PROJET TUTORE LICENCE INFORMATIQUE

DEVELOPPEMENT D'UNE CARTE DYNAMIQUE EN TYPESCRIPT & INTEGRATION D'ALGORITHMES DE FOUILLE DE DONNEES DANS KNIME

PROJET REALISE PAR :

JIMMY AVAE

ARTHUR ETAIX-BONNIN

JONATHAN MARSOEP

du *15 mai* au *3 juillet 2018*

TUTEURS DE STAGE : - **Nazha SELMAOUI**, MCF HDR

- **Jannai TOKOTOKO**, Doctorant à l'UNC

Table des matières

REMERCIEMENTS

Avant tout, nous souhaitons remercier **Nazha Selmaoui**, **Jannai Tokotoko** et **Grégory Poircuitte** qui nous ont accompagnés pendant ce sixième semestre.

Nous aimerions les remercier de nous avoir transmis leurs connaissances mais aussi d'avoir pris le temps de chercher avec nous des solutions à de nombreux problèmes qui se sont posés pendant toute la durée du projet.

Leur aide tant au niveau technique qu'organisationnel nous a sorti plusieurs fois d'impasses dans notre démarche de recherche.

En somme, sans leur implication il aurait été difficile et beaucoup plus long d'obtenir des résultats satisfaisants.

INTRODUCTION

Ce projet s'est déroulé durant le sixième semestre de Licence Informatique, sous la supervision de **Nazha Selmaoui** (MCF HDR en Informatique de l'ISEA/UNC) ainsi que de **Jannai Tokotoko** (Doctorant en Informatique de l'ISEA/UNC).

Dans le cadre d'un projet de recherche en collaboration entre l'ISEA et l'IFREMER concernant la biodiversité marine. Les chercheurs réalisant des relevés sur les habitats coralliens autour de la Grande-Terre. Pour chaque site de prélèvement (géo-référencé), ils répertorient la localisation du site en latitude et longitude, la date en année, la description du site et des variables environnementales.

L'objectif de ce projet est de montrer l'intérêt des méthodes de fouille de données et leur apport par rapport à des méthodes statistiques classiques. Le travail concerne l'application des méthodes de fouille de données sur ces relevés et la représentation des connaissances extraites spatialement sur une cartographie. Cette cartographie conceptuellement dynamique permet de situer spatialement les connaissances découvertes pour observer leur répartition géographique et éventuellement avoir une description des différents sites.

Ce rapport est la synthèse des connaissances et compétences que nous avons pu acquérir tout au long de ce projet.

D'abord, nous décrirons le cahier des charges ensuite nous présenterons les outils que nous avons utilisés avec leurs avantages et inconvénients. Ensuite, nous présenterons les développements que nous avons effectués lors de ce projet.

Enfin, nous conclurons et donnerons quelques perspectives.

I. DESCRIPTION DU PROJET

Commenté [NS1]: Je vais la rédiger cette partie

Dans le cadre de l'aide à la décision pour la gestion des espaces côtiers et marins, la compréhension et l'évaluation des changements dans la répartition spatiale de la macrofaune face à des facteurs de stress anthropiques et environnementaux sont nécessaires.

Les variations de l'habitat constituent un facteur explicatif essentiel pour l'analyse des tendances spatiales et doivent être prises en compte dans la surveillance et l'évaluation.

Pour cela, des données ont été acquises grâce à des observations vidéo sous-marines sur des sites répartis dans le lagon de la Nouvelle Calédonie. Les sites ont été classés par des typologies d'habitats.

Le but du travail actuel est de décrire ces typologies par les variables environnementales et du milieu qui ont été observées. Les chercheurs utilisent pour cela des méthodes d'extraction de motifs et plus précisément des règles d'association particulières de la forme A1, …, Ap ⇒ typo (classes du milieu) que l'on appelle également des règles de classification.

Cependant, les règles extraites sont souvent textuelles et nombreuses. Une analyse de ces règles une à la fois nécessite une interface de restitution de ces règles conviviales et facile à utiliser pour interpréter les résultats. Pour extraire les règles de classification, les chercheurs utilisent un logiciel, développé par P. Fournier-Viger [référence], appelé SPMF. Ce logiciel est codé en Java, avec plusieurs algorithmes codés. Ces algorithmes fournissent les résultats textuellement.

L'idée est de proposer une interface graphique permettant d'afficher les règles extraites de donner en même temps les sites vérifiant la règle et également visualiser leur position sur la carte pour observer la répartition spatiale des sites par règle de classification en typologie.

Le but de notre projet actuel est à partir d'un jeu de données représenté par une table à deux dimensions (stocké dans un fichier Excel ou CSV) et un fichier texte contenant les règles extraites par l'un des algorithmes de SPMF, faire un post-traitement de ces règles pour donner les sites vérifiant ces règles et pour chaque règle afficher les sites qu'elle couvre.

Les lignes du tableau de données originales représentent les informations d'un site récifal géo-référencé. Les informations qui sont représentées en colonne, décrivent le site : identifiant, position géographique en latitude et longitude, et descriptions du milieu. Enfin, la dernière colonne représente la typologie du milieu correspondant. Le fichier de règles en format texte contient les règles données par ligne et sont écrites sous la forme A1, …, Ap ⇒ typo #SUP 231 #CONF 0.6. #SUP donne le nombre de sites vérifiant la règle et #CONF donne la confiance de la règle.

II. REDEFINITION DU PROJET

Après une semaine de tests et de prise en main du logiciel et des algorithmes de SPMF, nous avons rencontré plusieurs problèmes.

D'une part, les structures de données et les classes créées par Phillipe Fournier-Viger étaient spécifiques à chaque algorithme et pas génériques à toute la bibliothèque d'algorithmes (SPMF) voire à un ensemble d'algorithmes (problème abordé en détails dans la partie 2).

D'autre part, le seul nœud inclus dans KNIME (utilisant OpenStreetMap) pour afficher une carte n'était pas du tout documenté, donc beaucoup trop long à intégrer dans le workflow.

Nous séparons ainsi le workflow en deux et l'exécution des algorithmes se retrouve dans KNIME comme convenu tandis que nous choisissons de créer l'interface intégrant les règles satisfaisantes sur une carte en dehors de KNIME.

Nous découvrons alors l'infrastructure de développement Ionic et un nouveau langage : TypeScript.

III. DESCRIPTION DES OUTILS UTILISES

A. KNIME[3] : un ETL (extract-transform-load) orienté fouille de données

Entièrement codée en JAVA, KNIME repose sur le principe de lier des « nœuds » par leurs entrées et sorties et ainsi former des suites de tâches (workflows).

La dénomination ETL indique en fait que le logiciel permet de récupérer de la donnée, exécuter des prétraitements dessus et charger la donnée transformée dans d'autres structures de données (par exemple dans une base de données extérieure), un autre logiciel fortement répandu est Talend open studio (leader dans l'intégration de données).

Pour en revenir à KNIME, chacun de ces nœuds, qu'il soit codés par l'équipe KNIME ou bien par des contributeurs externes (plateforme public), est en fait une petite application réalisant un ensemble de fonctions (exemple : le nœud input .csv permet de récupérer des données d'un fichier .csv sous la forme d'un tableau, objet JAVA « BufferedDataTable » instancié à partir du constructeur de la classe du même nom).

Les workflows dans KNIME sont manipulables de deux manières.

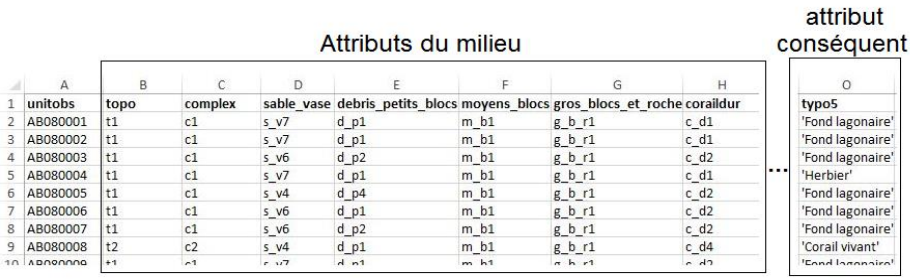
D'une part, l'utilisateur se sert de l'interface graphique de KNIME qui permet de glisser et déposer (drag and drop) les nœuds sur un espace de travail et relier les entrées et sorties des nœuds entre elles.

D’autre part, il est possible, en utilisant des kits de développement logiciel (SDK) fournis par KNIME (en l’occurrence sous Eclipse), de réaliser des nouveaux nœuds sous le langage JAVA.

B. SPMF[1]

Une capture d’écran (annexe 1) nous présente l’interface de SPMF pour l’algorithme ‘TopKclassRules’ dont le fonctionnement technique est présenté plus bas.

Le fichier de départ (CSV ou XLS, annexe 2) doit être dans un premier temps converti en ARFF et les guillemets doubles supprimés (annexe 3). D’après la figure 1, un attribut est associé à une colonne (ex : attribut 1 = topo, attribut 2 = complex, etc) et peut avoir plusieurs modalités différentes (listées entre les accolades {}). Notez ici que le dernier attribut (« typo5 ») correspond aux attributs conséquents, c'est-à-dire que les attributs précédents impliquent forcément une valeur de « typo5 » (Algueraie, Corail vivant etc).



