

UNIVERSITE LIBRE DE BRUXELLES

Faculté des Sciences

Département d'Informatique

**Apprentissage inductif
appliqué à la conception
d'un éditeur intelligent de
“workflows”
bio-informatiques**

Mémoire présenté en vue
de l'obtention du grade de
Licencié en Informatique
par Sangwo Yves

Directeur de mémoire
M. Bersini Hugues

Année Académique 2003 - 2004

Table des matières

Table des figures	iv
Liste des tableaux	v
Remerciements	i
Introduction	1
1 Théorie de l'apprentissage	3
1.1 Apprentissage inductif	4
1.2 Arbres de décision	6
1.2.1 Principe de la méthode	6
1.2.2 Algorithme C4.5	7
1.3 Espaces des versions	11
1.3.1 Principe	11
1.3.2 Algorithme CEA	11
1.3.3 Traitement du bruit	12
1.4 Raisonnement à base des cas	14
1.4.1 Principe	14
1.4.2 Systèmes CBR	14
1.4.3 Modèles CBR	16
1.4.4 Performances	18
1.5 Induction de règles	18
1.5.1 Principe	18
1.5.2 Performances	19
1.6 Approche combinatoire	19
1.6.1 Description de RISE	20
1.6.2 Les règles et les instances	20
1.6.3 Mesure de similarité	21
1.6.4 Prédiction et apprentissage	21
1.7 Classifieur Naïf de Bayes	22
1.7.1 Théorie	22
1.7.2 Performances	23
1.8 Apprentissage des habitudes	23
1.8.1 Définitions	23
1.8.2 Principe	24
1.9 Conclusion	25

2	BIGRE	26
2.1	Le projet BIGRE	27
2.1.1	Présentation	27
2.1.2	Objectifs	27
2.2	Le système BIGRE	28
2.2.1	Architecture générale	28
2.2.2	Notion de <i>Plug-ins</i>	32
2.2.3	IHM du système	33
2.2.4	Introduction du Workflow	33
2.3	Interfaces Intelligentes	36
2.3.1	Définitions et objectifs	36
2.3.2	Workflow Intelligent	37
2.4	Réalisation du WI	43
2.4.1	Profil de l'utilisateur	44
2.4.2	Knowledge Data Base	46
2.4.3	Historique des actions	48
2.4.4	Choix algorithmiques	50
2.5	Cahier des charges	51
2.6	Conclusion	53
3	Etude des performances	55
3.1	Données	56
3.1.1	Description des données	56
3.1.2	Description des workflows	58
3.1.3	Ensembles d'apprentissage	59
3.1.4	Hypothèses	59
3.1.5	Améliorations possibles	62
3.2	Méthodologie	63
3.2.1	Expérimentations	64
3.2.2	Weka	68
3.3	Résultats	70
3.3.1	Cas de base	70
3.3.2	Découpe en groupes	73
3.3.3	Approche <i>IKDB</i>	77
3.4	Conclusion	78
	Conclusion	80
	Annexe A	82
	Annexe B	91
	Références	106

Table des figures

1.1	Arbre construit par CALM pour l'ensemble des littéraux $\{a,b,c,d\}$	13
1.2	Modèle générique d'un système CBR	15
1.3	Processus de prédiction	24
2.1	Architecture générale de BIGRE.	29
2.2	Diagramme de séquence général pour l'utilisation d'un service.	31
2.3	Diagramme de séquence détaillé pour l'utilisation d'un service.	32
2.4	L'IHM du service Web Blast.	34
2.5	IHM schématique du Workflow	39
2.6	IHM schématique amélioré du Workflow.	41
2.7	Interaction entre les agents et le processus principal	42
2.8	Expert Workflow	47
2.9	Construction de la IKDB	48
2.10	Utilisation de l'historique des actions de l'utilisateur	49
3.1	Choix possibles des configurations des expériences à l'étape de pré-traitement des données (à gauche) et lors de l'analyse des groupes (à droite).	67
3.2	Evaluation du taux de prédiction en fonction du nombre de voisins sur l'ensemble des workflows disponibles. Les règles sont constituées de deux variables.	71
3.3	Evaluation du taux de prédiction en fonction du nombre de voisins sur l'ensemble des workflows disponibles. Les règles sont constituées de trois variables.	72
3.4	Cas de base. Les instances sont constituées de trois variables.	72
3.5	IIQ de C4.5 avec et sans élagage. Instances de deux variables.	73
3.6	Tailles des groupes isolées.	74
3.7	Taux de prédiction pour différents groupes avec/sans doublons.	75
3.8	IIQ dans le cas de <code>biofor</code> sans élimination des doublons.	76
3.9	Validation croisée au niveau des workflows appliquée au groupe <code>biofor</code> .	77

