour une regression, une moyenne des valeurs obtenues pour chaque arbre sera calculée.

Pour arriver à ce résultat, chaque arbre doit être le plus différent possible des autres arbres formant la forêt tout en classifiant au mieux les données d'entraînement. Dans ce but, chaque arbre sera construit à partir d'un ensemble partiel aléatoire de données d'entrainement sur lequel chaque décision pourra être choisie parmi un ensemble partiel aléatoire de caractéristiques.

Grâce à l'introduction de cette variabilité parmi les arbres, une série d'experts spécialisés dans un domaine (sous ensemble de données d'entraînement) sera générée. Ces experts, très étroits d'esprit, auront tendance à surapprendre les données fournies résultant en une série de classifieurs spécialisés très peu généralisables et très sensibles au bruit.

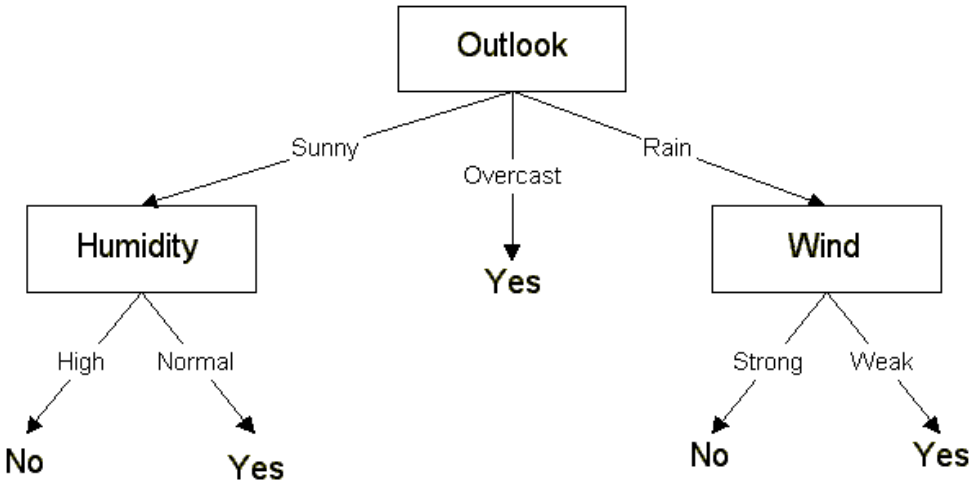


Figure 8: *Exemple d'arbre décisionnel simple montrant l'impact de la météo sur la possibilité de jouer au baseball. Deux "classes" sont possible : "yes", un match a été joué ou "no" pour l'inverse. Trois caractéristiques météorologiques sont analysées : le temps ambiant, l'humidité et la force du vent.[36]*

L'utilisation conjointe des ces experts via un système de vote ou de moyenne permet de réduire cette haute variance individuelle afin d'obtenir un modèle final complet performant.[37]

Dans la plupart des cas, les problèmes réels de classification ne sont pas équilibrés : une des classes est représentée par une minorité de données. Dès lors, les forêts d'arbres décisionnels, visant à minimiser le taux d'erreur global, vont porter peu d'attention à la classification correcte de ce petit nombre de données. Une des techniques appliquée généralement pour contrecarrer ce problème consiste à forcer l'équilibrage des classes en sous-échantillonnant la classe majoritaire ou en sur-échantillonnant la minoritaire.[38]

Dans notre cas, il est clair que le nombre de gènes présents dans le corpus CRAFT est largement inférieur au nombre total de mots constituant ce même corpus. Pour rééquilibrer ce problème, chaque arbre fut construit à partir d'un jeu de données constitué de l'entièreté des gènes et d'un nombre équivalent de non-gènes, sélectionnés aléatoirement pour chaque arbre.
Cependant, un tel sous-échantillonnage de la classe majoritaire entraîne souvent une perte d'information, une grande partie de la classe majoritaire n'étant pas utlisée. Dès lors, ces arbres spécialisés dans la reconnaissance des gènes furent combinés avec d'autres arbres, construits suivant la méthode habituelle des forêts d'arbres décisionnels, aidant plus à la reconnaissance de la classe majoritaire.

3.8 Mesure de performance

La performance d'un classifieur est calculée en appliquant celui-ci sur un jeu de données où la classe de chaque mot aura été identifiée au préalable. Il s'agit généralement d'une partie du corpus de départ servant uniquement à cette tâche de validation et qui ne sera donc pas utilisée lors de la phase d'entraînement du modèle.

Ce faisant, quatre cas peuvent être observés lors d'une classification binaire :

- Un mot est prédit comme positif et sa classe réelle est positive : la prédiction est correcte et l'ensemble des mots correspondant à ce cas sont appelés "vrais positifs".
- Un mot est prédit comme négatif et sa classe réelle est négative : la prédiction est correcte et l'ensemble des mots correspondant à ce cas sont appelés "vrais négatifs".
- Un mot est prédit comme positif et sa classe réelle est négative : la prédiction est incorrecte (erreur de type I) et l'ensemble des mots correspondant à ce cas sont appelés "faux positifs".
- Un mot est prédit comme négatif et sa classe réelle est positive : la prédiction est incorrecte (erreur de type II) et l'ensemble des mots correspondant à ce cas sont appelés "faux négatifs".

Connaissant le nombre de mots assignés à chaque cas ci-dessus, deux mesures de performances (Figure 9) peuvent être extraites :

La précision du modèle correspondant à la fraction de mots correctement assignés à la classe positive. Elle est calculée en divisant le nombre de vrais positifs par la somme des vrais positifs et des faux positifs.

Le rappel du modèle correspondant à la fraction de mots positifs récupérée. Il est calculé en divisant le nombre de vrais positifs par la somme des vrais positifs et des faux négatifs.

Ces deux mesures peuvent être combinées afin de faciliter la comparaison entre différents modèles de classification. Dans le but de favoriser les modèles possédant à la fois une bonne précision et un bon rappel, la moyenne harmonique de ces deux valeurs est calculée. La valeur de cette moyenne est appellée le F1-score et est calculée selon la formule suivante :

$$F_1 = 2 * \frac{1}{\frac{1}{Rappel} + \frac{1}{Precision}} = 2 * \frac{Precision * Rappel}{Precision + Rappel}$$

Bien que cette méthode d'évaluation de la performance des modèles de classification binaire ne tienne pas compte du taux de vrais négatifs obtenus, celle-ci nous aura permis, dans le cadre de ce mémoire, de calculer la véracité du modèle pour l'identification des gènes, considérés ici comme la classe "positive".

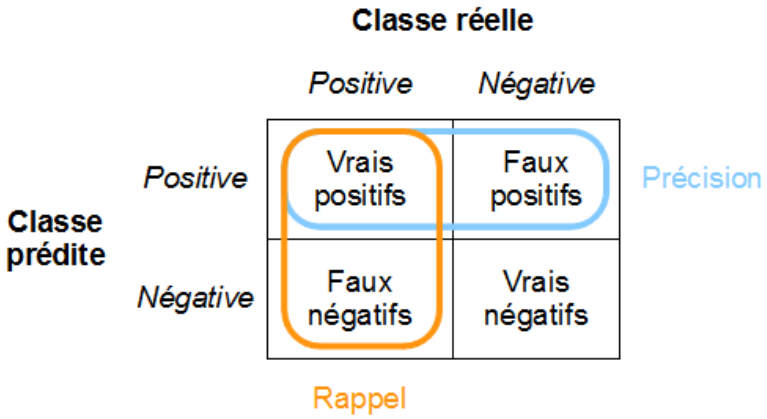


Figure 9: *Représentation sous forme d'une matrice de confusion des quatre cas possibles rencontrés lors de la validation d'une classification binaire.*

Dans le but de faciliter le calcul, le décompte s'est fait mot par mot, et ce, même pour les entités multi-tokens, en transformant les mots classés "O" en "Non-gène" et les autres en "gène".

3.9 Interactions entre gènes

Une fois les gènes isolés des autres mots, les relations entre ceux-ci peuvent être identifiées. La façon la plus instinctive d'obtenir ces relations consiste à utiliser l'hypothèse de co-occurence. Celle-ci suppose que tous mots apparaissant dans un environnement proche tendent à être corrélés.[39] Cet environnement est généralement limité à une phrase mais peut s'étendre à des concepts plus vastes tels qu'un paragraphe voire un résumé entier (abstract) d'un article scientifique.

Ainsi, plus deux gènes sont cités souvent ensemble au travers des différents articles analysés, plus leur relation sera considérée comme forte. Pour chaque interaction, on peut analyser le nombre de phrases où la relation apparaît ou bien le nombre de documents possédant cette même relation. Enfin, la véracité de chaque interaction peut être normalisée en tenant compte de la probabilité d'obtenir cette co-occurence de façon aléatoire. Concrètement, ce rapport consiste à calculer l'indice de Jaccard selon la formule suivante, où $|A \bigcap B|$, $|A|$ et $|B|$ représentent respectivement le nombre de co-occurences entre les gènes A et B, le nombre total d'occurences de A et le nombre total d'occurences de B :

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A \bigcap B|}{|A| + |B| - |A \bigcap B|}$$

$$0 \leq J(A,B) \leq 1$$

4 Résultats et discussion

4.1 Résultats intermédiaires

4.1.1 Recherche de gènes via nomenclature

Dans un premier temps, une simple recherche de gènes via utilisation de nomenclatures fixes officielles fut testée. Pour *Chlamydomonas reinhardtii*, quatre règles préférentielles de nomenclature[40] furent extraites :

- Gènes nucléaires : trois lettres majuscules suivies par un nombre (ex : TSB1).
- Alternative pour les gènes nucléaires : trois lettres majuscules suivies par au moins une autre lettre majuscule (ex : PETC).
- Gènes des organelles : trois lettres minuscules suivies par un nombre (ex : atp1).
- Alternative pour les gènes des organelles : trois lettres minuscules suivies par au moins une lettre majuscule (ex : psaB).