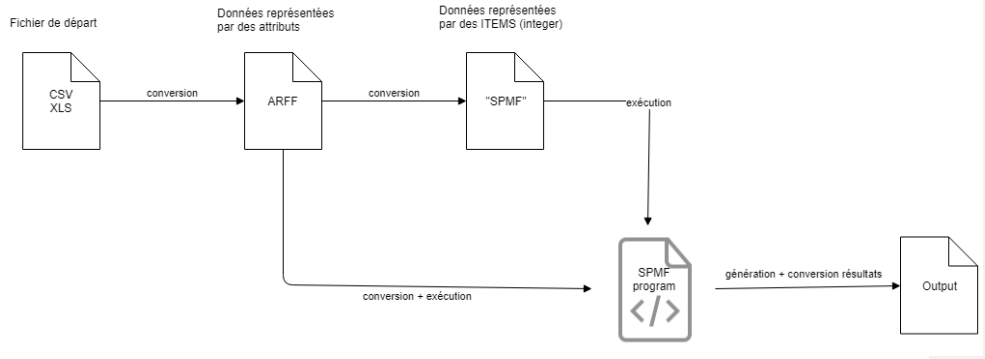
	A	B	C	D	E	F	G	H	O
1	unitobs	topo	complex	sable_vase	debris_petits_blocs	moyens_blocs	gros_blocs_et_roche	corraildur	typo5
2	AB080001	t1	c1	s_v7	d_p1	m_b1	g_b_r1	c_d1	'Fond lagonaire'
3	AB080002	t1	c1	s_v7	d_p1	m_b1	g_b_r1	c_d1	'Fond lagonaire'
4	AB080003	t1	c1	s_v6	d_p2	m_b1	g_b_r1	c_d2	'Fond lagonaire'
5	AB080004	t1	c1	s_v7	d_p1	m_b1	g_b_r1	c_d1	'Herbier'
6	AB080005	t1	c1	s_v4	d_p4	m_b1	g_b_r1	c_d2	'Fond lagonaire'
7	AB080006	t1	c1	s_v6	d_p1	m_b1	g_b_r1	c_d2	'Fond lagonaire'
8	AB080007	t1	c1	s_v6	d_p2	m_b1	g_b_r1	c_d2	'Fond lagonaire'
9	AB080008	t2	c2	s_v4	d_p1	m_b1	g_b_r1	c_d4	'Corail vivant'

Figure 1 : exemple de données sous un format particulier

Il est alors possible d’exécuter l’algorithme avec le fichier (ARFF), celui-ci sera converti automatiquement en format « SPMF » (remplace les attributs en chaine de caractères par des entiers pour optimiser la vitesse de traitement et représente les attributs sous forme « d’items »).

Cependant il est préférable d’exécuter l’algorithme en utilisant un fichier déjà en format « SPMF » afin de connaitre les identifiants (Items ID) des attributs conséquents. Un fichier en format « SPMF » (peut être préalablement converti par le programme) se présente sous la forme suivante : un ITEM (attribut) a un identifiant et est associé à une valeur d’attributs (par exemple, pour les conséquentes ou trouvera @ITEM=14=typo5=’Fond lagonaire’, @ITEM=21=typo5='Herbier', voir annexe 4).

Une fois l'algorithme exécuté on récupère un fichier contenant les règles générées (annexe 5) et quelques informations les concernant (SUP, CONF etc).



C. Ionic[6] Framework pour une interface graphique facile à réaliser

TypeScript[7] est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript). Il permet un typage statique optionnel des variables et des fonctions, la création de classes et d'interfaces, l'import de modules, tout en conservant l'approche non-contraignante de JavaScript. Ce langage est orienté objet donc adapté à l'implémentation d'interface graphique, chaque élément de l'interface peut alors être traité comme un objet facilement manipulable.

Ionic est une infrastructure de développement (framework) open-source, basé initialement sur AngularJS et Apache Cordova. Ionic permet de créer un code multi support en utilisant des outils Web comme HTML, CSS, JavaScript, afin de générer des applications iOS, Android, Chrome, Windows Phone et bien d'autres. Il nous permet surtout d'importer des bibliothèques de cartographie telle que Leaflet.

IV. INTEGRATION DES DONNES & D'ALGORITHMES DANS KNIME

A. Fonctionnement technique de SPMF

La figure 2 présente l'architecture de SPMF sans rentrer dans les détails d'exécution d'un algorithme en particulier. Il faut savoir que pour des raisons d'optimisation, les attributs donnés en entrée sous format de chaines de caractères sont convertis en entiers. Au moment de cette conversion, une variable JAVA Map<Integer, String> est créée pour faire correspondre ces entiers aux chaines de caractères. Cette variable est ensuite utilisée une fois les résultats générés pour les retransformer en chaines de caractères.

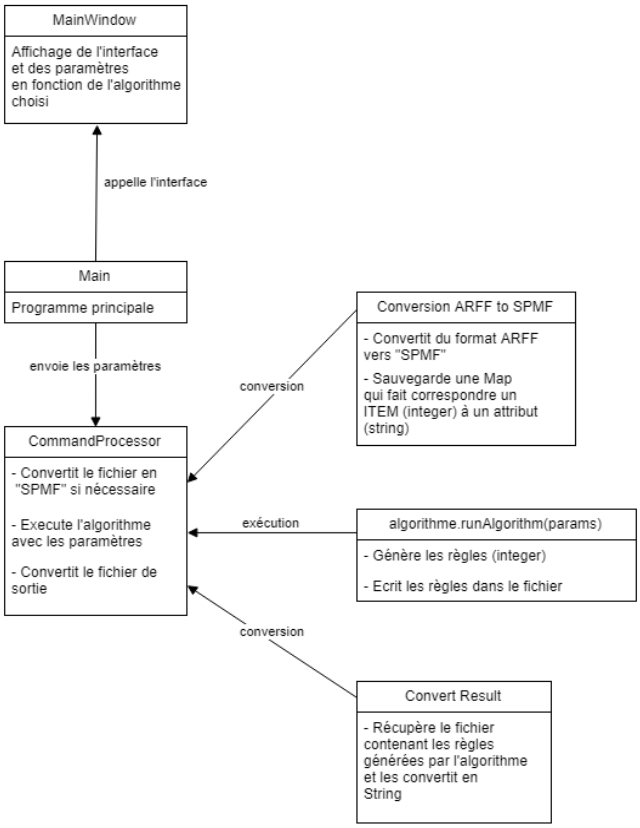


Figure 2 : Architecture de SPMF

D'après la figure 3, le résultat de l'exécution de l'algorithme TopKClassRules et ClosedClassAssociationRules sont inscris dans un fichier txt comme suit :

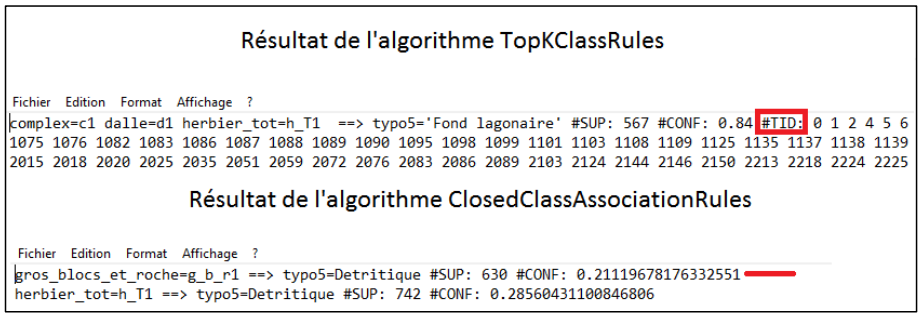
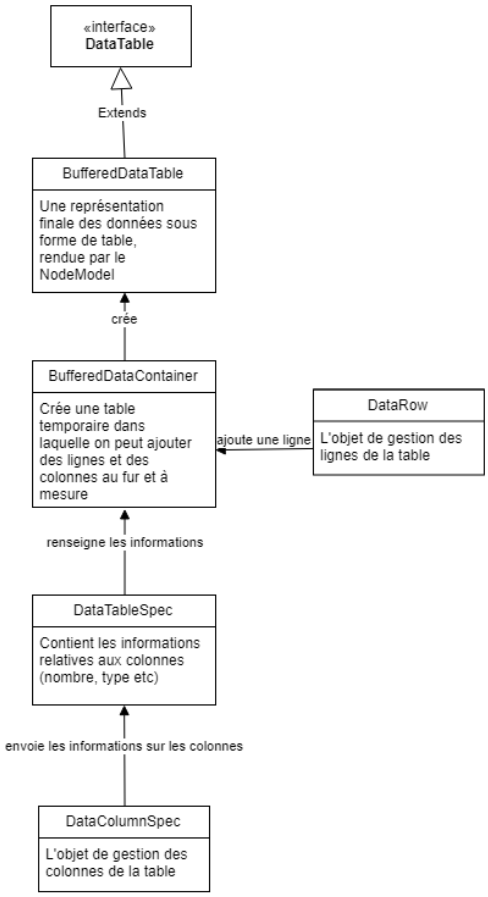


Figure 3 : Résultats des algorithmes

Afin d'exploiter ces résultats et de les afficher sur une carte, nous proposons dans la suite de lire le fichier résultant ou de récupérer directement le résultat en sortie des méthodes JAVA. Afin d'identifier les relevés géo-localisés qui vérifient les règles d'association obtenues, l'algorithme TopKClassRules fournit un nouvel identifiant par relevé. Cet identifiant est directement corrélé à l'identifiant du fichier source suivant son indice de ligne. L'identifiant sera noté TIDs dans la suite. Les TIDs sont donc les sites de relevé qui sont les instances positives des règles. Dans l'algorithme ClosedClassAssociationRules les TIDs ne sont pas renseignés.