Liste des tableaux

1.1	Procédure récursive de construction de l'arbre de décision de C4.5	7
2.1	Le cahier des charges	52
3.1	Echantillon des comptes utilisateurs repris dans les <i>logs</i> Emboss . .	56
3.2	Extrait du fichier des <i>logs</i> Emboss	57
3.3	Exemple de workflow extrait des <i>logs</i> en format pseudo-BPML . .	58
3.4	Traduction du workflow en ensemble d'apprentissage.	59
3.5	Algorithme récursif de parcours en profondeur de graphes	61
3.6	Parcours inverse de graphes avec impression des règles	61
3.7	Algorithme de Hirsh.	63
3.8	Algorithme de prédiction IDHYS pour deux concepts c_1 et c_2	63
3.9	Exemple de description de workflow en format .arff	69
3.10	Exemples de listes de k plus proches voisins	73

Remerciements

Je tiens sincèrement à remercier mon directeur de mémoire, Monsieur Hugues Bersini, directeur de recherche au sein de l'Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle, d'avoir accepté de diriger mon travail de recherche, pour ses conseils, sa patience, sa disponibilité, sa sympathie et son attitude positive et encourageante tout au long de la réalisation de mon mémoire.

Je remercie également l'équipe de recherche chargée du projet BIGRE à l'Institut de Biologie et de Médecine Moléculaires, Monsieur Marc Colet, Olivier Dugas et Joseph Mavor, pour toutes leurs critiques judicieuses et les nombreux conseils en bio-informatique fort appréciés. J'adresse par la même occasion mes remerciements les plus sincères à l'équipe des chercheurs chargés du projet BIGRE à la FUNDP, Monsieur Vincent Englebert, Pierre Buyle et Quentin Dallons, pour leur généreuse contribution apportée à mon travail.

Je remercie, Amin Mantrach et Utku Salihoglu, pour leurs encouragements et leurs commentaires scientifiques. Un merci spécial à Amin et Olivier pour les longues heures passées à lire et à relire les premières versions de ce mémoire.

Finalement, je remercie ma famille, mes parents et ma soeur, pour le soutien moral qu'ils n'ont cessé de m'accorder.

Yves Sangwo

Introduction

De nos jours, l'augmentation constante et spectaculaire de la quantité de données générées automatiquement par les différents laboratoires aux quatre coins du monde a atteint des proportions phénoménales. La bio-informatique est l'une des sciences les plus concernées actuellement par cette augmentation fulgurante de la quantité de données générées grâce notamment aux récents progrès de la biologie dans le domaine de séquençage de génomes. La conséquence logique de cet état des choses est l'apparition d'une nouvelle génération d'applications, conçues spécialement dans le but d'une gestion efficace de cette énorme quantité de données mise à la disposition des chercheurs. C'est en effet dans le cadre d'un projet, appelé BIGRE¹, chargé de la conception d'une telle application que s'inscrit ce mémoire.

L'objectif principal du projet est de fournir aux utilisateurs un outil hautement dynamique et complètement transparent capable d'enchaîner de manière quelconque des expérimentations biologiques disponibles sous forme de services bio-informatiques au sein d'une fédération distribuée d'applications. Cet outil de base, appelé dans la suite Workflow, se veut être un vrai laboratoire biologique virtuel et constitue l'une des composantes clés de BIGRE. Comme dans tout autre laboratoire, le travail des expérimentateurs se voit considérablement facilité par toute aide extérieure intelligente, telle que par exemple un schéma opérationnel d'une expérience antérieure indiquant ses conditions de départ, le câblage effectué et les résultats obtenus ou encore un simple conseil de la part d'un expert spécialisé dans le domaine de l'expérience en question. Dans le même ordre d'idées, il serait extrêmement intéressant d'épauler les utilisateurs du Workflow par un sous-système indépendant capable de leur fournir le même genre d'aide intelligente. C'est ainsi très naturellement que la recherche de solutions à la problématique de conception d'un tel Workflow amélioré, baptisé Workflow Intelligent, se tourne vers le domaine de l'intelligence artificielle. L'idée principale à la base de l'approche "intelligente" adoptée dans notre travail consiste à extraire des informations en relation avec le *workflow* en cours de réalisation de l'ensemble des expérimentations antérieures (des workflows approuvés par des experts) et de les réutiliser pour la prédiction des actions que l'expérimentateur est susceptible d'effectuer à son tour.