Cependant, ces quatre règles seules n'ont permis que de récupérer 69.48 % de la liste de gènes de *Chlamydomonas reinhardtii* téléchargée le 14 octobre 2016 sur le site internet de la base de données Uniprot.
Il est toutefois important de préciser que les noms d'ORF (CHLREDRAFT) ne furent pas gardés dans cette liste de référence. Afin de porter ce nombre à 87.42 %, quatre autres règles de nomenclatures, basées sur des combinaisons des quatre nomenclatures précédentes, furent ajoutées :

- Quatre lettres majuscules suivies par un nombre (ex : TUBA1).
- Trois ou quatre lettres majuscules suivies par un nombre puis une autre lettre majuscule (ex : DHC1B).
- Trois ou quatre lettres majuscules suivies par un nombre puis une autre lettre minuscule (ex : SCLB1a).
- Une lettre majuscule suivie d'au moins deux lettres minuscules et d'un nombre (ex : Ppc1).

Lorsque ces huit règles ont été appliquées sur l'ensemble des 4943 résumés d'articles scientifiques concernant *Chlamydomonas reinhardtii* disponibles sur la base de données PubMed le 17 octobre 2016, seul 24.72 % des "gènes" identifiés dans ces articles étaient présents dans la liste de référence d'Uniprot. Cela est notamment dû au fait que ces huit règles ne sont pas uniquement spécifiques aux gènes de *Chlamydomonas reinhardtii*. En effet, de nombreuses autres entités suivent ces nomenclatures telles que les noms de souche (HTY217), les noms d'enzymes (NADH) ou encore des mutations ponctuelles (Leu51). En plus de produire des résultats médiocres, cette technique de comparaison de nomenclatures est limitée à la reconnaissance des gènes de *Chlamydomonas reinhardtii* car elle

a été construite spécifiquement à partir de règles de nomenclature de cette espèce.

Cette expérience aura tout de même permis de démontrer que l'utilisation de règles de nomenclature seules ne permettait pas de récupérer tous les gènes existant à cause de leur grande diversité de motifs et que ces nomenclatures n'étaient pas uniquement spécifiques aux noms de gènes.
L'ajout d'autres caractéristiques était donc nécessaire afin d'identifier les gènes de n'importe quel organisme dans un document texte.

4.1.2 Sélection d'un algorithme de modèlisation

Avec l'augmentation du nombre de caractéristiques analysées, il fallut trouver un moyen de combiner les informations fournies par ces dernières de la meilleure façon possible afin d'identifier la nature d'un gène. C'est dans ce but que différentes techniques d'apprentissage automatique supervisé furent testées sur le corpus CRAFT. Leur but est d'inférer la probabilité pour un mot d'être un gène sur base de l'analyse d'un jeu de données où les gènes auront été marqués au préalable.

Trois de ces algorithmes furent testés :

La classification naïve bayésienne : méthode de classification simple basée sur la multiplication de probabilités considérées naïvement comme indépendantes les unes des autres. Cette technique nous servira de ligne de base pour considérer les performances des autres méthodes.

La classification par séparateurs à vaste marge : méthode de classification populaire qui, grâce à l'utilisation de l'astuce du noyau (kernel trick), va permettre de séparer linéairement un problème complexe en représentant ses données dans un espace à plus grande dimension. Leur efficacité est toutefois limitée par le nombre de données et le nombre de caractéristiques analysées.

La classification par forêt d'arbres décisionnels : méthode de classification probabilistique basée sur la combinaison d'un grand nombre d'arbres décisionnels. Très adaptée au traitement de larges quantités de données grâce à la sélection aléatoire de données pour la construction de chaque arbre, elle permet aussi de sélectionner automatiquement les caractéristiques les plus utiles pour la classification de par l'utilisation de la théorie de l'information.

Les fonctions nécessaires à l'application de ces méthodes furent fournies par les librairies R "e1071" pour les classifications naïve bayésienne et par séparateurs à vaste marge ainsi que la librairie "randomForest" pour la classification par forêt d'arbres décisionnels. Chacune de ces méthodes fut appliquée sur le corpus CRAFT, formant un modèle de classification à partir des données d'entraînement avant d'être appliquée sur les données de validation. Les performances des trois méthodes sont reprises dans le tableau ci-dessous :

Table 1: Comparaison des performances sur le corpus CRAFT de la classification naïve bayésienne (NB), par séparateurs à vaste marge (SVM) et par forêt d'arbres décisionnels (RF).

Méthode	Précision (%)	Rappel (%)	F1-score (%)	Temps
NB	7.29	**92.69**	13.52	**3m 30s**
SVM	67.00	39.33	49.57	1h 23m
RF	**69.82**	61.28	**65.27**	8m 5s

Notons tout de même que cette comparaison s'est faite en cours de développement et que les caractéristiques des gènes utilisées lors de celle-ci furent différentes de celles référencées au point 3.6. Pour plus d'informations sur les caractéristiques utilisées et sur les différentes options appliquées, veuillez consulter le papier de conférence suivant, publié spécialement à cet effet.[41]

Avec son algorithme simpliste, c'est la classification naïve bayésienne qui obtient le meilleur score de rappel en récupérant plus de 92 % des gènes présents dans le jeu de données de validation. Toutefois, ce résultat s'accompagne d'un grand nombre de mots ayant été mal classifiés en tant que gène, comme l'indique la faible valeur de précision obtenue. Cette tendance s'inverse avec l'utilisation de la classification par séparateurs à vaste marge. Celle-ci, avec les options par défaut, obtient une bien meilleure précision au dépend du nombre total de gènes récupérés. Néanmoins, en plus de prendre beaucoup plus de temps à modéliser, cette technique fut largement surpassée par les performances obtenues avec la classification par forêt d'arbres décisionnels, rééquilibrant les données d'entraînement suivant la méthode expliquée au point 3.7.3.

4.1.3 Sélection des caractéristiques des gènes

Une fois la méthode de classification sélectionnée, les caractéristiques spécifiques aux gènes ont pu être adaptées afin d'obtenir le meilleur score de performance possible pour cette méthode. Chacune de ces caractéristiques fut identifiée sur base de notre représentation, en tant que biologiste, d'un nom de gène. Par exemple, la présence de lettres et de chiffres dans le même mot est souvent associée à la présence d'un gène. Dès lors, deux caractéristiques peuvent être identifiées : le nombre de lettres et le nombre de chiffres composant les différents mots.
Deux mesures furent utilisées afin de déterminer, par essai-erreur, si une caractéristique devait être inclue au modèle final : leur utilité lors de la classification et leur impact sur la performance des prédictions.

La première de ces deux valeurs est déterminée automatiquement durant l'étape d'entraînement du modèle. En effet, l'utilité d'une caractéristique est directement reflétée par sa fréquence d'utilisation au sein des différents arbres décisionnels composant le modèle. De par l'utilisation de la théorie de l'information lors de la construction des arbres, les caractéristiques les plus informatives seront utilisées plus souvent que les autres. Dans notre cas, l'information apportée par une caractéristique est calculée sur base de la diminution moyenne du coefficient de Gini observée lorsque cette caractéristique est sélectionnée (figure 10).

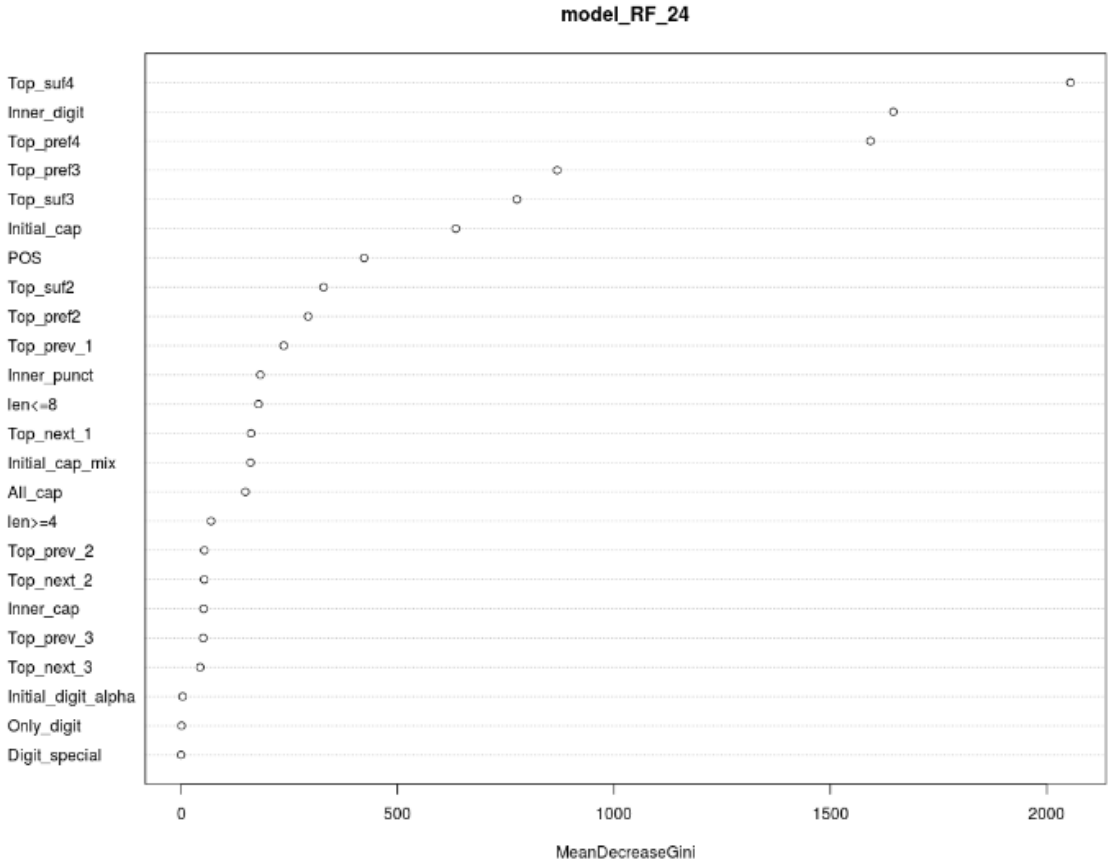


Figure 10: *Graphique représentant la diminution moyenne du coefficient de Gini pour chaque caractéristique durant la phase d'entraînement du modèle de forêt d'arbres décisionnels.*

Toutefois, seules les données d'entraînement sont utilisées lors de cette évaluation. Pour mesurer l'impact réel d'une caractéristique sur le nombre de bonnes prédictions, celle-ci doit être incluse temporairement au modèle avant d'être appliqué sur les données de validation. En pratique, si le score F1 s'améliore après l'ajout de cette caractéristique, celle-ci sera gardée dans le modèle final. Dans le cas contraire, elle ne fournit pas plus d'informations utiles pour la classification et sera donc écartée pour la suite des expériences.