B. Intégration de l'algorithme TopKClassRules dans KNIME

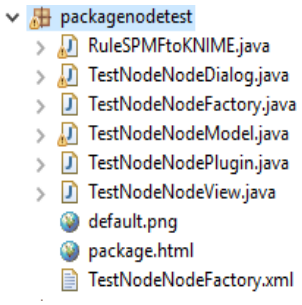
Nous avons développé un nœud KNIME qui réalise l'exécution et l'injection des résultats de l'algorithme « TopKClassRules » dans KNIME sans passer par les fichiers de résultats (il est plus judicieux de garder les résultats au sein de KNIME sans les écrire sur le disque). Nos données seront stockées dans une table spécifique à KNIME (BufferedDataTable), sa création et son remplissage sont présentés par le schéma ci-dessous.

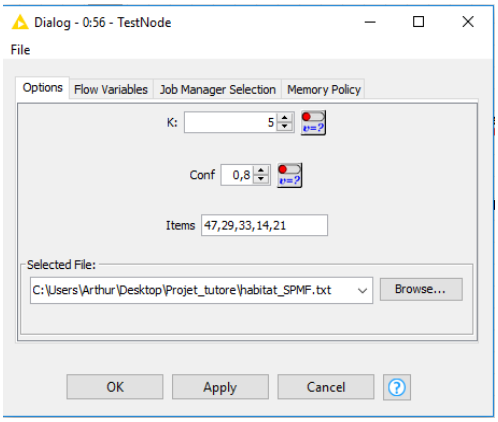


Au sein de l'algorithme TopK, le calcul des TIDs se fait en utilisant un type JAVA BitSet. Un Bitset est une représentation de 0 à x bits. A sa création un Bitset est « vide », tous ses bits sont à 0 (false) et le Bitset ne contient aucune valeur. Quand on passe la valeur d'un bit 'y' à 1 (true), la valeur 'y' apparait alors dans le Bitset. Ainsi, pour chaque règle, si le relevé (ligne) du fichier de départ vérifie cette règle, on passe le bit correspondant à la règle à la valeur vraie.

Le package de notre nœud se présente de la manière suivante. Nous allons nous intéresser seulement à 3 classes :

- RuleSPMFtoKNIME : formatage des résultats de l'algorithme en type KNIME
- NodeDialog : interface de configuration du nœud
- NodeModel : code à exécuter au déclenchement du nœud





L'interface générée par le NodeDialog se présente en figure 4. On retrouve plusieurs champs déjà observés sur l'interface SPMF (k, Minconf etc). Notez que les 2 icones à côté des champs 'k' et 'Conf' permettent de lier la valeur de ces champs à une variable de Workflow KNIME (utile dans les boucles par exemple).

Pour chaque champ déclaré dans le NodeDialog (annexe 6), on associe une variable dans le NodeModel (annexe 7), dont on pourra alors récupérer la valeur.

Figure 4 : interface NodeDialog

Dans le NodeModel, on instancie un objet TopKClassRules (annexe 8). On crée un objet Database (type SPMF optimisé notamment pour l'algorithme TopK). Ensuite, on formate les Items conséquents en tableau d'entiers, dans un objet Map qui nous servira à convertir le résultat (Integer -> String) une fois l'algorithme exécuté. Le nœud exécute alors l'algorithme et injecte les résultats dans notre objet RuleSPMFtoKNIME qui va s'occuper du formatage des données en type de données bufferedDataTable de KNIME (on envoie les règles générées, un booléen pour générer les TIDs et la Map).

Pour la réalisation du format de données en sortie de knime on instancie alors un objet RuleSPMFtoKNIME dont la fonction convertResults() rend une ou plusieurs BufferedDataTable . Nous proposons deux sorties différentes.

1. Première sortie

Notre premier affichage (voir annexe 9) se présentera avec 4 colonnes : une pour la règle, une pour le SUP (nombre de TIDs générés), une pour la confiance CONF et une pour la liste des TIDs qui vérifient la règle (cf. Figure 5). Une règle sera représentée par une ligne. Notre méthode fonctionne de la manière suivante : pour chaque règle générée par l'algorithme, nous allons ajouter une ligne au tableau avec les champs nécessaires. Nous préparons également une liste avec les TIDs qui nous servira à traiter le deuxième affichage (annexe 10).

Vous remarquez que sur le résultat obtenu ci-après les attributs de la classe sont codés en entiers. En effet, l'algorithme a convertit les attributs (au départ en chaine de caractères) en entier pour des soucis d'optimisation. La conversion inverse pour récupérer les valeurs en chaine de caractères ne se fait pas automatiquement dans l'algorithme TopKClassRules. On se charge alors dans notre méthode d'effectuer cette conversion (inspirée d'une méthode déjà crée dans SPMF).

Figure 5 : Tableau en sortie d'algorithme

Out-Port name - 0:56 - TestNode

File Hilite Navigation View

Table "default" - Rows: 5 | Spec - Columns: 4 | Properties | Flow Variables

Row ID	Rules	SUP	CONF	TID
0	1 2 8 11 ==> 14	499	0.844	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 30, 32, 35, 41,
1	2 5 6 8 11 ==> 14	547	0.843	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 30, 32, 35, 41,
2	2 5 8 11 ==> 14	556	0.844	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 30, 32, 35, 41,
3	2 6 8 11 ==> 14	556	0.839	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 30, 32, 35, 41,
4	2 8 11 ==> 14	567	0.84	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26, 27, 30, 32, 35, 41,

Après application de la méthode de conversion (annexe 11) on obtient le résultat montré en Figure 6. On arrive bien à récupérer l'affichage des règles sous forme de chaine de caractères, comme si on exécutait l'algorithme en passant par l'interface SPMF.

Figure 6 : Tableau en sortie d'algorithme avec conversion

Out-Port name - 0:56 - TestNode

File Hilite Navigation View

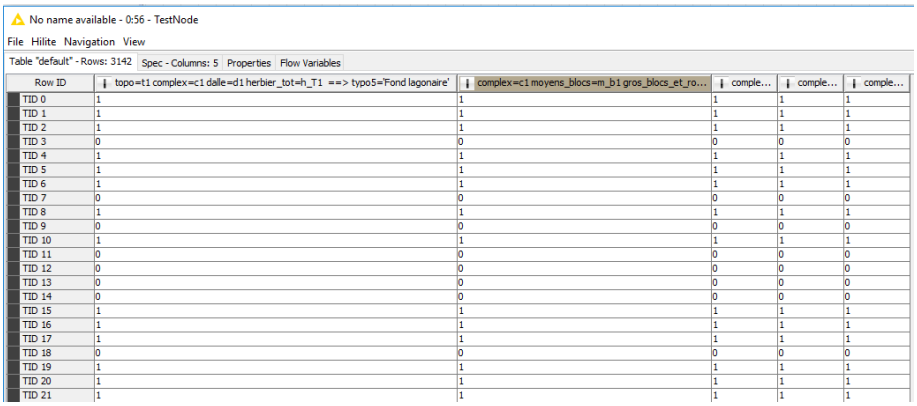
Table "default" - Rows: 5 | Spec - Columns: 4 | Properties | Flow Variables

Row ID	Rules	SUP	CONF	TID
0	topo=t1 complex=c1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	499	0.844	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23
1	complex=c1 moyens_blocs=m_b1 gros_blocs_et_roche=g_b_r1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	547	0.843	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23
2	complex=c1 moyens_blocs=m_b1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	556	0.844	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23
3	complex=c1 gros_blocs_et_roche=g_b_r1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	556	0.839	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23
4	complex=c1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	567	0.84	0, 1, 2, 4, 5, 6, 8, 10, 15, 16, 17, 19, 20, 21, 22, 23

2. Deuxième sortie

Notre deuxième sortie consiste à afficher les règles en colonnes et les TIDs en ligne (allant de 0 à 3141), ainsi pour chaque TID on aura un booléen (à 0 ou 1) selon que le TID vérifie ou non la règle. Nous allons cette fois compter de 0 à 3141 et pour chaque règle, si on trouve l'indice de la ligne courante égal à une valeur de TID contenue dans la liste de la règle

en cours, alors on met la valeur de la cellule à 1, 0 sinon (annexe 12). On trouve alors une table plus facile à traiter par notre carte (voir Figure 7).