La réalisation de la couche intelligente du Workflow requiert un grand nombre d'adaptations spécialisées de l'architecture actuelle de BIGRE que ce soit au niveau des clients, des médiateurs ou des services. En effet, le côté Client doit exécuter les algorithmes "intelligents" et supporter un protocole de communication bien défini

¹Bio-Informatics Grid Resources and Environments.

avec les médiateurs qui de leur côté doivent bien entendu également supporter ce protocole, mais aussi stocker toutes les données en relation avec la couche intelligente du système. Quant aux différents services, il est de leur ressort, à l'image du service Workflow Intelligent décrit dans notre travail, de fournir aux clients toutes les informations indispensables sur la nature de leur "intelligence"¹ (les codes de leurs algorithmes "intelligents", la description des différents formats de données qu'ils utilisent ainsi que toute autre information supplémentaire nécessaire).

Les bases théoriques de notre Workflow Intelligent trouvent leurs racines dans les travaux sur les interfaces prédictives et plus récemment les agents d'interfaces. Ces interfaces visent à affranchir l'utilisateur de tout effort pour rejouer des actions répétitives. Pour cela elles tentent d'apprendre, à partir des corrélations entre les situations rencontrées précédemment et les actions que l'utilisateur accomplit, à prédire ses actions. Ainsi elles facilitent son travail en lui proposant de rejouer automatiquement certaines actions. Le principal avantage de ces interfaces est qu'elles n'obligent pas l'utilisateur à interrompre son travail et n'exigent que très peu d'efforts de sa part pour rejouer (ou ignorer) les actions répétitives prédites car celles-ci lui sont proposées de manière automatique au moment opportun.

Ce mémoire est composé de trois chapitres. La notion d'apprentissage est au coeur de notre travail et il ne peut dès lors être appréhendé sans une connaissance des concepts qui sont à la base des algorithmes d'apprentissage. Le premier chapitre propose donc un état de l'art de la théorie d'apprentissage. Le deuxième chapitre propose une analyse détaillée de la conception du Workflow Intelligent. Il décrit en outre l'architecture générale de BIGRE, ainsi que l'état d'avancement actuel du projet. A la fin du chapitre, nous établissons le Cahier des charges reprenant les étapes essentielles à réaliser en vue de rendre le Workflow Intelligent opérationnel. Le dernier chapitre montre, au moyen d'expérimentations réalisées sur des données réelles, que le principe à la base de notre travail est vérifié en pratique, et qu'il est donc possible de prédire les actions des utilisateurs avec un taux de réussite convenable sur base des expérimentations "similaires" antérieures.

¹Réalisé techniquement à l'aide d'une description fournie par *un code mobile*.

Chapitre 1

Théorie de l'apprentissage

Pour des raisons évidentes il est impossible de mettre en oeuvre, et encore moins d'adapter à nos propres besoins spécifiques, les multiples techniques d'apprentissage que nous offre le domaine de l'intelligence artificielle sans une bonne connaissance préalable de différents principes et concepts qui sont à la base de ces techniques. Ainsi, puisque la notion d'apprentissage est au coeur même de ce mémoire, il est tout à fait logique de commencer notre travail par une introduction détaillée à la théorie de l'apprentissage, ou plus précisément, à la théorie de l'apprentissage inductif - une branche particulière de la théorie de l'apprentissage.