Cependant, même si une caractéristique seule n'est pas utile pour la classification, il se peut qu'elle le soit en présence d'autre caractéristique. Ainsi, les différentes caractéristiques furent le plus souvent ajoutées par groupe puis l'impact de la suppression individuelle de chacune de ces caractéristiques sur le score F1 fut analysé (figure 11).

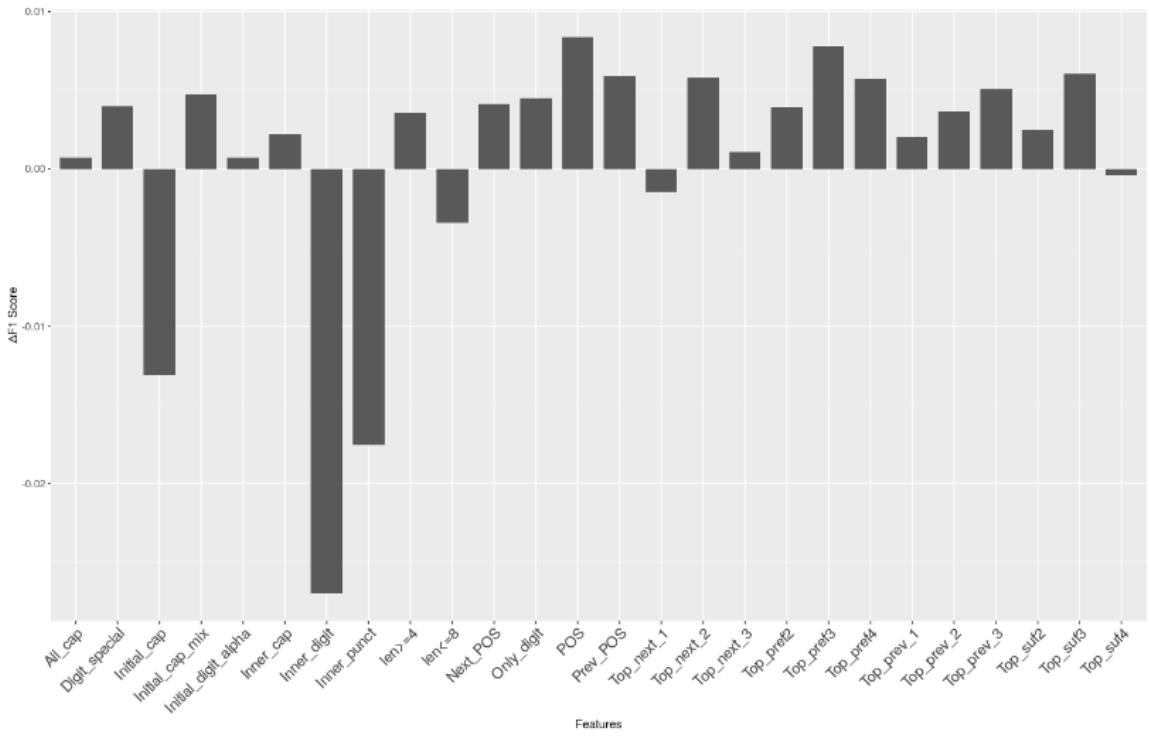


Figure 11: *Représentation graphique de la modification du score F1 après suppression individuelle de plusieurs caractéristiques.*

Ces deux étapes furent répétées tout au long de ce mémoire afin d'obtenir les caractéristiques mentionnées au point 3.6.
Suite à la découverte d'une erreur majeure due à la création de dictionnaires à partir des données d'entraînement et de validation, la plupart des caractéristiques basées sur l'utilisation de ces dictionnaires furent supprimées. La majorité des tests binaires cédèrent leur place à des caractéristiques numériques, plus diversifiées (ex: présence de majuscules remplacée par nombre de majuscules). Enfin, les informations concernant le contexte dans lequel le mot apparaît, au départ fournies par l'analyse des différentes caractéristiques des mots situés de part et d'autre du mot courant, furent définies par la méthode de partitionnement par contexte expliquée au point 3.6.3.
Concernant ce partitionnement, celui-ci permit de remplacer un grand nombre de caractéristiques, simplifiant donc l'étape de construction du modèle. En y regardant de plus près, il s'avère que près de 55% de l'entièreté des gènes fut classé dans le groupe 1, correspondant à un environnement de sept noms consécutifs, tandis que seuls 30% des non-gènes figurent dans ce groupe.

Table 2: Distribution du nombre de gènes (B,I,L,U) et de non-gènes (O) dans les dix premiers ensembles de mots regroupés en fonction de leur contexte.

Classe	Groupe 1	Groupe 2	Groupe 3	Groupe 4	Groupe 5	Groupe 6	Groupe 7	Groupe 8	Groupe 9	Groupe 10
B	350	2	44	1	0	1	0	2	2	0
I	203	3	9	1	1	4	0	0	0	0
L	305	3	47	2	1	2	1	1	0	1
O	40372	13337	17615	6095	4582	3550	1289	562	733	534
U	1715	184	360	84	28	11	17	5	8	3

4.2 Évaluation du modèle

Une fois les caractéristiques optimales identifiées, celles-ci furent combinées dans un modèle final et appliquées sur le jeu de données de validation du cropus CRAFT. Les performances obtenues pour ce modèle furent les suivantes :

Table 3: Décompte du nombre de mots assignés à chaque classe lors de la prédiction en fonction de leur classe réelle.

Classe prédite	B	I	L	O	U
B	2	0	0	8	3
I	0	0	0	4	0
L	7	2	22	49	55
O	275	230	196	133409	1035
U	204	37	270	1029	2401

Soit une précision de 73.37%, un rappel de 63.37% et donc un score F1 de 68% pour l'identification des gènes (B, I, L, U). Les mauvaises prédictions, représentant la perte de 32% de score F1, peuvent être observées sur la figure 12 de par la divergence entre les courbes réelles et prédites. On notera aussi le fait que très peu d'entités à plusieurs mots (B, I, L) ont été correctement classifiées. Bien que ces résultats ne soient pas parfaits, ceux-ci restent tout de même convenables compte tenu des circonstances. En effet, même si certaines méthodes d'identification de gènes dans des documents arrivent à obtenir des scores de F1 supérieurs à 80%, celles-ci sont souvent limitées à l'identification des gènes d'une espèce bien particulière dans un environnement textuel non moins particulier.

Dans notre cas, les différents articles du corpus CRAFT n'ont pas de relations particulières entre eux rendant dès lors l'environnement textuel très diversifié. De plus, une grande partie des ces méthodes a recours à des ressources externes au corpus d'entraînement telles que des listes d'exemples de gènes ou des listes de mots souvent présents dans les phrases mentionnant des gènes.

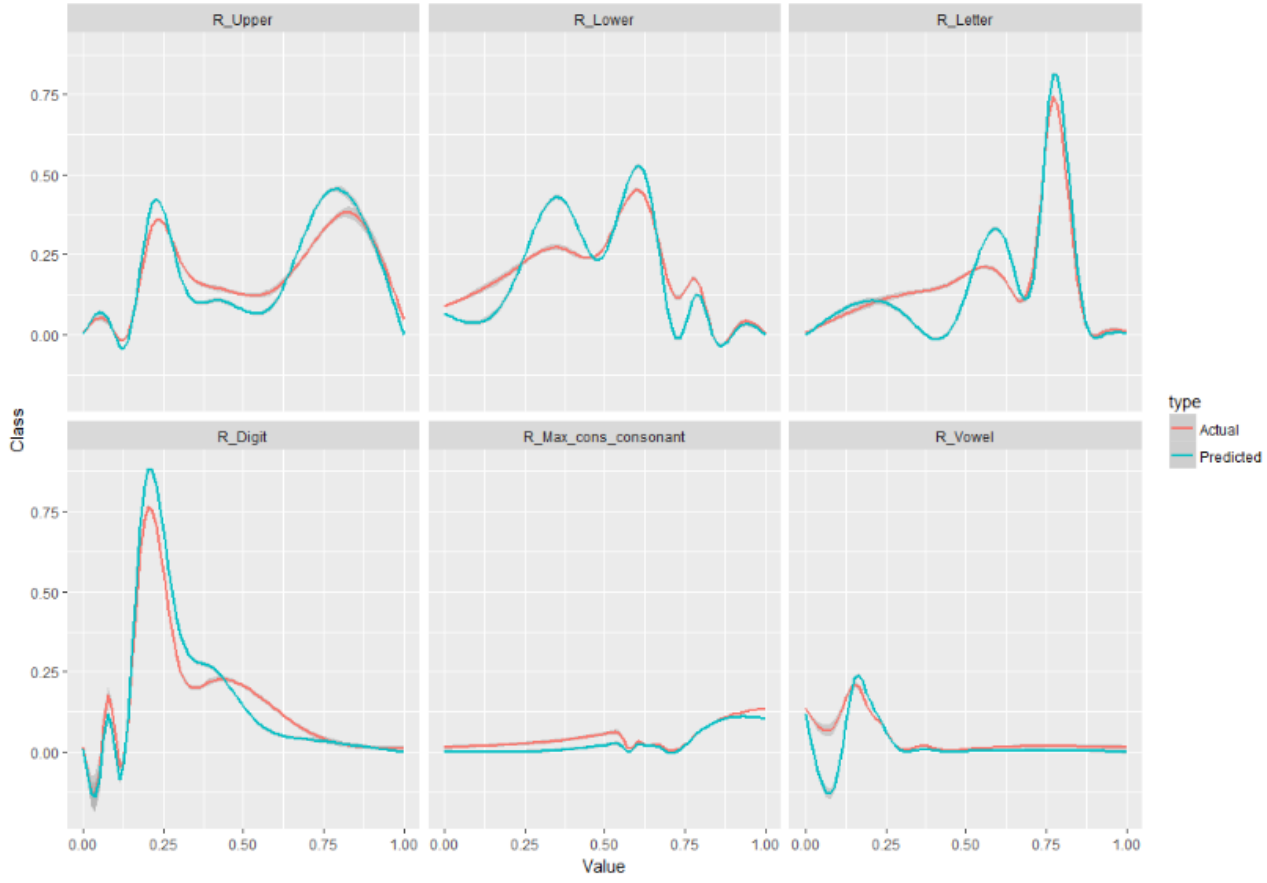


Figure 12: *Comparaison des courbes de probabilités pour un mot d'être un gène en fonction de leurs valeurs pour six caractéristiques. Les courbes rouges représentent les probabilités réellement observées dans le jeu de données de validation et les bleues représentent les probabilités des prédictions générées par le modèle.*

4.3 Recherche de gènes dans un corpus de résumés (abstract) PubMed

4.3.1 *Escherichia coli*

Finalement, une fois les performances de la technique validées, un nouveau modèle fut entraîné sur l'intégralité du corpus CRAFT (jeux d'entraînement et de validation combinés) dans le but de rechercher les gènes dans une situation réelle.