No name available - 0:56 - TestNode

File Hilite Navigation View

Table "default" - Rows: 3142 | Spec - Columns: 5 | Properties | Flow Variables

Row ID	topo=t1 complex=c1 dalle=d1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'	complex=c1 moyens_blocs=m_b1 gros_blocs_et_ro...	comple...	comple...	comple...
TID 0	1	1	1	1	1
TID 1	1	1	1	1	1
TID 2	1	1	1	1	1
TID 3	0	0	0	0	0
TID 4	1	1	1	1	1
TID 5	1	1	1	1	1
TID 6	1	1	1	1	1
TID 7	0	0	0	0	0
TID 8	1	1	1	1	1
TID 9	0	0	0	0	0
TID 10	1	1	1	1	1
TID 11	0	0	0	0	0
TID 12	0	0	0	0	0
TID 13	0	0	0	0	0
TID 14	0	0	0	0	0
TID 15	1	1	1	1	1
TID 16	1	1	1	1	1
TID 17	1	1	1	1	1
TID 18	0	0	0	0	0
TID 19	1	1	1	1	1
TID 20	1	1	1	1	1
TID 21	1	1	1	1	1

Figure 7

Enfin, notre Workflow KNIME présente une boucle qui exécute plusieurs fois notre nœud de traitement et pour chaque itération fait varier la confiance et génère un nouveau fichier CSV (voir Figure 8).

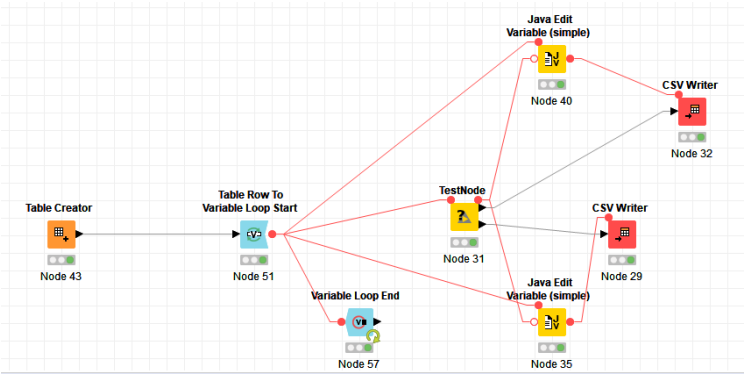
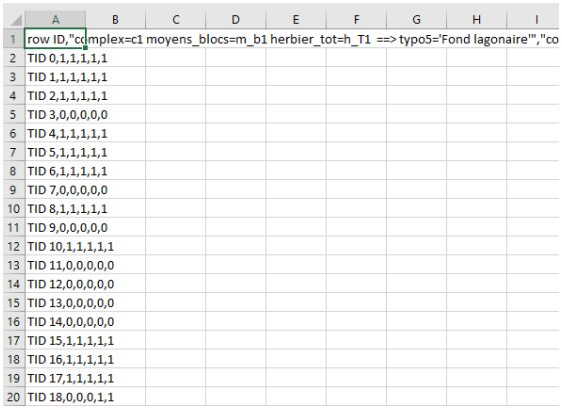


Figure 8 : Workflow avec boucle



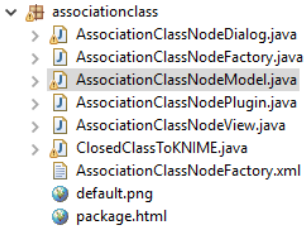
	A	B	C	D	E	F	G	H	I
1	row ID,"complex=c1 moyens_blocs=m_b1 herbier_tot=h_T1 ==> typo5='Fond lagonaire'","co								
2	TID 0,1,1,1,1,1								
3	TID 1,1,1,1,1,1								
4	TID 2,1,1,1,1,1								
5	TID 3,0,0,0,0,0								
6	TID 4,1,1,1,1,1								
7	TID 5,1,1,1,1,1								
8	TID 6,1,1,1,1,1								
9	TID 7,0,0,0,0,0								
10	TID 8,1,1,1,1,1								
11	TID 9,0,0,0,0,0								
12	TID 10,1,1,1,1,1								
13	TID 11,0,0,0,0,0								
14	TID 12,0,0,0,0,0								
15	TID 13,0,0,0,0,0								
16	TID 14,0,0,0,0,0								
17	TID 15,1,1,1,1,1								
18	TID 16,1,1,1,1,1								
19	TID 17,1,1,1,1,1								
20	TID 18,0,0,0,1,1								

L'affichage de la figure 9 (fichier CSV) avec les TIDs en ligne nous intéresse plus particulièrement pour la carte.

Figure 9

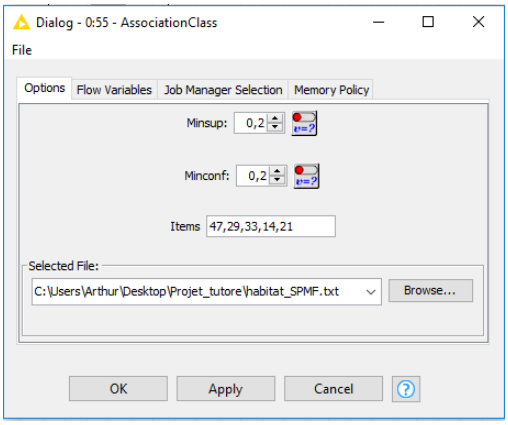
C. Intégration de l'algorithme ClosedClassAssociationRules dans KNIME

Le deuxième algorithme à intégrer dans KNIME est l'algorithme « ClosedClasAssociationRules ». L'absence de documentation SPMF sur cet algorithme a rendu son utilisation plus compliquée. On utilisera la même méthode d'implémentation que pour l'algorithme précédent. Le résultat sera rendu dans une table KNIME (BufferedDataTable).



Notre package de nœud (voir Figure 10) ressemble à celui de l'algorithme TopK. Le formatage du résultat se fera dans la classe « ClosedClassToKNIME ».

Figure 10



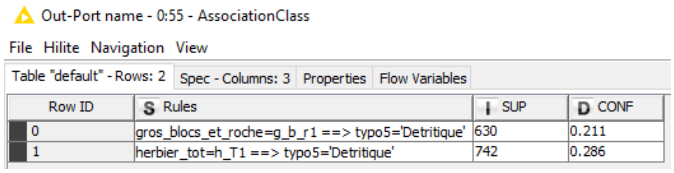
L'interface générée par le NodeDialog (voir Figure 11), on retrouve des champs de configuration adaptés aux paramètres de l'algorithme ClosedClassAssociationRule.

Comme pour TopK, on associe les champs dans le NodeDialog (annexe 13) à des variables dans le NodeModel (annexe 14).

Figure 11

On prépare dans le NodeModel l'exécution de l'algorithme (annexe 15) comme pour TopK. Comme vous pouvez le constater sur le code, on exécute 2 algorithmes à la suite, il faut savoir que l'algorithme ClosedClassAssociationRules est un dérivé d'un autre algorithme (ClosedAssociationRules) mais avec une méthode de traitement plus fine. Une fois les règles générées, on les transmet à notre classe chargée du formatage des données en table KNIME.

On affichera les résultats d'une manière similaire à TopK mais cette fois sans colonne pour les TIDs (l'algorithme ClosedClassAssociationRules ne les génére pas). On trouve alors le résultat montré en Figure 12.


Out-Port name - 0:55 - AssociationClass

File Hilite Navigation View

Table "default" - Rows: 2 | Spec - Columns: 3 | Properties | Flow Variables

Row ID	Rules	SUP	CONF
0	gros_blocs_et_roche=g_b_r1 ==> typo5='Detritique'	630	0.211
1	herbier_tot=h_T1 ==> typo5='Detritique'	742	0.286

Figure 12

Finalement, notre objectif final était de générer nous-mêmes les TIDs grâce au fichier de données de départ. Par manque de temps, nous somme seulement en mesure de présenter un schéma théorique de récupération de ces TIDs.

Nous ne sommes pas censé modifier le code des algorithmes, il nous est alors impossible de générer les TIDs comme avec TopK (dont les TIDs sont générés pendant la génération résultats). Notre méthode aurait consisté à comparer les règles générées avec le fichier d'entrée pour trouver nous-mêmes les lignes (TIDs) respectant les règles.

Prenons l'exemple suivant :

Règle 1: att1 att2 att3 => att4
Règle 2: att1 att2 att6 => att4
Règle 3: att5 => att10

Voici le traitement théorique que nous pouvons appliquer :

Pour chaque règle :

Pour chaque attribut (de la partie de la règle) :

Pour chaque ligne dans le fichier de départ :

Vérifier si l'attribut est respecté ou non par la ligne courant

Fin

Fin

Fin

On soulève ici un problème évident de complexité en temps. Pour un grand nombre de règles, elles-mêmes formées par un grand nombre d'attributs et avec un fichier de départ composé

de plusieurs milliers de lignes (ou plus), le temps d'exécution risquerait d'être extrêmement long. Même sur des machines modernes, nous avons observé un ralentissement de KNIME allant jusqu'au plantage complet de l'application, dû à un traitement sur des bases de données trop grosses.

V. DEVELOPPEMENT D'UNE CARTE AVEC IONIC FRAMEWORK ET LEAFLET

A. Créer une application avec Ionic

Avec Ionic, la première chose à faire est d'installer Node.js.

Node.js est une plateforme logicielle libre en JavaScript. Concrètement, c'est un environnement bas niveau permettant l'exécution de JavaScript côté serveur.

Ensuite, il faut installer Ionic à l'aide de NPM[5].