L'apprentissage inductif permet de généraliser de manière implicite ou explicite des concepts (on parle aussi de la création des descriptions de classes) à partir d'exemples. De manière générale, les problèmes d'induction peuvent être décrits comme suit : étant donné *un ensemble d'apprentissage constitué d'exemples*, où chaque exemple (également appelé *observation* ou *cas*) est un vecteur *de valeurs d'attributs*, il faut créer une description capable de classer de nouvelles observations avec une grande précision. De nos jours, plusieurs approches indépendantes à l'apprentissage inductif ont été élaborées. Les plus importantes parmi celles-ci sont les suivantes : l'induction des arbres de décisions, l'induction des règles, l'apprentissage basé sur les instances, les classifieurs Bayesiens, l'inférence grammaticale, les réseaux de neurones à retro-propagation ou encore les algorithmes génétiques.

L'objectif de ce chapitre est de présenter en détail les différents algorithmes qui se cachent derrière la plupart de ces approches et de fournir les éléments nécessaires à la compréhension des chapitres suivants. Le chapitre est composé de huit sections : la première propose un ensemble de notations et de définitions, et introduit la problématique de l'apprentissage ; les sections 1.2 à 1.8 présentent le fonctionnement des différentes techniques d'apprentissage inductif susmentionnées ; et finalement, la conclusion fournit un bref récapitulatif de la matière présentée.

1.1 Apprentissage inductif

L'induction est une manière de raisonner qui consiste à aller des faits particuliers aux lois qui les régissent. De manière analogique, les algorithmes d'apprentissage inductif de concepts visent à déterminer la définition générale d'un concept à partir d'exemples et de contre-exemples de ce concept. Une définition plus formelle de la problématique de l'apprentissage inductif, nécessite l'introduction au préalable d'un ensemble de notations et de définitions.

Définition 1 (Instance) Une instance x est une observation, décrite par un ensemble d'attributs $\{a_1, a_2, \dots, a_n\}$, où $\forall a_i$ prend ses valeurs dans un domaine d_i .

Un test du type $a_i = \nu_j$, qui peut être vrai ou faux selon que l'attribut a_i a ou non ν_j pour valeur, permet d'associer une formule logique à toute instance. Cette formule est une conjonction des littéraux (tests) qui sont vrais pour l'instance. L'ensemble des littéraux qui peuvent être formés sur d_1, d_2, \dots, d_n est appelé langage de description des instances et c'est le choix de ce langage qui fixe l'ensemble \mathbf{X} de toutes les instances qui peuvent être décrites.

Définition 2 (Concept) Un concept c est un sous-ensemble de l'ensemble \mathbf{X} .

Une instance $x \in \mathbf{X}$ est un exemple de c si $x \in c$ et un contre-exemple si $x \notin c$.

Définition 3 (Echantillon) Un échantillon est un ensemble fini de couples $(x, c(x))$, dont les instances sont tirées de l'ensemble d'apprentissage \mathbf{X} avec une probabilité de distribution inconnue.

L'échantillon à partir duquel l'algorithme apprend un concept est appelé ensemble d'apprentissage et celui sur lequel la qualité des résultats de l'algorithme d'apprentissage est évaluée est l'ensemble de test. Ces deux ensembles sont souvent appelés données d'apprentissage. Lorsque ces ensembles sont fournis au début de l'apprentissage par un "superviseur" inconnu, on parle alors d'apprentissage supervisé, et ce par opposition à l'apprentissage non-supervisé, ou *clustering*, dont le but est de regrouper les instances données en classes.

Définition 4 (Prédire) La prédiction de l'hypothèse h pour l'instance x est la valeur de $h(x)$.

Etant donné un ensemble d'apprentissage E_a d'un concept inconnu c , le but d'un algorithme d'apprentissage inductif est de déterminer c . La plupart des algorithmes d'apprentissage procèdent par exploration d'un espace \mathbf{H} de concepts possibles ou hypothétiques appelé l'espace des hypothèses. Le but est d'identifier dans \mathbf{H} l'hypothèse h telle que $h = c$, car dans ce cas h sera capable de prédire pour toute instance x si elle est ou non un exemple du concept recherché. En pratique, l'ensemble d'apprentissage E_a ne constitue souvent qu'un petit sous-ensemble de l'ensemble des instances, et un algorithme d'apprentissage, quel qu'il soit, ne peut garantir, dans le meilleur des cas que $h(x) = c(x), \forall x \in E_a$. Ainsi, on n'aura réellement appris que lorsqu'on aura certaines garanties que $h = c$ pour la majorité des instances de \mathbf{X} et en particulier pour celles qui ne figuraient pas dans les données d'apprentissage. Pour caractériser ces garanties sont utilisées les deux notions suivantes :