Pour ce faire, ce nouveau modèle fut appliqué sur l'ensemble des résumés (abstracts) disponibles sur PubMed pour l'organisme modèle *Escherichia coli*. Ainsi, un total de 131605 articles correspondant au sujet "*Escherichia+coli*[MeSH+Major+Topic]" fut téléchargé le 16 mai 2017. Toutefois, seul 102416 de ces articles possèdent en réalité un résumé, les autres n'étant composés que d'un titre. Après plus de 53 heures d'analyse, 12866 interactions entre 4118 "gènes" furent isolées de ces documents.

Toutefois, en comparant la liste de gènes obtenus avec ceux disponibles dans la base de données EcoliNet, téléchargée le premier juin 2017, seul 414 gènes semblent appartenir à *Escherichia coli* soit un peu plus que 10% du total de gènes identifiés. En tout, 2337 gènes identifiés (57.75 %) correspondent au moins partiellement à un nom de gène connu dans la base de données "Entrez Gene" du NCBI. Ces "vrais" gènes interviennent dans 5904 interactions c'est à dire 45.89 % du nombre total d'interactions trouvées.

Pour éliminer certaines de ces fausses interactions, il fut décidé d'appliquer un seuil sur le nombre d'apparitions de ces interactions au travers des différents documents. Ce nombre d'occurences, normalisé via le calcul de leur indice de Jaccard (point 3.9), fut calculé par phrase et par document. Dans les deux cas, bien que certaines interactions soient beaucoup plus présentes que d'autres, les médianes du nombre d'occurences des vraies et fausses interactions sont égales, rendant difficile la détermination d'un seuil de séparation.

Limiter la portée de la recherche initiale à un processus biologique spécifique tel que le "métabolisme" ne change pas cette distribution de résultats (70 "gènes", 6 de *E. coli*). Ces résultats peuvent s'expliquer par la non-spécificité de notre modèle à retrouver les gènes d'*Escherichia coli*. En effet, celui-ci fut construit dans le but d'identifier les gènes de n'importe quelle espèce et *Escherichia coli* étant un organisme modèle très souvent soumis à des transformations génétiques, il est probable que notre modèle identifie des gènes d'autres espèces.

4.3.2 *Caenorhabditis elegans*

Afin de vérifier cette hypothèse, notre algorithme fut utilisé sur la littérature scientifique d'un organisme intervenant dans un nombre plus limité d'expériences de transfert de gènes entre espèces différentes : le ver nématode *Caenorhabditis elegans*.
12128 résumés d'articles concernant cet organisme furent ainsi téléchargés le 12 juin 2017 avant d'être analysés afin d'obtenir une liste de 3980 interactions entre 1762 "gènes". Parmi ceux-ci, 848, soit environ 48 %, ont pu être retrouvés dans la base de données d'UniProt pour l'organisme *Caenorhabditis elegans*. De plus, contrairement à *Escherichia coli*, plus de 91 % des gènes identifiés par notre algorithme (1609 gènes) correspondent au moins en partie à un nom de gène connu dans la base de données "Entrez Gene" du NCBI.

Cette distribution de pourcentages peut être plus ou moins observée sur la figure 13 où l'échantillon des 100 gènes intervenant dans le plus grand nombre d'interactions est représenté.

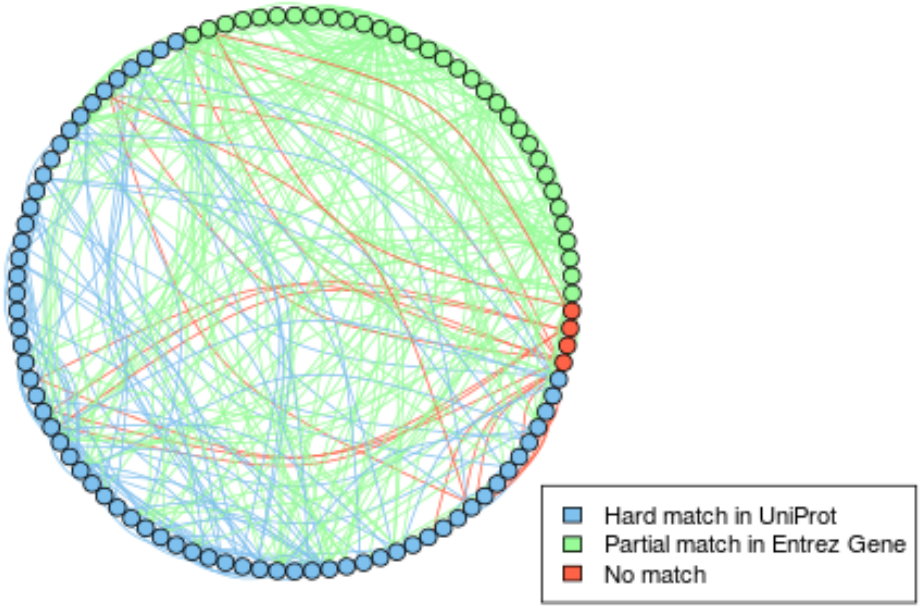


Figure 13: *Représentation graphique des 100 gènes les plus impliqués dans les différentes interactions extraites de la littérature de Caenorhabditis elegans. Les points colorés en bleu correspondent aux gènes pouvant être retrouvés dans la base de données d'UniProt spécifique à Caenorhabditis elegans, les verts sont ceux possèdant au moins une correspondance partielle dans la base de donnée "Entrez Gene" du NCBI tandis que les rouges ne sont retrouvés dans aucune de ces deux bases de données. Les lignes reliant deux points indiquent la présence d'une interaction entre les deux gènes.*

Ces diffèrents gènes non référencés par la base de donnée UniProt de *Caenorhabditis elegans* correspondent à nouveau en majeure partie à des gènes appartenant à d'autres espèces (le gène humain SUMO-1 par exemple) ou à des gènes ne correspondant pas exactement à la nomenclature spécifique de *Caenorhabditis elegans* (Orai1 au lieu de ORAI-1, Smc3 au lieu de SMC-3, …).

Ce changement d'organisme, et donc de nomenclature des gènes, ainsi que la diminution du nombre d'articles ne contenant aucun gène pourrait être responsable de l'amélioration du taux de bonnes prédictions. Au final, l'ensemble des gènes identifiés par au moins une des deux bases de données intervient dans 3470 interactions soit à peu près 87 % du nombre total d'interactions identifiées.

4.3.3 *Chlamydomonas reinhardtii*

Finalement, 2723 articles correspondant au terme Mesh majeur "*Chlamydomonas reinhardtii*" furent téléchargés et analysés le 8 août 2017 en à peu près 47 minutes et 30 secondes, isolant 1692 interactions entre 722 "gènes".
Parmi ceux-ci, 372 (52 %) sont disponibles dans la base de données d'UniProt concernant *Chlamydomonas reinhardtii* et 548 (76 %) possèdent au moins une correspondance partielle dans la base de données "Entrez Gene" du NCBI. Les 157 gènes n'ayant pas de correspondance sont composés de mutations ponctuelles (Arg64, Pro64, …), de nom de mutants (abl-10, Crpgr5, Nfr-4 …), de composés biologiques divers (acyl-CoA, 3Chl, …), d'allèles (amc5, ars44, …) ou encore de souches (CrChR2, …).

	FROM	TO	SCORE
16	ACK1	ACK2	0.20000000
17	ACK1	PAT1	0.20000000
18	ACK1	PAT2	0.20000000
19	ACK2	ACK1	0.20000000
20	ACK2	PAT1	0.29411765
21	ACK2	PAT2	0.29411765
25	ADH1	HYD1	0.11111111
26	ADH1	HYD2	0.14285714
27	ADH1	HYDEF	0.14285714
28	ADH1	LHCSR3	0.09090909
39	Amt1	Irg6	1.00000000
41	Aox1	Aox2	1.00000000
42	Aox2	Aox1	1.00000000
46	APX1	CAT1	0.20000000
47	APX1	FSD1	0.20000000

Figure 14: *Exemples d'interactions entre gènes identifiés automatiquement dans la littérature scientifique de Chlamydomonas reinhardtii disponible sur la base de données PubMed.*

4.4 Librairie R "GeneMining"

La librairie R regroupant les différentes fonctions réalisées au cours de ce mémoire est disponible en téléchargement à l'adresse suivante : http://www.biosys.ulg.ac.be/students/Lete/GeneMining_0.4.0.tar.gz.

Celle-ci peut être installée aisément grâce à la commande :

```
install.packages("http://www.biosys.ulg.ac.be/students/Lete/GeneMining_0.4.0.tar.gz", repos = NULL)
```

Avant sa première utilisation, il est recommandé à l'utilisateur de télécharger les fichiers java nécessaires au bon déroulement de l'étape d'étiquettage morpho-syntaxique réalisé par la librairie R "coreNLP" via la commande :

```
downloadCoreNLP()
```

L'utilisateur devra aussi désigner un dossier de travail via la fonction "setGMWD" où les articles seront téléchargés et/ou analysés.

```
setGMWD("~/Chlamydomonas")
```

Tous les documents présent au format ".txt" dans ce dossier ainsi que ses sous-dossiers seront utilisés par les fonctions d'analyses. Ce dossier de travail devra donc être modifié entre chaque recherche indépendante.