NPM est le gestionnaire de paquets officiel de Node.js. Depuis la version 0.6.3 de Node.js, il fait partie de l'environnement et est donc automatiquement installé par défaut. NPM fonctionne avec un terminal et gère les dépendances pour une application. Il permet également d'installer des applications Node.js disponibles sur le dépôt NPM.

Les commandes d'installation de paquets sont de la forme suivante :

« npm install –g [paquet à installer] »

Enfin, reste à créer un projet Ionic (générer un répertoire contenant l'application) grâce à la commande :

« ionic start [nom du projet] [type d'interface de base souhaitée*] »

*Par exemple, nous avons choisi l'interface blank (« vide » : projet à compléter).

L'architecture de l'application est fortement basée sur celle d'AngularJS.

On trouve d'abord 2 répertoires principaux :

« node_modules » dans lequel se trouve tous les paquets installés ;

« src » dans lequel on trouve le contenu de l'application,

Dans ce-dernier, le développeur ne travaille presque que sur le dossier « pages ». Le dossier « assets » contient des fichiers utiles comme des images ou encore des pages de CSS. On trouve aussi un répertoire « app » sur lequel nous reviendrons plus tard.

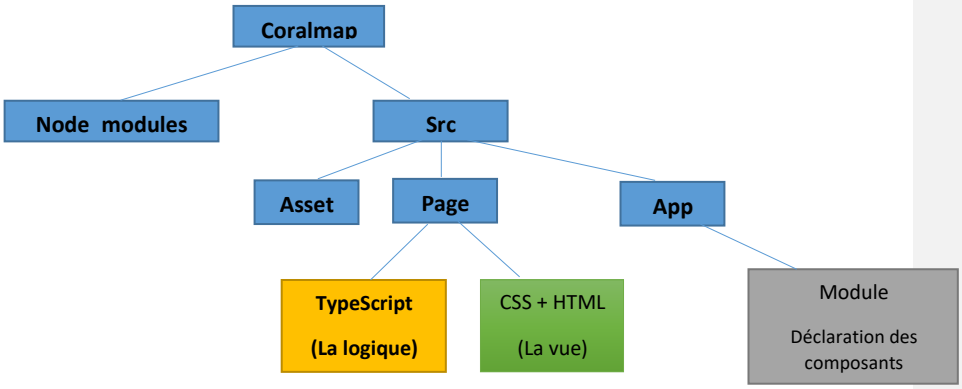
Le répertoire « pages » contient les pages qui composent l'application comme une application Web. Chaque « page » est associée à un fichier .html, un fichier .scss (Syntactically

Awesome Stylesheets, langage compilé en CSS qui intègre des concepts manquants en CSS comme les variables) et un fichier .ts (Typescript, langage compilé en Javascript).

D'abord, le fichier HTML reprend toutes les balises usuelles avec « ion- » en préfixe.

Ensuite, le fichier SCSS contient le style des objets de la page HTML.

Enfin, le fichier TypeScript contient la logique du fichier HTML homonyme. Le schéma ci-dessous représente l'architecture.



B. Leaflet[9], bibliothèque JavaScript pour des cartes interactives

Leaflet est une bibliothèque JavaScript libre de cartographie en ligne. Elle est notamment utilisée par le projet de cartographie libre et ouverte OpenStreetMap.

Comme tous les autre paquets utilisés par l'application, Leaflet nécessite d'être installée dans le projet concerné grâce à la commande :

« npm install leaflet »

Dans l'API, la classe « map » est centrale, elle instancie l'objet qui contient tout ce qui compose la carte.

```
L.map(<String> id, <Map options>options?)
```

Exemple :

```
var map = L.map('map', {
    center: [51.505, -0.09],
    zoom: 13
});
```

On ajoute ensuite à cet objet des couches (Layers) de plusieurs sortes comme les marqueurs (markers).

Exemple :

```
L.marker([50.5, 30.5]).addTo(map);
```

Ou encore des ensembles d'images formant des cartes (TileLayer).

Exemple :

```
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
      attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">
      OpenStreetMap</a> contributors'
}).addTo(mymap);
```

Figure 13 : Exemple de map auquel on ajoute le tilelayer

C. Coralmap, une cartographie des règles d'association

D'après la Figure 14, notre application (coralmap), devait répondre à plusieurs critères :

1. Permettre à l'utilisateur d'importer un fichier .csv contenant des règles d'associations vu précédemment ainsi que les sites sur lesquels ces règles s'appliquent, ainsi qu'un fichier .csv contenant tous les sites de relevés et leur géolocalisation (longitude et latitude).
2. Afficher sur une carte de la Nouvelle-Calédonie des marqueurs associés à chaque site présents dans le fichier importé.
3. Chaque site doit avoir une couleur correspondant à la combinaison de règles qu'il confirme.

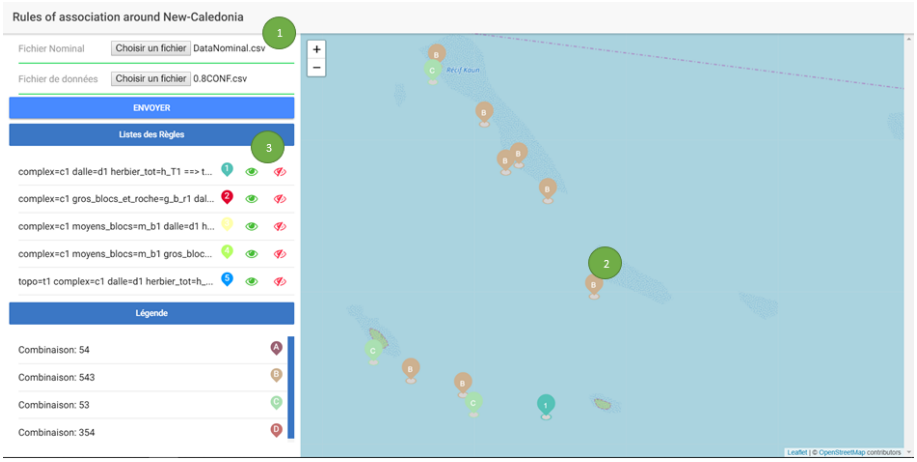


Figure 14 : Interface de coralmap

Expliquons d'abord la récupération des fichiers et la mise en variable de leurs données :

- On importe un paquet «npm install file-reader » qui permet d'utiliser des fonctions telle que .onload() qui lit le fichier. Ces fonctions sont asynchrones, on les a ainsi synchronisé entre elles pour mettre les données dans des tableaux.

Ensuite, l'affichage de chaque site sur la carte :

- On fait appel aux tableaux instanciés précédemment pour créer des marqueurs représentants les sites présents dans le fichier de règles (Tids).

Enfin, le choix des couleurs des marqueurs et de la légende :

- A chaque nouvelle règle, on attribue une couleur, lorsqu'un site vérifie plusieurs règles, on utilise un paquet «npm install color-mixer » que l'on importe afin qu'il mélange les couleurs des règles pour lesquelles le site est positif. Chaque combinaison de règle, en plus d'avoir une couleur unique, est associée à une lettre.

CONCLUSION

Pour ce projet, nous avons créé une carte des sites selon des règles d'association à partir d'un fichier csv qui contenait des relevés d'habitats coralliens géo-référencés. Pour cela, nous avons intégré une bibliothèque de méthodes d'extraction de motifs dans la plateforme libre dédiée à l'analyse de données (KNIME). Cette plateforme propose de réaliser des suites de tâches dans lesquelles nous souhaitions intégrer puis formater les dits relevés, les traiter avec les méthodes d'extractions et enfin les afficher dans une carte. Nous avons formaté les résultats des algorithmes, étant donné que le format de données imposé par KNIME et le format de données en sortie des algorithmes de la bibliothèque SPMF sont spécifiques à chacun des environnements.

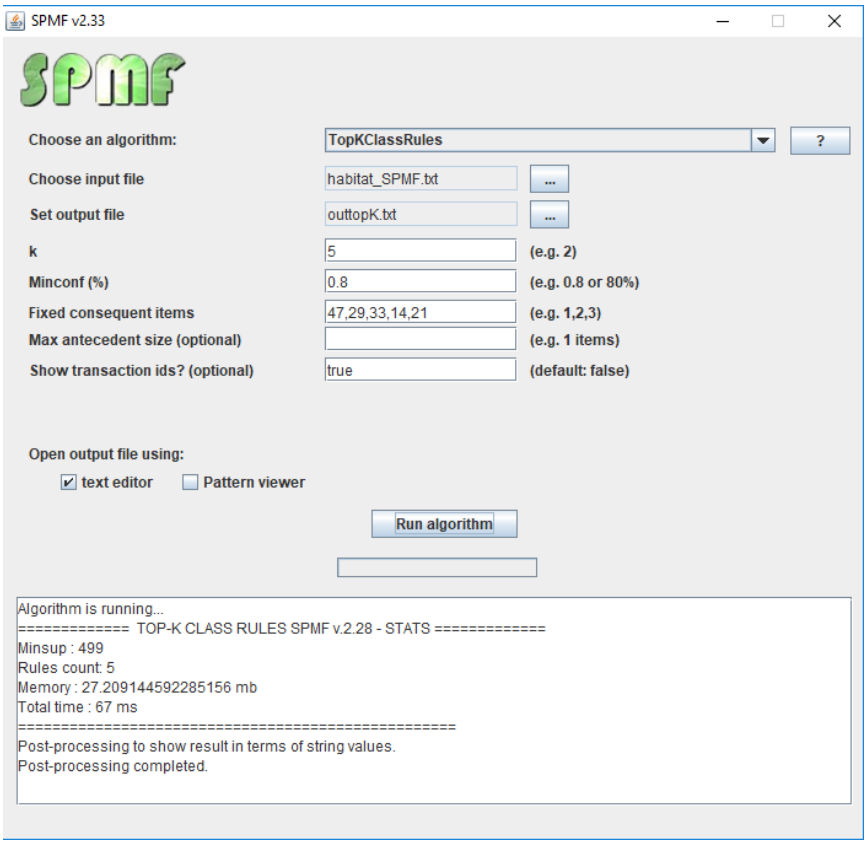
Nous avons utilisé Ionic, un framework en TypeScript, afin de créer une interface graphique permettant de récupérer un fichier contenant des règles d'associations ainsi que les sites qui les vérifient. Cette interface cartographie les sites de relevés en fonction des règles récupérées qu'ils vérifient.