- La probabilité d'erreur d'une hypothèse h (ou erreur de généralisation), notée $P_{err}(h)$, est la probabilité que la prédiction de h pour une instance de \mathbf{X} , tirée suivant une probabilité de distribution inconnue, soit incorrecte.
- Le taux d'erreur d'une hypothèse h est donné par la relation

$$T_{err}(h) = \frac{err}{|E_a|}, \text{ où } err = |\{x \in E_a, h(x) \neq c(x)\}|.$$

Définition 5 (Apprendre) Apprendre c'est identifier dans un espace d'hypothèses, l'hypothèse ayant le taux d'erreur et la probabilité d'erreur minimaux.

Les hypothèses de \mathbf{H} sont généralement représentées par des formules logiques θ_{h_i} d'ordre 0 et 1, exprimées à l'aide des littéraux du langage de description des instances et des opérateurs booléens usuels (\vee, \wedge, \neg) qui, ensemble, constituent le langage de description des hypothèses.

Définition 6 (Généraliser/Spécialiser) Généraliser/Spécialiser une instance ou une hypothèse consiste à substituer à un des littéraux composant la description de l'instance ou la représentation de l'hypothèse respectivement, un littéral plus général/spécifique.

Une hypothèse h_1 est plus spécifique qu'une hypothèse h_2 (h_2 est plus générale), noté $h_1 < h_2$, si et seulement si h_1 est incluse dans h_2 . Cet ordre peut être transposé au langage de représentation des hypothèses au moyen de l'implication logique. Si l'on suppose que toute valeur d'un attribut hiérarchisé implique toute valeur plus spécifique de la hiérarchie, nous dirons que h_1 est plus spécifique que h_2 si et seulement si $\theta_{h_1} \rightarrow \theta_{h_2}$. Cet ordre permet de déterminer l'appartenance d'une instance à une hypothèse ou à un concept sans avoir à en énumérer tous les éléments. En effet, $x \rightarrow h$ est équivalent à $h(x) = 1$, ou $x \in h$. Dans ce cas, nous dirons que h couvre x , ou encore que x satisfait h . Lorsque h couvre tous les exemples d'un échantillon E (consistance complète) et aucun contre-exemple (consistance correcte), i.e. $\forall x \in E, h(x) = c(x)$, alors h est cohérente avec E .

Définition 7 (Biais) Le biais est l'ensemble minimal de faits \mathbf{B} tel que

$$\forall x \in \mathbf{X}, [(\mathbf{B} \wedge E_a \wedge x) \vdash h(x)], \text{ où } \vdash \text{ est une conséquence logique.}$$

Autrement dit, selon [Mitchell, 1997] le biais est l'ensemble minimal de faits permettant de justifier de manière déductive l'induction effectuée par l'algorithme d'apprentissage. Le biais assure la convergence des algorithmes d'apprentissage vers l'hypothèse ayant le taux d'erreur et la probabilité d'erreur les plus faibles lors de l'exploration de l'espace d'hypothèses \mathbf{H}^1 . Le biais peut être subdivisé en quatre catégories :

- **Biais de description** qui dénote le fait que le langage de description des instances fixe implicitement l'espace des hypothèses à explorer. Si les instances sont mal décrites ou décrites de manière incomplète, alors \mathbf{H} peut ne pas contenir le concept c ou même une hypothèse ayant une faible probabilité d'erreur. Ce qui signifie très logiquement qu'un algorithme d'apprentissage ne peut apprendre que ce qu'on lui décrit.

¹Généralement l'énumération de toutes les hypothèses est tout simplement impossible, raison pour laquelle une grande majorité des algorithmes d'apprentissage procède plutôt à une exploration partielle de \mathbf{H} visant à converger vers une telle hypothèse.

- **Biais de représentation** qui exprime le fait que le langage de représentation des hypothèses fixe explicitement l'espace des hypothèses à explorer. Comme dans le cas du biais de description, un mauvais langage nuit à l'efficacité de l'apprentissage.
- **Biais d'induction** qui peut être vu comme l'heuristique employée par l'algorithme pour diriger l'exploration de l'espace des hypothèses.
- **Biais d'échantillon** ou biais d'évaluation qui exprime le fait que l'on ne peut pas apprendre et évaluer la probabilité d'erreur d'une hypothèse sur le même échantillon. $T_{err}(h)$ est souvent, en effet, une estimation très optimiste de $P_{err}(h)$.