Une fois ces étapes réalisées, l'utilisateur peut utiliser la fonction "searchfFetch" pour télécharger automatiquement des résumés d'articles sur PubMed. Pour télécharger l'ensemble des résumés concernant *Chlamydomonas reinhardtii* par exemple, l'utilisateur devra soumettre la commande suivante :

```
searchNfetch("Chlamydomonas reinhardtii")
```

Finalement, les interactions entre gènes contenus dans ces articles pourront être extraites et affichées sur forme de liste grâce aux lignes de codes suivantes :

```
geneMining()
el <- edgeList()
```

La fonction "geneMining" analysera les documents au format ".txt" présent dans le dossier de travail et ses sous dossiers par groupe de 10,000. Pour chaque lot, les prédictions seront sauvées dans le sous dossier "Results", créé automatiquement dans le dossier de travail durant l'analyse. Ces fichiers de prédictions ".preds" seront ensuite utilisés par la fonction "edgeList" permettant d'afficher les interactions entre gènes sous forme d'une liste FROM / TO / SCORE.

5 Conclusion

Au final, la construction d'une librairie R capable d'extraire des interactions entre gènes dans la littérature scientifique de n'importe quel organisme aura abouti à la construction d'une série de fonctions capables de télécharger automatiquement ces documents, ou du moins leur résumé, sur la base de données PubMed, d'en isoler les différentes phrases et mots avant de les classifier sur base de leurs caractéristiques morphologique et syntaxique afin de retenir uniquement les gènes pour finalement observer leurs relations entre eux.

Bien que ces fonctions ne fournissent pas des résultats aussi optimaux que d'autres méthodes préalablement existantes, elles peuvent tout de même être considérées comme satisfaisantes compte tenu du fait qu'elles n'utilisent aucunes ressources externes spécifiques au type d'organisme analysé. En effet, le but ici n'était pas de surpasser les performances obtenues par ces techniques mais plutôt de fournir un modèle de reconnaissance de gènes applicable à n'importe quel organisme.
Toutefois, cette dernière particularité a le désavantage d'identifier des relations entres gènes de différentes espèces, pour peu que ces deux gènes apparaissent dans la même phrase.

De plus, en augmentant la gamme de nomenclatures reconnues en tant que gène on augmente aussi les chances de faux positifs avec d'autres éléments semblables tel que les noms de souches ou de mutants.
Signalons enfin qu'il est possible que le changement d'environnement des mots de par le style "résumé" des articles téléchargés sur PubMed soit à l'origine d'une perte de précision, l'algorithme de reconnaissance de gènes ayant été construit en analysant des données provenant de documents entiers.

6 Perspectives

Dans un futur proche, les performances de la librairie pourraient être améliorées via la réécriture de certaines parties de code dans le langage “C”, plus efficace pour les différentes étapes impliquant l’utilisation intensive de mémoire vive.
Dans ce même but, l’utilisation d’une librairie “R” telle que “parallel” permettant la prise en charge et la distribution des différents calculs sur plusieurs processeurs permettrait de diminuer le temps nécessaire à l’obtention des prédictions.

Une des étapes limitant le plus la vitesse de traitement des données est celle de l’étiquetage morpho-syntaxique via la librairie “coreNLP”. L’utilisation d’une autre librairie, la réécriture complète de ces fonctions voire le remplacement de cette caractéristique serait recommandée.

Les réseaux inférés pour *Escherichia coli* et *Caenorhabditis elegans* pourront aussi être comparés à différents réseaux créés à l’aide d’autres techniques afin de comparer leurs performances.

Enfin, la précision des différentes prédictions pourrait être améliorée en changeant complètement la manière de représenter les mots sous forme de vecteurs. Dans notre algorithme, chaque mot est représenté par un vecteur dont chaque élément correspond à la valeur d’une caractéristique spécifique. D’autres modèles, tel que “word2vec”[42] par exemple, représentent les mots sous formes de vecteurs de plusieurs centaines de dimensions générés automatiquement en fonction de la tâche recherchée à partir du contexte linguistique de chaque mot via l’utilisation d’un réseau de neurones artificiels.

7 Bibliographie

1. Wood, A.P., Aurikko, J.P., and Kelly, D.P. (2004). *A challenge for 21st century molecular biology and biochemistry: What are the causes of obligate autotrophy and methanotrophy?* FEMS Microbiology Reviews *28*, 335–352.

2. Kitano, H. (2002). *Systems biology : A brief overview. 295*, 1662–4.

3. Eisen, M.B., Spellman, P.T., Brown, P.O., and Botstein, D. (1998). *Cluster analysis and display of genome-wide expression patterns.* Proc Natl Acad Sci USA *95*, 14863–14868.

4. Hecker, M., Goertsches, R.H., Engelmann, R., Thiesen, H.-J., and Guthke, R. (2009). *Integrative modeling of transcriptional regulation in response to antirheumatic therapy.* BMC bioinformatics *10*, 262.

5. Chen, G., Cairelli, M.J., Kilicoglu, H., Shin, D., and Rindflesch, T.C. (2014). *Augmenting Microarray Data with Literature-Based Knowledge to Enhance Gene Regulatory Network Inference.* PLoS Computational Biology *10*.

6. NIH (2016). *MEDLINE/PubMed Resources - Detailed Indexing Statistics: 1965-2016.* Available at: https://www.nlm.nih.gov/bsd/index_stats_comp.html [Accessed August 7, 2017].

7. Sang, E.F.T.K., and De Meulder, F. (2003). *Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition.* CONLL '03 Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 *4*, 142–147. Available at: http://arxiv.org/abs/cs/0306050.

8. Wikipedia (2017). *University of Liège.* Available at: https://en.wikipedia.org/wiki/University_of_Li%C3%A8ge.

9. Qiu, J., Wu, Q., Ding, G., Xu, Y., and Feng, S. (2016). *A survey of machine learning for big data processing.* EURASIP Journal on Advances in Signal Processing *2016*, 67. Available at: https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-016-0355-x.

10. Raschka, S. (2015). *Python Machine Learning* (Packt Publishing).

11. Rudin, C., and L. Wagstaff, K. (2014). *Machine learning for science and society.* Machine Learning *95.1*, 1–9. Available at: http://hdl.handle.net/1721.1/103130.

12. Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python* Available at: http://www.amazon.com/dp/0596516495.

13. Chowdhury, G.G. (2003). *Natural language processing.* Annual Review of Information Science and Technology *37*, 51–89. Available at: http://doi.wiley.com/10.1002/aris.1440370103.

14. D.Liddy, E. (2001). *Natural Language Processing.* Encyclopedia of Library and Information Science.

15. Poibeau, T., and Kosseim, L. (2001). *Proper Name Extraction from Non-Journalistic Texts.* Computa-

tional Linguistics in the Netherlands *44*, 144–157.

16. Field, D., Tiwari, B., Booth, T., Houten, S., Swan, D., Bertrand, N., and Thurston, M. (2006). *Open software for biologists: from famine to feast.* Nature Biotechnology *24*, 801–803. Available at: http://www.nature.com/nbt/journal/v24/n7/full/nbt0706-801.html.

17. Wikipedia (2017). *Unix.* Available at: https://fr.wikipedia.org/wiki/Unix.

18. R Core Team (2017). *R: A language and environment for statistical computing* (Vienna, Austria: R Foundation for Statistical Computing) Available at: https://www.R-project.org/.

19. Arnold, T., and Tilton, L. (2016). *CoreNLP: Wrappers around stanford corenlp tools* Available at: https://CRAN.R-project.org/package=coreNLP.

20. Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2017). *E1071: Misc functions of the department of statistics, probability theory group (formerly: E1071), tu wien* Available at: https://CRAN.R-project.org/package=e1071.

21. Liaw, A., and Wiener, M. (2002). *Classification and regression by randomForest.* R News *2*, 18–22. Available at: http://CRAN.R-project.org/doc/Rnews/.

22. Bouchet-Valat, M. (2014). *SnowballC: Snowball stemmers based on the c libstemmer utf-8 library* Available at: https://CRAN.R-project.org/package=SnowballC.

23. Verspoor, K., Cohen, K.B., Lanfranchi, A., Warner, C., Johnson, H.L., Roeder, C., Choi, J.D., Funk, C., Malenkiy, Y., and Eckert, M. *et al.* (2012). *A corpus of full-text journal articles is a robust evaluation tool for revealing differences in performance of biomedical natural language processing tools.* BMC bioinformatics *13*, 207. Available at: http://www.biomedcentral.com/1471-2105/13/207.

24. Agarwala, R., Barrett, T., Beck, J., Benson, D.A., Bollin, C., Bolton, E., Bourexis, D., Brister, J.R., Bryant, S.H., and Canese, K. *et al.* (2016). *Database resources of the National Center for Biotechnology Information.* Nucleic Acids Research *44*, D7–D19.

25. Manning, C.D. (2009). *Introduction to Information Retrieval - Chapter 2: The termvocabulary and postings lists.* Information Retrieval, 1–18.

26. Breu, F., Guggenbichler, S., and Wollmann, J. (2008). *Developing a robust part-of-speech tagger for biomedical text.* Vasa, 382–392. Available at: http://medcontent.metapress.com/index/A65RM03P4874243N.pdf.

27. Arthur, D., and Vassilvitskii, S. (2007). *K-Means++: the Advantages of Careful Seeding.* Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms *8*, 1027–1025.

28. Manning, C.D., Ragahvan, P., and Schutze, H. (2009). *An Introduction to Information Retrieval - Chapter 16: Flat clustering.* Information Retrieval, 1–27.

29. Huang, Z. (1998). *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical*

Values. Data Mining and Knowledge Discovery *2*, 283–304. Available at: http://link.springer.com/article/10.1023/A:1009769707641.

30. Nadeau, D. (2007). *A survey of named entity recognition and classification*. Linguisticae Investigationes, 3–26. Available at: http://nlp.cs.nyu.edu/sekine/papers/li07.pdf.