En perspective de ce projet, il serait intéressant d'intégrer toutes ces tâches dans un seul environnement afin d'automatiser la génération des règles d'association ainsi que leur cartographie.

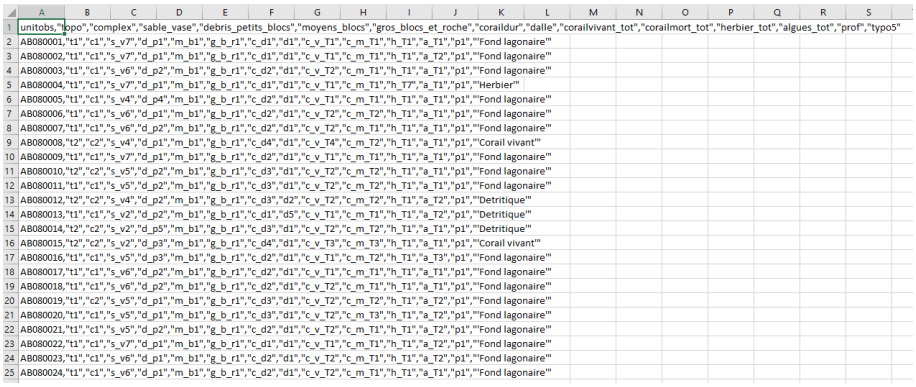
REFERENCES

1. http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php
2. https://www.knime.com/developers
3. https://www.knime.com/documentation
4. https://www.knime.com/docs/api/overview-summary.html
5. https://www.npmjs.com/
6. https://ionicframework.com/docs/
7. https://www.typescriptlang.org/docs/home.html
8. https://angular.io/docs
9. https://leafletjs.com/reference-1.3.2.html

ANNEXES



Annexe 1 : interface SPMF



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	unitobs,"topo","complex","sable_vase","debris_petits_blocs","moyens_blocs","gros_blocs_et_roche","corraildur","dalle","corailvivant_tot","corailmort_tot","herbier_tot","algues_tot","prof","typo5"																		
2	AB080001,"t1","c1","s_v7","d_p1","m_b1","g_b_r1","c_d1","d1","c_v_T1","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
3	AB080002,"t1","c1","s_v7","d_p1","m_b1","g_b_r1","c_d1","d1","c_v_T1","c_m_T1","h_T1","a_T2","p1",""Fond lagonaire""																		
4	AB080003,"t1","c1","s_v6","d_p2","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
5	AB080004,"t1","c1","s_v7","d_p1","m_b1","g_b_r1","c_d1","d1","c_v_T1","c_m_T1","h_T7","a_T1","p1",""Herbier""																		
6	AB080005,"t1","c1","s_v4","d_p4","m_b1","g_b_r1","c_d2","d1","c_v_T1","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
7	AB080006,"t1","c1","s_v6","d_p1","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T2","h_T1","a_T1","p1",""Fond lagonaire""																		
8	AB080007,"t1","c1","s_v6","d_p2","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
9	AB080008,"t2","c2","s_v4","d_p1","m_b1","g_b_r1","c_d4","d1","c_v_T4","c_m_T2","h_T1","a_T1","p1",""Corail vivant""																		
10	AB080009,"t1","c1","s_v7","d_p1","m_b1","g_b_r1","c_d2","d1","c_v_T1","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
11	AB080010,"t2","c2","s_v5","d_p2","m_b1","g_b_r1","c_d3","d1","c_v_T2","c_m_T2","h_T1","a_T1","p1",""Fond lagonaire""																		
12	AB080011,"t1","c1","s_v5","d_p2","m_b1","g_b_r1","c_d3","d1","c_v_T2","c_m_T2","h_T1","a_T1","p1",""Fond lagonaire""																		
13	AB080012,"t2","c2","s_v4","d_p2","m_b1","g_b_r1","c_d3","d2","c_v_T2","c_m_T2","h_T1","a_T2","p1",""Detritique""																		
14	AB080013,"t1","c1","s_v2","d_p2","m_b1","g_b_r1","c_d1","d5","c_v_T1","c_m_T1","h_T1","a_T2","p1",""Detritique""																		
15	AB080014,"t2","c2","s_v2","d_p5","m_b1","g_b_r1","c_d3","d1","c_v_T2","c_m_T2","h_T1","a_T1","p1",""Detritique""																		
16	AB080015,"t2","c2","s_v2","d_p3","m_b1","g_b_r1","c_d4","d1","c_v_T3","c_m_T3","h_T1","a_T1","p1",""Corail vivant""																		
17	AB080016,"t1","c1","s_v5","d_p3","m_b1","g_b_r1","c_d2","d1","c_v_T1","c_m_T2","h_T1","a_T3","p1",""Fond lagonaire""																		
18	AB080017,"t1","c1","s_v6","d_p2","m_b1","g_b_r1","c_d2","d1","c_v_T1","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
19	AB080018,"t1","c1","s_v6","d_p2","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		
20	AB080019,"t1","c2","s_v5","d_p1","m_b1","g_b_r1","c_d3","d1","c_v_T2","c_m_T2","h_T1","a_T2","p1",""Fond lagonaire""																		
21	AB080020,"t1","c1","s_v5","d_p1","m_b1","g_b_r1","c_d3","d1","c_v_T2","c_m_T3","h_T1","a_T2","p1",""Fond lagonaire""																		
22	AB080021,"t1","c1","s_v5","d_p2","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T2","p1",""Fond lagonaire""																		
23	AB080022,"t1","c1","s_v7","d_p1","m_b1","g_b_r1","c_d1","d1","c_v_T1","c_m_T1","h_T1","a_T2","p1",""Fond lagonaire""																		
24	AB080023,"t1","c1","s_v6","d_p1","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T2","p1",""Fond lagonaire""																		
25	AB080024,"t1","c1","s_v6","d_p1","m_b1","g_b_r1","c_d2","d1","c_v_T2","c_m_T1","h_T1","a_T1","p1",""Fond lagonaire""																		

Annexe 2 : fichier de départ en CSV

```
@RELATION DataNominals

@ATTRIBUTE topo {t1,t2,t3,t4}
@ATTRIBUTE complex {c1,c2,c3,c4}
@ATTRIBUTE sable_vase {s_v1,s_v2,s_v3,s_v4,s_v5,s_v6,s_v7}
@ATTRIBUTE debris_petits_blocs {d_p1,d_p2,d_p3,d_p4,d_p5,d_p6,d_p7}
@ATTRIBUTE moyens_blocs {m_b1,m_b2,m_b3,m_b4,m_b5,m_b6}
@ATTRIBUTE gros_blocs_et_roche {g_b_r1,g_b_r2,g_b_r3,g_b_r4,g_b_r5,g_b_r7}
@ATTRIBUTE corraildur {c_d1,c_d2,c_d3,c_d4,c_d5,c_d6,c_d7}
@ATTRIBUTE dalle {d1,d2,d3,d4,d5,d6}
@ATTRIBUTE corailvivant_tot {c_v_T1,c_v_T2,c_v_T3,c_v_T4,c_v_T5,c_v_T6,c_v_T7}
@ATTRIBUTE corailmort_tot {c_m_T1,c_m_T2,c_m_T3,c_m_T4,c_m_T5,c_m_T6,c_m_T7}
@ATTRIBUTE herbier_tot {h_T1,h_T2,h_T3,h_T4,h_T5,h_T6,h_T7}
@ATTRIBUTE algues_tot {a_T1,a_T2,a_T3,a_T4,a_T5,a_T6,a_T7}
@ATTRIBUTE prof {p1,p2,p3,p4,p5,p6}
@ATTRIBUTE typo5 {'Algueraie','Corail vivant','Detritique','Fond lagonaire','Herbier'}

@DATA
t1,c1,s_v7,d_p1,m_b1,g_b_r1,c_d1,d1,c_v_T1,c_m_T1,h_T1,a_T1,p1,'Fond lagonaire'
t1,c1,s_v7,d_p1,m_b1,g_b_r1,c_d1,d1,c_v_T1,c_m_T1,h_T1,a_T2,p1,'Fond lagonaire'
t1,c1,s_v6,d_p2,m_b1,g_b_r1,c_d2,d1,c_v_T2,c_m_T1,h_T1,a_T1,p1,'Fond lagonaire'
t1,c1,s_v7,d_p1,m_b1,g_b_r1,c_d1,d1,c_v_T1,c_m_T1,h_T7,a_T1,p1,'Herbier'
t1,c1,s_v4,d_p4,m_b1,g_b_r1,c_d2,d1,c_v_T1,c_m_T1,h_T1,a_T1,p1,'Fond lagonaire'
t1,c1,s_v6,d_p1,m_b1,g_b_r1,c_d2,d1,c_v_T2,c_m_T2,h_T1,a_T1,p1,'Fond lagonaire'
t1,c1,s_v6,d_p2,m_b1,g_b_r1,c_d2,d1,c_v_T2,c_m_T1,h_T1,a_T1,p1,'Fond lagonaire'
t2,c2,s_v4,d_p1,m_b1,g_b_r1,c_d4,d1,c_v_T4,c_m_T2,h_T1,a_T1,p1,'Corail vivant'
t1,c1,s_v7,d_p1,m_b1,g_b_r1,c_d2,d1,c_v_T1,c_m_T1,h_T1,a_T1,p1,'Fond lagonaire'
t2,c2,s_v5,d_p2,m_b1,g_b_r1,c_d3,d1,c_v_T2,c_m_T2,h_T1,a_T1,p1,'Fond lagonaire'
t1,c1,s_v5,d_p2,m_b1,g_b_r1,c_d3,d1,c_v_T2,c_m_T2,h_T1,a_T1,p1,'Fond lagonaire'
t2,c2,s_v4,d_p2,m_b1,g_b_r1,c_d3,d2,c_v_T2,c_m_T2,h_T1,a_T2,p1,'Detritique'
```