Définition 8 (Bruit) *Les données d'apprentissage peuvent contenir des erreurs.*

Il y a deux types d'erreurs (bruit) :

- *Incohérence.* $\exists(x,y) \in E_a \mid c(x) \neq y$.
- *Valeur inconnue.* La valeur de certains attributs manque ou est incorrecte.

1.2 Arbres de décision

L'apprentissage d'arbres de décision est une des méthodes les plus utilisées en apprentissage inductif. Nommées initialement partitionnement récursif, les développements importants de [Breiman L. and Stone, 1984] les ont fait connaître sous l'acronyme CART (Classification and Regression Trees) ou encore de C4.5 [Quinlan, 1993] dans la communauté informatique.

1.2.1 Principe de la méthode

Les algorithmes d'apprentissage basés sur la construction d'un arbre de décision réduisent le problème de l'apprentissage d'un ensemble de concepts à celui du choix d'un ensemble de tests permettant de diviser l'ensemble d'apprentissage en parties ne contenant (presque) que des instances d'un même concept. Ils construisent pour cela un arbre, appelé *arbre de décision*, dont chaque noeud interne est étiqueté par un test sur la valeur d'un des attributs de la description des instances, chaque arc par un des résultats possibles de ce test et chaque feuille par un concept auquel appartient la majorité des instances de la partie associée à cette feuille. Ainsi, à chaque noeud d'un arbre de décision correspond la partie de l'ensemble d'apprentissage constituée des instances qui satisfont les ancêtres du noeud.

L'espace des hypothèses exploré par ces algorithmes est celui des conjonctions de littéraux du langage de description des instances. La conjonction des littéraux étiquetant une branche de l'arbre (de la racine à la feuille) constitue la représentation logique de l'hypothèse apprise pour le concept étiquetant la feuille de cette branche. Autrement dit, toute instance satisfaisant tous les littéraux (les tests effectués dans les noeuds de l'arbre) étiquetant une branche est considérée

comme étant un exemple du concept étiquetant la feuille de cette branche.

Généralement les hypothèses les plus simples (en termes de nombre de littéraux par exemple) ont souvent une faible probabilité d'erreur, ce qui semble bien rejoindre le fameux principe d'Ockham (de plus en plus contesté) qui veut qu'entre deux théories expliquant un même phénomène, la plus simple est toujours préférable. Ainsi l'objectif des algorithmes d'apprentissage est de déterminer l'ensemble minimal de tests qui permet de partitionner au mieux l'ensemble d'apprentissage.

1.2.2 Algorithme C4.5

L'algorithme de C4.5 de construction d'un arbre de décision, présenté dans le tableau 1.1, consiste à partitionner de manière récursive l'ensemble d'apprentissage jusqu'à ce que les ensembles ainsi obtenus soient suffisamment *purs*. Ces ensembles sont considérés comme étant purs, lorsqu'ils ne contiennent presque que des instances du même concept. L'algorithme procède à une exploration de l'espace des hypothèses dirigée de l'hypothèse la plus générale (l'arbre vide) vers les hypothèses les plus spécifiques. Les hypothèses sont construites par additions successives de littéraux. Dans la suite nous présentons les deux principales procédures de l'algorithme, ainsi que ses trois fonctions de simplification de l'arbre final.

-
- Si l'ensemble E_a est suffisamment pur
 \hookrightarrow Retourner le noeud étiqueté par le concept le plus fréquent.
 - Sélectionner le test qui partitionne au mieux l'ensemble E_a .
 - Créer un noeud N étiqueté par ce test.
 - Partitionner E_a en fonction des résultats possibles de ce test.
 - Pour chaque partie de l'ensemble E_a créer un arbre de décision.
 - Faire de la racine de l'arbre ainsi créé un fils de N .
 - Retourner N .
-