31. Ratinov, L., and Roth, D. (2009). *Design challenges and misconceptions in named entity recognition*. Proceedings of the Thirteenth Conference on Computational Natural Language Learning CoNLL 09, 147. Available at: http://portal.acm.org/citation.cfm?doid=1596374.1596399.

32. Rish, I. (2001). *An Empirical Study of the Naïve Bayes Classifier*. IJCAI 2001 workshop on empirical methods in artificial intelligence *3*, 41–46.

33. Manning, C.D., Raghavan, P., and Schütze, H. (2008). *Chapter 15: Support vector machines and machine learning on documents*. Introduction to Information Retrieval, 319–348. Available at: http://www-nlp.stanford.edu/IR-book/.

34. Kazama, J., Makino, T., Ohta, Y., and Tsujii, J. (2002). *Tuning Support Vector Machines for Biomedical Named Entity Recognition*. Proceedings of the Workshop on Natural Language Processing in the Biomedical Domain, 1–8. Available at: http://portal.acm.org/citation.cfm?doid=1118149.1118150.

35. Loh, W.-Y. (2011). *Classification and regression trees*. WIREs Data Mining and Knowledge Discovery *1*.

36. UF CISE (1997). *The ID3 Algorithm*. Available at: https://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm.

37. Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Bayesian Forecasting and Dynamic Models *1*, 1–694.

38. Chen, C., Liaw, A., and Breiman, L. (2004). *Using random forest to learn imbalanced data*. University of California, Berkeley, 1–12.

39. Rodriguez-Esteban, R. (2009). *Biomedical text mining and its applications*. PLoS Computational Biology *5*, 1–5.

40. Wood, R. (1998). *Genetic nomenclature guide*. Elsevier Trends. Available at: http://www.cell.com/pb-assets/journals/trends/GeneticNomenclatureGuide.pdf.

41. Noll, M., Lété, J., and E. Meyer, P. (2017). *Gene Entity Recognition of Full Text Articles* (Singapore: 6th International Conference on Bioinformatics and Biomedical Science).

42. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., Chen, K., Dean, J., Mikolov, T., and Chen, K. (2013). *Distributed Representations of Words and Phrases and their Compositionality*. In NIPS'14, pp. 3111–3119.

8 Annexes

8.1 Scripts R

8.1.1 Assignation et affichage du dossier de travail de la librairie GeneMining

```
# Print current GeneMining working directory
getGMWD <- function(){
  return(get("wd",envir = as.environment("package:GeneMining")))
}

# Set a new GeneMining working directory
setGMWD <- function(path){
  if(is.character(path) == F){stop('"path" argument must be a character string !!!')}
  if(substr(path,nchar(path),nchar(path)) != .Platform$file.sep){
    path <- paste0(path,.Platform$file.sep)
  }
  if(dir.exists(path) == F){
    d <- dir.create(path, recursive = T)
    if(d == F){stop}
  }
  unlockBinding("wd", as.environment("package:GeneMining"))
  assign("wd",path, envir = as.environment("package:GeneMining"))
  lockBinding("wd", as.environment("package:GeneMining"))
  return(path)
}
```

8.1.2 Automatisation des E-utilities Esearch et Efetch

```
searchNfetch <- function(organism){
  # Load info about working directory
  path <- getGMWD()
  # Stop if no working directory set
  if(is.null(path)){
    stop('GeneMining working directory must be set beforehand with function "setGMWD"')
    }

  # esearch
  base <- "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db="
  if(length(strsplit(organism,"MeSH")[[1]])==1){ organism<-paste(organism,"[MeSH+Major+Topic]",sep="") }
  if(length(strsplit(organism," ")[[1]])>1){ organism<-gsub(" ","+",organism) }
  db <- "pubmed"

  url <- paste(base,db,"&term=",organism,"&usehistory=y",sep="")
  res <- readLines(url)

  web <- res[grep("<QueryKey>",res)]
  Count <- gsub(".*<Count>\\s*|</Count>.*", "", web)
  QueryKey <- gsub(".*<QueryKey>\\s*|</QueryKey>.*", "", web)
```

```
  WebEnv <- gsub(".*<WebEnv>\\s*|</WebEnv>.*", "", web)

  if(Count == 0){stop(paste(Count,"hit(s) found for the query :",organism))}
  message(paste(Count,"hit(s) found for the query :",organism))

  # efetch
  base <- "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db="
  rettype <- NULL
  retmode <- "xml"

  url <- paste(base,db,"&rettype=",rettype,"&retmode=",retmode,"&WebEnv=",WebEnv,"&query_key=",QueryKey,sep="")
  xml <- c()
  for(x in 0:(floor(as.integer(Count)/10000))){
    xml <- c(xml,readLines(paste0(url,"&retstart=",x*10000)))
  }


  # pubmed xml parser
  art <- grep("<PubmedArticle>\\s*|</PubmedArticle>", xml)
  res <- lapply(seq(1,length(art),2),function(i){
    article <- xml[art[i]:art[i+1]]
    journal <- gsub(".*<Title>\\s*|</Title>.*", "", article[grep("<Title>",article)])
    title <- gsub(".*<ArticleTitle>\\s*|</ArticleTitle>.*", "", article[grep("<ArticleTitle>",article)])
    abstract <- gsub(".*<AbstractText NlmCategory=\"UNASSIGNED\">\\s*|</AbstractText>.*", "",
                     article[grep("<AbstractText NlmCategory=\"UNASSIGNED\">",article)])
    if(identical(abstract,character(0))){abstract <- gsub(".*<AbstractText>\\s*|</AbstractText>.*", "",
                                                          article[grep("<AbstractText>",article)])}
    pmid <- gsub(".*<ArticleId IdType=\"pubmed\">\\s*|</ArticleId>.*", "",
                 article[grep("<ArticleId IdType=\"pubmed\">",article)])
    doi <- gsub(".*<ArticleId IdType=\"doi\">\\s*|</ArticleId>.*", "",
                article[grep("<ArticleId IdType=\"doi\">",article)])
    return(list(title=title,journal=journal,doi=doi,pmid=pmid,abstract=abstract))
  })
  for(x in res){
    fileConn <- file(paste0(path,"/",x$pmid,".txt"))
    tw <- c(paste("#",x$journal,"#"), paste("#",x$title,"#"), paste("#",x$doi,"#"), x$abstract)
    writeLines(tw, fileConn)
    close(fileConn)
  }
  return(Count)
}
```