Annexe 3 : fichier converti en ARFF

input_converted.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
@ITEM=1=topo=t1@ITEM=2=complex=c1@ITEM=3=sable_vase=s_v7@ITEM=4=debris_petits_blocs=d_p1@ITEM
@ITEM=15=algues_tot=a_T21 2 3 4 5 6 7 8 9 10 11 13 14 15
@ITEM=16=sable_vase=s_v6@ITEM=17=debris_petits_blocs=d_p2@ITEM=18=coraildur=c_d2@ITEM=19=cora
@ITEM=20=herbier_tot=h_T7@ITEM=21=typo5='Herbier'1 2 3 4 5 6 7 8 9 10 12 13 20 21
@ITEM=22=sable_vase=s_v4@ITEM=23=debris_petits_blocs=d_p41 2 5 6 8 9 10 11 12 13 14 18 22 23
@ITEM=24=corailmort_tot=c_m_T21 2 4 5 6 8 11 12 13 14 16 18 19 24
1 2 5 6 8 10 11 12 13 14 16 17 18 19
@ITEM=25=topo=t2@ITEM=26=complex=c2@ITEM=27=coraildur=c_d4@ITEM=28=corailvivant_tot=c_v_T4@IT
1 2 3 4 5 6 8 9 10 11 12 13 14 18
@ITEM=30=sable_vase=s_v5@ITEM=31=coraildur=c_d35 6 8 11 12 13 14 17 19 24 25 26 30 31
1 2 5 6 8 11 12 13 14 17 19 24 30 31
@ITEM=32=dalle=d2@ITEM=33=typo5='Detritique'5 6 11 13 15 17 19 22 24 25 26 31 32 33
@ITEM=34=sable_vase=s_v2@ITEM=35=dalle=d51 2 5 6 7 9 10 11 13 15 17 33 34 35
@ITEM=36=debris_petits_blocs=d_p55 6 8 11 12 13 19 24 25 26 31 33 34 36
@ITEM=37=debris_petits_blocs=d_p3@ITEM=38=corailvivant_tot=c_v_T3@ITEM=39=corailmort_tot=c_m_
@ITEM=40=algues_tot=a_T31 2 5 6 8 9 11 13 14 18 24 30 37 40
1 2 5 6 8 9 10 11 12 13 14 16 17 18
1 2 5 6 8 10 11 12 13 14 16 17 18 19
1 4 5 6 8 11 13 14 15 19 24 26 30 31
1 2 4 5 6 8 11 13 14 15 19 30 31 39
1 2 5 6 8 10 11 13 14 15 17 18 19 30
```

Annexe 4 : fichier en format « SPMF »

outtopK.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
topo=t1 complex=c1 dalle=d1 herbier_tot=h_T1  ==> typo5='Fond lagonaire' #SUP: 499 #CONF: 0.8443316412859561 #TID: 0 1 2 4 5 6 8 10 15
1099 1103 1108 1109 1135 1137 1138 1139 1152 1153 1154 1156 1157 1168 1169 1170 1180 1181 1182 1188 1189 1191 1192 1193 1194 1196 1198
2446 2452 2462 2470 2473 2486 2489 2491 2492 2500 2501 2502 2504 2505 2510 2523 2525 2531 2534 2535 2539 2540 2542 2544 2545 2546 2548
complex=c1 moyens_blocs=m_b1 gros_blocs_et_roche=g_b_r1 dalle=d1 herbier_tot=h_T1  ==> typo5='Fond lagonaire' #SUP: 547 #CONF: 0.84283
1071 1073 1074 1075 1076 1082 1083 1086 1087 1088 1089 1090 1095 1098 1099 1101 1103 1108 1109 1125 1135 1137 1139 1143 1144 1152 1153
2020 2025 2035 2051 2059 2072 2076 2083 2086 2089 2103 2124 2144 2146 2150 2213 2218 2224 2225 2231 2236 2238 2242 2243 2244 2249 2250
complex=c1 gros_blocs_et_roche=g_b_r1 dalle=d1 herbier_tot=h_T1  ==> typo5='Fond lagonaire' #SUP: 556 #CONF: 0.8386123680241327 #TID:
071 1073 1074 1075 1076 1082 1083 1086 1087 1088 1089 1090 1095 1098 1099 1101 1103 1108 1109 1125 1135 1137 1139 1143 1144 1152 1153
2012 2015 2018 2020 2025 2035 2051 2059 2072 2076 2083 2086 2089 2103 2124 2144 2146 2150 2213 2218 2224 2225 2231 2236 2238 2242 2243
complex=c1 moyens_blocs=m_b1 dalle=d1 herbier_tot=h_T1  ==> typo5='Fond lagonaire' #SUP: 556 #CONF: 0.8437025796661608 #TID: 0 1 2 4 5
73 1074 1075 1076 1082 1083 1086 1087 1088 1089 1090 1095 1098 1099 1101 1103 1108 1109 1125 1135 1137 1138 1139 1143 1144 1152 1153 1
018 2020 2025 2035 2051 2059 2072 2076 2083 2086 2089 2103 2124 2144 2146 2150 2213 2218 2224 2225 2231 2236 2238 2242 2243 2244 2249
complex=c1 dalle=d1 herbier_tot=h_T1  ==> typo5='Fond lagonaire' #SUP: 567 #CONF: 0.84 #TID: 0 1 2 4 5 6 8 10 15 16 17 19 20 21 22 23
1075 1076 1082 1083 1086 1087 1088 1089 1090 1095 1098 1099 1101 1103 1108 1109 1125 1135 1137 1138 1139 1143 1144 1152 1153 1154 1156
2015 2018 2020 2025 2035 2051 2059 2072 2076 2083 2086 2089 2103 2124 2144 2146 2150 2213 2218 2224 2225 2231 2236 2238 2242 2243 2244
```

Annexe 5 : fichier de sortie de SPMF, algorithme TopKClassRules

```
30 public class TestNodeNodeDialog extends DefaultNodeSettingsPane {
31
32     /**
33      * New pane for configuring TestNode node dialog.
34      * This is just a suggestion to demonstrate possible default dialog
35      * components.
36      */
37     protected TestNodeNodeDialog() {
38         super();
39
40         addDialogComponent(new DialogComponentNumber(TestNodeNodeModel.kRules, "K:", 1, 10, this.createFlowVariableModel("k", FlowVariable.Type.INTEGER)));
41
42         addDialogComponent(new DialogComponentNumber(TestNodeNodeModel.conf, "Conf", 0.01, this.createFlowVariableModel("conf", FlowVariable.Type.DOUBLE)));
43
44         addDialogComponent(new DialogComponentString(TestNodeNodeModel.items, "Items"));
45
46         addDialogComponent(new DialogComponentFileChooser(TestNodeNodeModel.file, "File", JFileChooser.OPEN_DIALOG, false));
47
48     }
49 }
```

Annexe 6 : code KNIME pour le NodeDialog

```
56    static SettingsModelInteger kRules = new SettingsModelInteger("k", 5);
57    static SettingsModelDouble conf = new SettingsModelDouble("conf", 0.8);
58    static SettingsModelString items = new SettingsModelString("items","47,29,33,14,21");
59    static SettingsModelString file = new SettingsModelString("file", "");
```