TAB. 1.1 – Procédure récursive de construction de l'arbre de décision de C4.5

1.2.2.1 Calcul de l'entropie

A chaque noeud de l'arbre est associée une quantité appelée *entropie*, ou encore impureté, qui mesure le degré de mélange des concepts dans la partie de l'ensemble d'apprentissage associée à ce noeud. Plus l'entropie est élevée, plus le mélange des concepts est important. Un échantillon est dit pur si l'entropie est nulle, et donc si toutes les instances appartiennent au même concept. Si $E_a(d)$ est la partie de l'ensemble d'apprentissage de l'ensemble de concepts $\{c_1, c_2 \dots c_p\}$ associé au noeud d , l'entropie de cet échantillon est donnée par :

$$\text{entropie}(E_a(d)) = - \sum_{i=1}^p P(c_i|E_a(d)) \times \log_2 P(c_i|E_a(d)),$$

où $E_a(d)$ peut être vu comme fixant un littéral précis pour l'ensemble des instances ayant un attribut correspondant à l'étiquette du noeud d ("passant" par d), et où $P(c_i|E_a(d))$ est donc la probabilité conditionnelle qui représente la proportion d'exemples du concept c_i parmi les instances satisfaisant le littéral fixé en d . Dans la théorie de l'information, la valeur $-\log_2 P(c_i|E_a(d))$ représente la *quantité de l'information*¹ que $E_a(d)$ offre sur la conclusion c_i , et donc multipliée par la probabilité conditionnelle, cette valeur donne bien le poids du concept i au noeud d . Ainsi, la sommation sur l'ensemble des concepts associés au noeud d , donne effectivement une mesure, qui peut être interprétée comme le degré de mélange de ces concepts. Lorsque l'entropie atteint une certaine valeur arbitraire (généralement fixée à zéro), l'algorithme crée une feuille (condition d'arrêt de la récursivité).

1.2.2.2 Choix du meilleur test

Un test est une question portant sur un des attributs de la description des instances. A chaque résultat possible du test est associée une partie de l'ensemble d'apprentissage. Cette approche possède deux limites. La première est que les attributs continus peuvent engendrer un nombre considérable de parties, et la seconde est due au fait qu'en présence de bruit dans les données, la valeur de certains attributs peut ne pas être connue. Clairement, chacun de ces trois cas possibles, nécessite un traitement spécifique.

Lorsque les attributs sont discrets, le problème est celui du choix du test à effectuer dans un noeud d , dont l'ensemble d'apprentissage associé est $E_a(d)$. L'entropie de la partition induite par un test X de l'échantillon $E_a(d)$ est donné par :

$$\text{entropie}_X(E_a(d)) = \sum_{j=1}^m \frac{|E_a(d_j)|}{|E_a(d)|} \times \text{entropie}(E_a(d_j)),$$

où $\{d_1, d_2 \dots d_m\}$ est l'ensemble des noeuds (fils de d) créés à la suite du test X . Ainsi, en sommant les entropies de chacun des fils, pondérées par le poids relatif de ce fils au sein de l'ensemble initial $E_a(d)$, on obtient un gain de réduction de l'entropie :

$$\text{gain}(X) = \text{entropie}(E_a(d)) - \text{entropie}_X(E_a(d)).$$

Ce gain favorise les tests sur les attributs possédant beaucoup de valeurs possibles, puisque dans ce cas, les poids relatifs des fils diminuent considérablement (il y aura beaucoup plus de noeuds fils), en entraînant par la même occasion la diminution de $\text{entropie}_X(E_a(d))$. Pour limiter l'importance de tels attributs, C4.5 fait appel à une correction :

$$\text{correction}(X) = - \sum_{j=1}^m \frac{|E_a(d_j)|}{|E_a(d)|} \times \log_2 \frac{|E_a(d_j)|}{|E_a(d)|},$$

qui permet de caractériser l'importance des poids des noeuds fils (qui dépendent justement des valeurs possibles de l'attribut du noeud d) en fonction de la quantité d'information que représentent ces poids pour chaque fils. Ainsi, le meilleur test

¹La quantité d'information $h(x)$ associée à la réalisation d'un événement x de probabilité P_x est donnée par la relation $h(x) = \log_2(\frac{1}{P_x}) = -\log_2 P_x$.

est celui qui, parmi tous les tests possibles (autant que d'attributs descripteurs), maximise le gain corrigé :

$$\text{gain} - \text{corrigé}(X) = \frac{\text{gain}(X)}{\text{correction}(X)}.$$