8.1.3 Extraction des mots, de leurs caractéristiques et prédiction de leur classe

```
geneMining <- function(){
  # sentence tokenizer
  .sent_tok<-function(text){
    return(strsplit( gsub("(\\.)[ \n\t]+([[:upper:]\"\'\`])","\\1\n\\2",text), "\n" )[[1]])
  }

  # word tokenizer
  .word_tok<-function(sent){
```

```
  words <- strsplit( gsub("([^\\.])\\s","\\1~",sent), "~" )[[1]]
  words <- gsub("^[[:punct:]]+|[[:punct:]]+$","",words)

  # fix ugly number in bracket
  for(i in 1:length(words)){
    word <- words[i]
    if(grepl("\\(",word) && !grepl("\\)",word)){words[i] <- paste0(word,")")}
  }

  return(words)
}

# POS tagger
.extract_POS_feature<-function(data){
  tags<-.extract_POS(data)
  POS<-rep("UNDEF",nrow(data))

  i=1
  z=1
  c=0
  while(i<=nrow(tags) && z<=nrow(data)){
    word<-as.character(tags[i,1])
    tag<-as.character(tags[i,2])

    not_tagged<-data[z,]

    wl <- nchar(word)
    ss <- ceiling(wl/2)

    if (substr(word,1,ss)==substr(not_tagged[,"Word"],1,ss)){
      POS[z]<-tag
      c=0
      ntoken <- length(coreNLP::annotateString(as.character(not_tagged[,"Word"]))$token[,"token"])
      if(ntoken > 1){i=i+ntoken-1}
      z=z+1
    }
    else{
      c=c+1
    }

    # Skip word if no match in a scope of 5 tags
    if(c==5){
      z=z+1
      i=i-5
      c=0
    }
    i=i+1
  }
  POS<-factor(POS,levels = c("","CC", "CD", "DT", "EX", "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD", "NN",
                             "NNS", "NNP", "NNPS", "PDT", "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP",
                             "SYM", "TO", "UH", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ", "WDT", "WP", "WP$",
                             "WRB","UNDEF"))
  POS[is.na(POS)]="UNDEF"
  return(POS)
```

```
}

# Apply the POS tagger sentence by sentence
.extract_POS<-function(data){
  x<-intersect(intersect(which(data[,"Next_1"]==""), which(data[,"Next_2"]==="")), which(data[,"Next_3"]==""))
  ltags<-mapply(function(x,y){
    sentence<-paste(data[(x+1):y,"Word"], collapse = " ")
    sentence<-paste(sentence,".",sep="")
    annotation<-coreNLP::annotateString(sentence)
    tags<-annotation$token[,"POS"]
    tokens<-annotation$token[,"token"]
    return(list("tokens"=tokens,"tags"=tags))
  },c(0,x[1:(length(x)-1)]),x)

  tokens<-as.character(unlist(ltags["tokens",]))
  tags<-as.character(unlist(ltags["tags",]))
  return(data.frame(tokens,tags))
}

# Clustering function
.get_cluster <- function(data,medoids){
  groups <- c()
  closeness <- c()
  for(i in 1:nrow(data)){
    distances <- c()
    for(m in 1:nrow(medoids)){
      distances[m] <- .distance(data[i,],medoids[m,])
    }
    x <- which.min(distances)
    groups[i] <- x
    closeness[i] <- distances[x]
  }
  return(data.frame(data,groups,closeness))
}

# Distance function (clustering)
.distance <- function(token1,token2){
  res <- 0
  for(i in 1:7){
    res <- res + dis[as.character(token1[[i]]),as.character(token2[[i]])]
  }
  return(res/7)
}

# Word pattern extraction function
.get_nom<-function(gene){
  nom<-gsub("[[:upper:]]","A",gene)
  nom<-gsub("[[:lower:]]","a",nom)
  nom<-gsub("[[:digit:]]","0",nom)
  nom<-gsub("[[:punct:]]","-",nom)
  return(nom)
}

# Word abridged pattern extraction function
```

```
.get_abr_nom<-function(pattern){
  abr<-sapply(pattern, function(nom){
    nom<-strsplit(as.character(nom),"")[[1]]
    abr_nom<-nom[1]
    if(length(nom)>=2){
      for(i in 2:length(nom)){
        if(nom[i-1]!=nom[i]){
          abr_nom<-append(abr_nom,nom[i])
        }
      }
    }
    abr_nom<-paste(abr_nom,collapse = "")
    return(abr_nom)
  })
  return(as.character(abr))
}

# Features extraction function
.extractFeatures<-function(data, pattern_levels, abr_pattern_levels, stem_levels, prefix, suffix){
  df<-data
  l_nchar<-sapply(as.character(data[,"Word"]),nchar)

  # Number of uppercase letters
  df[,"NB_Upper"]<-sapply(regmatches(df[,"Word"], gregexpr("[[:upper:]]", df[,"Word"])), length)
  # Ratio of uppercase letters
  df[,"R_Upper"]<-df[,"NB_Upper"]/l_nchar

  # Number of lowercase letters
  df[,"NB_Lower"]<-sapply(regmatches(df[,"Word"], gregexpr("[[:lower:]]", df[,"Word"])), length)
  # Ratio of lowercase letters
  df[,"R_Lower"]<-df[,"NB_Lower"]/l_nchar

  # Number of letters
  df[,"NB_Letter"]<-sapply(regmatches(df[,"Word"], gregexpr("[[:alpha:]]", df[,"Word"])), length)
  # Ratio of letters
  df[,"R_Letter"]<-df[,"NB_Letter"]/l_nchar

  # Number of digits
  df[,"NB_Digit"]<-sapply(regmatches(df[,"Word"], gregexpr("[[:digit:]]", df[,"Word"])), length)
  # Ratio of digits
  df[,"R_Digit"]<-df[,"NB_Digit"]/l_nchar

  # Number of special
  df[,"NB_Special"]<-sapply(regmatches(df[,"Word"], gregexpr("[[:punct:]]", df[,"Word"])), length)
  # Ratio of special
  df[,"R_Special"]<-df[,"NB_Special"]/l_nchar

  # Number of vowel
  df[,"NB_Vowel"]<-sapply(regmatches(tolower(df[,"Word"]), gregexpr("[a,e,i,o,u]", tolower(df[,"Word"]))),
                          length)
  # Ratio of vowel
  df[,"R_Vowel"]<-df[,"NB_Vowel"]/l_nchar

  # Number of consonant
```

```
df[,"NB_Consonant"]<-sapply(regmatches(tolower(df[,"Word"]), gregexpr("[b, c, d, f, g, h, j, k, l, m, n,
                                                                       p, q, r, s, t,v, w, x, y, z]",
                                                                       tolower(df[,"Word"]))), length)
# Ratio of consonant
df[,"R_Consonant"]<-df[,"NB_Consonant"]/l_nchar

# Maximun number of consecutive consonant
df[,"Max_cons_consonant"]<-sapply(gregexpr("[b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, y,
                                           z]+", tolower(df[,"Word"])),function(i){
  return(max(attr(i,"match.length")))
})
# Ratio of maximum consecutive consonant
df[,"R_Max_cons_consonant"]<-df[,"Max_cons_consonant"]/l_nchar

# Number of greek letter
df[,"NB_Greek"]<-sapply(regmatches(df[,"Word"], gregexpr("[ - Ω]", df[,"Word"])), length)
# Ratio of greek letter
df[,"R_Greek"]<-df[,"NB_Greek"]/l_nchar

# Pattern
df[,"Pattern"]<-.get_nom(df[,"Word"])
df[,"Abr_Pattern"]<-.get_abr_nom(df[,"Pattern"])

df[,"Pattern"]<-pattern_levels[df[,"Pattern"]]
df[is.na(df[,"Pattern"]),"Pattern"]<-0

# Abridged pattern
df[,"Abr_Pattern"]<-abr_pattern_levels[df[,"Abr_Pattern"]]
df[is.na(df[,"Abr_Pattern"]),"Abr_Pattern"]<-0

# Prefix
for(i in 1:5){
  f <- paste0("Prefix_",i)
  ps <- tolower(substr(df[,"Word"],1,i))
  df[,f] <- (ps %in% prefix)*1
  df[nchar(ps)!=i,f] <- 0
}

# Suffix
strReverse <- function(x){sapply(lapply(strsplit(as.character(x), NULL), rev), paste,collapse="")}
for(i in 1:5){
  f <- paste0("Suffix_",i)
  ps <- tolower(strReverse(substr(strReverse(df[,"Word"]),1,i)))
  df[,f] <- (ps %in% suffix)*1
  df[nchar(ps)!=i,f] <- 0
}

# End with roman number ?
df[,"End_w_roman"]<-(grepl("[XVI]+$",df[,"Word"]))*1

# Stem
df[,"Stem"]<-SnowballC::wordStem(tolower(df[,"Word"]),"english")
if(is.null(stem_levels)){
  stem_levels<-table(df[,"Stem"])
```

```
  }
  df[,"Stem"]<-stem_levels[df[,"Stem"]]
  df[is.na(df[,"Stem"]),"Stem"]<-0

  return(df)
}

# Gene prediciton function based on random forest build via CRAFT data
.genePredict<-function(test_data, model){
  not_features<-c("Previous_3", "Previous_2", "Previous_1", "Word", "Class", "Next_1", "Next_2", "Next_3")
  z<-test_data[,!colnames(test_data) %in% not_features]

  # Package randomForest does not export predict.randomForest without loading the package
  suppressMessages(require(randomForest))
  pred<-predict(model,z)
  return(data.frame(test_data[,colnames(test_data) %in% not_features], "Prediction"=pred))
}

# Class correction function
.BILOU_correct <- function(data){
  df<-data
  for(i in 1:nrow(df)){
    if(i==1 && df[i,"Prediction"]=="I"){
      if(df[i+1,"Prediction"]=="I" || df[i+1,"Prediction"]=="L"){
        df[i,"Prediction"]<-"B"
      } else {
        df[i,"Prediction"]<-"U"
      }
    } else if (i==nrow(df) && df[i,"Prediction"]=="I"){
      if(df[i-1,"Prediction"]=="B" || df[i-1,"Prediction"]=="I"){
        df[i,"Prediction"]<-"L"
      } else {
        df[i,"Prediction"]<-"U"
      }
    } else if (i>1 && i<nrow(df)){
      if((df[i+1,"Prediction"] %in% c("I","L")) && (df[i-1,"Prediction"] %in% c("O","U","L")) &&
         (df[i,"Prediction"]!="O")){
        df[i,"Prediction"]<-"B"
      }
      if((df[i+1,"Prediction"] %in% c("O","U","B")) && (df[i-1,"Prediction"] %in% c("B","I")) &&
         (df[i,"Prediction"]!="O")){
        df[i,"Prediction"]<-"L"
      }
    }
  }
  for(i in 2:(nrow(df)-1)){
    if(df[i,"Prediction"] %in% c("B", "I") && (df[i+1,"Prediction"] %in% c("O","U","B"))){
      df[i,"Prediction"]<-"U"
    }
    if(df[i,"Prediction"] %in% c("I", "L") && (df[i-1,"Prediction"] %in% c("O","U","L"))){
      df[i,"Prediction"]<-"U"
    }
  }
  return(df)
```

```
}


# Load info about working directory
path <- getGMWD()
# Stop if no working directory set
if(is.null(path)){
  stop('GeneMining working directory must be set beforehand with function "setGMWD"')
}

# load CRAFT data
centroids <- GeneMining::CRAFT_variables$centroids
dis <- GeneMining::CRAFT_variables$dis
pattern_levels <- GeneMining::CRAFT_variables$pattern_levels
abr_pattern_levels <- GeneMining::CRAFT_variables$abr_pattern_levels
stem_levels <- GeneMining::CRAFT_variables$stem_levels
prefix <- GeneMining::CRAFT_variables$prefix
suffix <- GeneMining::CRAFT_variables$suffix
model <- GeneMining::CRAFT_variables$model

# Initialize coreNLP
options(java.parameters = "- Xmx2048m")
coreNLP::initCoreNLP(type="english_fast",mem="2g")

# txt file parser
dirs <- list.dirs(path,F,recursive = F)
dirs <- dirs[dirs!=""]
dirs <- dirs[dirs!="Results"]
if(identical(dirs,character(0))){dirs <- ""}
for(dir in dirs){
  fpath <- paste0(path,dir,.Platform$file.sep)
  files <- list.files(fpath, ".txt$", recursive = T)
  if(length(files) == 0){warning(paste("No text file found in", fpath), noBreaks. = T)}

  message(paste("Analyzing", length(files), "file(s) in ", fpath, "..."))
  pb <- txtProgressBar(min = 0, max = length(files), style = 3,width = 100)
  z <- 0
  ldf <- lapply(files, function(file){
    z <<- z+1
    setTxtProgressBar(pb, z)

    filepath <- paste0(fpath,file)
    art <- readChar(filepath, file.info(filepath)$size)
    art <- gsub("#.*#","",art)
    sentences <- .sent_tok(art)
    sentences <- sentences[sentences!=""]

    if(identical(sentences, character(0))){return(sentences)}

    Previous_3 <- c()
    Previous_2 <- c()
    Previous_1 <- c()
    Word <- c()
    Next_1 <- c()
```

```
  Next_2 <- c()
  Next_3 <- c()
  for(sentence in sentences){
    words <- .word_tok(sentence)
    words <- words[words!=""]
    Word <- c(Word,words)
    Previous_3 <- c(Previous_3,head(c("","","",words),length(words)))
    Previous_2 <- c(Previous_2,head(c("","",words),length(words)))
    Previous_1 <- c(Previous_1,head(c("",words),length(words)))
    Next_1 <- c(Next_1,tail(c(words,""),length(words)))
    Next_2 <- c(Next_2,tail(c(words,"",""),length(words)))
    Next_3 <- c(Next_3,tail(c(words,"","",""),length(words)))
  }

  df <- data.frame(Previous_3,Previous_2,Previous_1,Word,Next_1,Next_2,Next_3)

  # POS tagging
  df[,"POS"] <- .extract_POS_feature(df)

  # cluster based on POS context
  cpos <- as.character(df[,"POS"])
  p1pos <- head(c("_",cpos),length(cpos))
  p1pos[df[,"Previous_1"]==""]<-"_"
  p2pos <- head(c("_","_",cpos),length(cpos))
  p2pos[df[,"Previous_2"]==""]<-"_"
  p3pos <- head(c("_","_","_",cpos),length(cpos))
  p3pos[df[,"Previous_3"]==""]<-"_"
  n1pos <- tail(c(cpos,"_"),length(cpos))
  n1pos[df[,"Next_1"]==""]<-"_"
  n2pos <- tail(c(cpos,"_","_"),length(cpos))
  n2pos[df[,"Next_2"]==""]<-"_"
  n3pos <- tail(c(cpos,"_","_","_"),length(cpos))
  n3pos[df[,"Next_3"]==""]<-"_"
  POS_df <- data.frame(p3pos,p2pos,p1pos,cpos,n1pos,n2pos,n3pos)
  POS_df <- as.matrix(POS_df)
  df[,"Cluster"] <- .get_cluster(POS_df,centroids)$groups
  df[,"Cluster"] <- factor(df[,"Cluster"], levels = 1:nrow(centroids))

  # features extraction
  df <- .extractFeatures(df, pattern_levels, abr_pattern_levels, stem_levels, prefix, suffix)

  # Class prediction
  df <- .genePredict(df,model)

  if(all(unique(df[,"Prediction"]) == "O") == F){
    # Class correction
    df <- .BILOU_correct(df)

    # Group B,I,L and set to "Gene"
    i <- 1
    td <- c()
    df[,"Prediction"] <- as.character(df[,"Prediction"])
    df[,"Word"] <- as.character(df[,"Word"])
    while(i <= nrow(df)){
```

```
          if(df[i,"Prediction"]=="U"){df[i,"Prediction"] <- "Gene"}
          if(df[i,"Prediction"]=="B"){
            start <- i
            while(i < nrow(df) && df[i,"Prediction"]!="L"){i <- i+1}
            end <- i
            df[start,] <- c(df[start,1:3], paste(df[start:end,4], collapse = " "), df[end,5:7],"Gene")
            td <- c(td, (start+1):end)
          }
          i <- i+1
        }
        if(!is.null(td)){df <- df[-td,]}
      }

      # Create result directory
      if(is.na(as.numeric(dir))){dir<-"1"}
      rdir <- paste0(path,"Results",.Platform$file.sep, dir,.Platform$file.sep)
      if(dir.exists(rdir) == F){
        d <- dir.create(rdir, recursive = T)
        if(d == F){stop}
      }
      write.table(df, paste0(path, "Results", .Platform$file.sep, dir, .Platform$file.sep,
                             strsplit(file,"\\.")[[1]][1], ".preds"), sep = "\t", row.names = F, append = F)

      return(T)
    })
    close(pb)

    message(paste("Predictions saved as", paste0(path, "Results", .Platform$file.sep, dir, .Platform$file.sep)))
  }
}
```