Annexe 7 : lien entre champs du NodeDialog et NodeModel (TopK)

```
protected BufferedDataTable[] execute(final BufferedDataTable[] inData,
        final ExecutionContext exec) throws Exception {

    //int k = 5;
    //double minConf = 0.8;
    //int[] itemToBeUsedAsConsequent = new int[]{47,29,33,14,21};

    RuleSPMFtoKNIME rstk = new RuleSPMFtoKNIME(exec);

    Database database = new Database();
    database.loadFile(file.getStringValue());

    int[] itemToBeUsedAsConsequent = Arrays.asList(items.getStringValue().split(",")).stream().mapToInt(Integer::parseInt).toArray();

    TransactionDatabaseConverter converter = new TransactionDatabaseConverter();
    Map<Integer, String> mapping = converter.convertARFFandReturnMap("C:/Users/Arthur/Desktop/Projet_tutore/DataNominals.arff", "C:/Us

    AlgoTopKClassRules algo = new AlgoTopKClassRules();
    algo.runAlgorithm(kRules.getIntValue(), conf.getDoubleValue(), database, itemToBeUsedAsConsequent);
    //algo.setOutputTransactionIdentifiers(true);
    //algo.writeResultTofile("C:/Users/Arthur/Desktop/Projet_tutore/asuppr.txt");

    //System.out.println(conf.getDoubleValue());

    //ResultConverter converter2 = new ResultConverter();
    //converter2.convert(mapping, "C:/Users/Arthur/Desktop/Projet_tutore/asuppr.txt", "C:/Users/Arthur/Desktop/Projet_tutore/asuppr2.t

    return rstk.convertResults(algo.getkRules(), algo.isOutputTransactionIdentifiers(), mapping);
```

Annexe 8 : code exécuté dans le NodeModel (TopK)

```
37        DataColumnSpec[] colOut= new DataColumnSpec[4];
38        colOut[0] = new DataColumnSpecCreator("Rules", StringCell.TYPE).createSpec();
39        colOut[1] = new DataColumnSpecCreator("SUP", IntCell.TYPE).createSpec();
40        colOut[2] = new DataColumnSpecCreator("CONF", DoubleCell.TYPE).createSpec();
41        colOut[3] = new DataColumnSpecCreator("TID", StringCell.TYPE).createSpec();
```

Annexe 9 : déclaration des colonnes de la table en sortie KNIME

```
cellsOut[0]=new StringCell(writer.toString());
cellsOut[1]=new IntCell(rule.getAbsoluteSupport());
cellsOut[2]=new DoubleCell(rule.getConfidence());

//Add TID to list
int compteur = 0;
int i = rule.common.nextSetBit(0);
ArrayList<IntCell> listTID = new ArrayList<IntCell>();
while ( compteur < 3142 ){
    if( compteur == i){
        listTID.add(new IntCell(i));
        tidString.append(i);
        tidString.append(", ");
        i = rule.common.nextSetBit(i + 1);
    }
    compteur ++;
}
//Add list of TID to list
listlistTID.add(listTID);

//CellsOut[3] = ListKnime of TID
//ArrayList<IntCell> cellValues = listlistTID.get(count);
//resultCellsOut[count] = CollectionCellFactory.createListCell(cellValues);
//cellsOut[3] = resultCellsOut[count];
cellsOut[3] = new StringCell(tidString.toString());
```

Annexe 10 : remplissage ligne par ligne pour chaque règle générée (contenu dans une boucle « for each rule »)

```
68          String tempRule = rule.toString();
69          String [] split = tempRule.split(" ");
70          boolean noItemsLeft = false;
71          StringBuilder writer = new StringBuilder();
72          for(int i=0; i< split.length; i++){
73              String token = split[i];
74              // if the character "#" is met, it means that the rest of the line
75              // does not contains items, we note it and we write the token directly
76              // to the file as well as all the following tokens.
77              if(token.startsWith("#") || noItemsLeft){
78                  noItemsLeft = true;
79                  // write the token as it is to the output file
80                  writer.append(token);
81              }else{
82                  // check if the current token is a positive integer >0.
83                  // if so, we will convert to its string representation
84                  Integer itemID = isInteger(token);
85                  if(itemID == null){
86                      // SPECIAL CASE
87                      // THE FOLLOWING CODE HAS BEEN ADDED SPECIALLY TO HANDLE SEQUENTIAL RULES
88                      // OF THE FORM 1,2 --> 5,6
89                      // If token contains a "," such as "2,5,6", then we will separate it
90                      // by "," and transform it
91                      // to the string representation and write it to file
92                      if(token.indexOf(',') >=0){
93                          // We split the parts by ","
94                          String[] parts = token.split(",");
95                          for(int m = 0; m < parts.length; m++){
96                              // We write the string representation of this item
97                              Integer item = Integer.parseInt(parts[m]);
98                              String stringRepresentation = mapItemIDtoStringValue.get(item);
99                              writer.append(stringRepresentation);
100                             // if not the last item, we will put back the "," to separate
101                             // this item from the next
102                             if(m < parts.length-1){
103                                 writer.append(",");
104                             }
105                         }
106                     }
107                     // END OF SPECIAL CASE
108                     else{
109                         // write the token as it is to the output file
110                         writer.append(token);
111                     }
112
113                 }else{
114                     // convert the item to its string and write it to the output file
115                     String name = mapItemIDtoStringValue.get(itemID);
116                     if(name == null){
117                         // if the number has no corresponding name
118                         writer.append(Integer.toString(itemID));
119                     }else{
120                         // if the number has a name
121                         writer.append(mapItemIDtoStringValue.get(itemID));
122                     }
123                 }
124             }
125
126             // if not the last item, we write a space to the output file
127             if(i != split.length-1){
128                 writer.append(" ");
129             }
130         }
```

Annexe 11 : méthode pour faire correspondre les attributs entiers en attributs chaines de caractères

```
182         for(int l=0;l < 3142;l ++){
183             RowKey keyOut2 = new RowKey("TID "+l);
184             int k = 0;
185             // Allocate memory of cells per line out2
186             DataCell[] cellsOut2 = new DataCell[kRules.size()];
187 
188             for (Object ruleObj : rules) {
189                 ClassRuleG rule = (ClassRuleG) ruleObj;
190                 if(listlistTID.get(k).contains(new IntCell(l))){
191                     cellsOut2[k] = new IntCell(1);
192                 }else{
193                     cellsOut2[k] = new IntCell(0);
194                 }
195                 k ++;
196             }
197 
198             DataRow rowOut2 = new DefaultRow(keyOut2, cellsOut2);
199             container2.addRowToTable(rowOut2);
200         }
```

Annexe 12 : génération des résultats avec les TIDs en ligne

```
30  protected AssociationClassNodeDialog() {
31      super();
32 
33      addDialogComponent(new DialogComponentNumber(AssociationClassNodeModel.minsup, "Minsup:", 0.01, this.createFlowVariableModel("minsup", FlowVariable.Type.DOUBLE)));
34      addDialogComponent(new DialogComponentNumber(AssociationClassNodeModel.minconf, "Minconf:", 0.01, this.createFlowVariableModel("minconf", FlowVariable.Type.DOUBLE)));
35      addDialogComponent(new DialogComponentString(AssociationClassNodeModel.items, "Items"));
36      addDialogComponent(new DialogComponentFileChooser(AssociationClassNodeModel.file, "File", JFileChooser.OPEN_DIALOG, false));
37      
38  }
```

Annexe 13 : configuration des champs du NodeDialog

```
56      static SettingsModelDouble minsup = new SettingsModelDouble("sup", 0.2);
57      static SettingsModelDouble minconf = new SettingsModelDouble("conf", 0.2);
58      static SettingsModelString items = new SettingsModelString("items","47,29,33,14,21");
59      static SettingsModelString file = new SettingsModelString("file", "");
```

Annexe 14 : lien entre champs du NodeDialog et NodeModel (ClosedClass)

```
protected BufferedDataTable[] execute(final BufferedDataTable[] inData,
        final ExecutionContext exec) throws Exception {

    int[] itemToBeUsedAsConsequent = Arrays.asList(items.getStringValue().split(",")).stream().mapToInt(Integer::parseInt).toArray();

    // Loading the transaction database
    TransactionDatabase database = new TransactionDatabase();
    try {
        database.loadFile(file.getStringValue());
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    AlgoFPClose algo = new AlgoFPClose();
    // Run the algorithm
    // Note that here we use "null" as output file path because we want to keep the results into memory instead of saving to a file
    Itemsets patterns = algo.runAlgorithm(file.getStringValue(), null, minsup.getDoubleValue());

    // STEP 2: Generate all rules from the set of frequent itemsets (based on Agrawal & Srikant, 94)
    AlgoClosedClassRules_UsingFPClose algoClosedRules = new AlgoClosedClassRules_UsingFPClose();
    AssocRules rules = algoClosedRules.runAlgorithm(patterns, null, database.size(), minconf.getDoubleValue(), algo.cfiTree, itemToBeUsedAsConsequent);

    TransactionDatabaseConverter converter = new TransactionDatabaseConverter();
    Map<Integer, String> mapping = converter.convertARFFandReturnMap("C:/Users/Arthur/Desktop/Projet_tutore/DataNominals.arff", "C:/Users/Arthur/Desktop

    ClosedClassToKNIME cctk = new ClosedClassToKNIME(exec);

    return cctk.convertResults(rules.getRules(), mapping);

}
```

Annexe 15 : code exécuté dans le NodeModel(ClosedClass)