8.1.4 Décompte du nombres de documents dans lesquels une interaction est trouvée

```
# Analysis of the number of documents where an interaction is present
.Jaccard<- function(path){
  results <- list.files(path, ".preds$", recursive = T)
  genes <- unique(unlist(sapply(results, function(file){
    predictions <- read.table(paste0(path,file),sep="\t",header = T)
    genes <- unique(as.character(predictions[predictions[,"Prediction"]=="Gene","Word"]))
    return(genes)
  })))

  interactions <- matrix(0,ncol = length(genes), nrow = length(genes))
  colnames(interactions) <- sort(genes)
  rownames(interactions) <- sort(genes)

  for(file in results){
    o_int <- interactions
    predictions <- read.table(paste0(path,file),sep="\t",header = T)
    x <- c(0,which(predictions[,"Next_1"]==""))

    for(i in 1:(length(x)-1)){
```

```
      if(length(x)-1>0){
        sentence <- predictions[(x[i]+1):x[i+1],]
        cgenes <- as.character(sentence[sentence[,"Prediction"]=="Gene", "Word"])
        if(length(cgenes)>1){
          l <- t(combn(cgenes,2))
          for(c in 1:nrow(l)){
            A <- l[c,1]
            B <- l[c,2]
            if(A!=B && interactions[A,B]==o_int[A,B]){
              interactions[A,B] <- interactions[A,B] + 1
              interactions[B,A] <- interactions[B,A] + 1
            }
          }
        }
      }
    }
  }
  no_int <- which(rowSums(interactions)==0)
  interactions <- interactions[-no_int,-no_int]


  # Jaccard index
  tmpg <- colnames(interactions)
  for(x in tmpg){
    tmpg <- tmpg[tmpg != x]
    rsum <- sum(interactions[x,])
    for(y in tmpg){
      csum <- sum(interactions[,y])
      interactions[x,y] <- interactions[x,y]/(rsum+csum-interactions[x,y])
      interactions[y,x] <- interactions[x,y]
    }
  }
  return(interactions)
}

# Return interactions as a FROM/TO dataframe
edgeList <- function(){
  path <- paste0(getGMWD(),"Results",.Platform$file.sep)
  m <- .Jaccard(path)
  from <- c()
  to <- c()
  score <- c()
  for(x in 1:nrow(m)){
    for(y in 1:ncol(m)){
      if(m[x,y]>0){
        from <- c(from, rownames(m)[x])
        to <- c(to, colnames(m)[y])
        score <- c(score, m[x,y])
      }
    }
  }
  df <- data.frame(from, to, score)
  colnames(df) <- c("FROM","TO","SCORE")
  return(df)
```

```
}

# Return interactions as a matrix
edgeMatrix <- function(){
  path <- paste0(getGMWD(),"Results",.Platform$file.sep)
  return(.Jaccard(path))
}

# Return interactions as a FROM/TO dataframe ordered by score
oEdgeList <- function(){
  path <- paste0(getGMWD(),"Results",.Platform$file.sep)
  m <- .Jaccard(path)
  from <- c()
  to <- c()
  score <- c()
  for(x in 1:nrow(m)){
    for(y in 1:ncol(m)){
      if(m[x,y]>0){
        from <- c(from, rownames(m)[x])
        to <- c(to, colnames(m)[y])
        score <- c(score, m[x,y])
      }
    }
  }
  df <- data.frame(from, to, score)
  colnames(df) <- c("FROM","TO","SCORE")
  return(df[order(df[,"SCORE"], decreasing = T),])
}
```

8.1.5 Manuel de la libraire R “GeneMining”

Package 'GeneMining'

August 23, 2017

Type Package

Title Gene Mining Package

Version 0.4.0

Date 2017-08-10

Author LETE Jonathan

Maintainer Manuel Noll <faustfrankenstein@hotmail.de>

Description This package contains useful functions for creating genetic networks based on text mining

License GNU General Public License v2.0 (GPL-2.0)

LazyData TRUE

Imports coreNLP, randomForest, SnowballC

NeedsCompilation no

R topics documented:

`geneInteraction` *Identify gene interactions*

Description

Identify interactions between genes based on their co-occurency in the same sentence. Number of documents where an occurency happens between two genes are normalized using the Jaccard index.

Usage

```
edgeList(predictions)
oEdgeList(predictions)
edgeMatrix(predictions)
```

Details

Predictions will be extracted from the subfolder "Results" in the current GeneMining working directory created by the function "geneMining".

Value

edgeList returns a FROM/TO style dataframe containing the interactions between genes of the same sentence.

oEdgeList returns an ordered FROM/TO style dataframe containing the interactions between genes of the same sentence.

edgeMatrix returns the interactions between genes of the same sentence in a matrix format.

Author(s)

LETE Jonathan

Examples

```
setGMWD("~/GMDOCS")
searchNfetch("Chlamydomonas reinhardtii")
geneMining()

el <- edgeList()
oel <- oEdgeList()
em <- edgeMatrix()
```

---

geneMining *Identify genes in text documents*

Description

Assign values to words based on hand-crafted and statistical rules and compare them with a randomForest model build on the CRAFT corpus in order to identify potential gene names.

Usage

```
geneMining()
```

Details

Words will be extracted from all the documents present in the current GeneMining working directory. Predictions will be saved in the subfolder "Results".

Author(s)

LETE Jonathan

References

http://bionlp-corpora.sourceforge.net/CRAFT/

Examples

```
setGMWD("~/GMDOCS")
searchNfetch("Chlamydomonas reinhardtii")
geneMining()
```

`searchNfetch` *Retrieve PubMed abstract using E-utilities*

Description

Find and stock in the current GeneMining working folder relevant documents for a particular organism in the PubMed Entrez database using ESearch and EFetch E-utilities.

Usage

```
searchNfetch(organism, topics=NULL)
```

Arguments

`organism`	Character. The name of the organism to search documents for.
`topics`	Character. List of other specific Mesh topics to include in order to limit the search

Details

All documents found will be saved under the current GeneMining working directory as their "PubMedID.txt". Informations about the title, the journal and the DOI of each article can be found as commentary within the saved text files. Documents will be grouped in folders of 10,000 docs each.

Be careful to change the GeneMining working direcrory if you don't want to merge multiple search together.

Value

`Count`	The total number of documents retrieved for the organism in the PubMed database

Author(s)

LETE Jonathan

References

https://www.ncbi.nlm.nih.gov/books/NBK25499/

Examples

```
setGMWD("~/GMDOCS")
searchNfetch("Chlamydomonas reinhardtii")
```

setGMWD *Get or Set GeneMining Working Directory*

Description

getGMWD returns the filepath representing the current working directory of GeneMining. setGMWD(path) is used to set the working directory of GeneMining to path.

Usage

```
getGMWD()
setGMWD(path)
```

Arguments

path	Character. Path of the directory to set as GeneMining working directory.

Value

getGMWD returns a character string or NULL if the working directory is not available.

setGMWD returns the current directory after the change. It will give an error if it does not succeed.

Author(s)

LETE Jonathan

Examples

```
setGMWD("~/GMDOCS")
getGMWD()
```