Dans le cas où l'attribut est continu, l'ensemble d'apprentissage est divisé en deux parties, selon l'ensemble $\{\nu_1, \nu_2 \dots \nu_n\}$ des valeurs observées de cet attribut. Le problème du choix du test réside alors dans le choix d'un seuil de comparaison dans l'ensemble des seuils $\{\frac{\nu_1+\nu_2}{2}, \frac{\nu_2+\nu_3}{2} \dots \frac{\nu_{n-1}+\nu_n}{2}\}$. Le seuil qui sera utilisé pour l'attribut continu en question lors de la recherche du meilleur test est celui qui maximise la fonction suivante :

$$\text{gain}(X) = \text{entropie}(E_a(d)) - \text{entropie}_X(E_a(d)) - \log_2 \frac{n-1}{|E_a(d)|}.$$

Le problème du calcul de la partition engendré par un noeud, et donc de l'entropie de la partition induite par un test sur cet attribut se pose lorsque la valeur de l'attribut est inconnue. Dans ce cas on adopte une approche probabiliste. Une instance dont la valeur pour l'attribut testé est inconnue est insérée dans chacune des parties, mais pondérée par la probabilité d'appartenance à chacune de ces parties. Plus précisément, une telle instance, insérée dans la partie associée à la valeur ν_i , est pondérée par le rapport entre le nombre d'instances ayant ν_i pour valeur et le nombre d'instances dont la valeur pour l'attribut testé est connue. Dans ce cas le calcul du gain de l'entropie induit par un test sur un attribut dont certaines valeurs sont inconnues est donné par :

$$\text{gain}(X) = P \times (\text{entropie}(E_a(d)) - \text{entropie}_X(E_a(d))),$$

où P est la proportion d'instances de l'ensemble d'apprentissage $E_a(d)$ dont la valeur pour l'attribut testé est inconnue. La correction du gain est elle aussi altérée, et est alors calculée comme si le test créait non plus m parties mais $m + 1$, l'ensemble des instances dont la valeur pour l'attribut testé est inconnue étant considéré comme une partie supplémentaire.

1.2.2.3 Complexité de l'apprentissage

La complexité temporelle au pire des cas de la construction de l'arbre de décision est $O(a^2 n(p+m))$, où a est le nombre total d'attributs (profondeur maximale de l'arbre), n est le nombre d'instances de l'ensemble d'apprentissage, p est le nombre de concepts et m , le nombre moyen des valeurs possibles d'un attribut. Lorsque certaines instances contiennent des attributs continus dans leur description, cette complexité devient quadratique en n .¹

1.2.2.4 Elagage de l'arbre

L'arbre construit par la procédure présentée dans le tableau 1.1 est souvent plus complexe que l'arbre optimal (en terme de nombre de noeuds) et possède une probabilité d'erreur largement supérieure. Une procédure appelée *élagage* utilise un échantillon de test pour simplifier un arbre de décision en remplaçant certains de ses sous-arbres par des feuilles. L'élagage procède à un parcours de bas en

¹Pour un développement détaillé de l'analyse de la complexité de C4.5, voir [Quinlan, 1993].

haut de l'arbre (des feuilles vers la racine) et compare la probabilité d'erreur de chaque sous-arbre avec celle qu'aurait une feuille, étiquetée par le concept le plus fréquent dans l'ensemble d'apprentissage, associée à la racine de ce sous-arbre. Lorsque la probabilité d'erreur de la feuille est moindre, le sous-arbre est supprimé et remplacé par la feuille. La probabilité d'erreur d'une feuille est approchée au moyen de $N \times U(CF, e, N, 0)$ ¹, où N est la taille de l'échantillon de test et e le nombre d'erreurs de l'arbre sur cet échantillon (i.e. nombre d'instances dont le concept d'appartenance est incorrectement prédit). CF peut être vu comme étant l'incertitude autorisée sur cette estimation, fixée par défaut à 25%. La probabilité d'erreur d'un arbre est estimée par la somme des probabilités d'erreur de ses sous-arbres. L'arbre ainsi élagué a en général un taux d'erreur supérieur à l'arbre initial, mais une probabilité d'erreur moindre.

1.2.2.5 Transformation en un ensemble de règles

Une hypothèse peut être associée à chacune des branches de l'arbre : la conjonction des littéraux (des tests) étiquetant une branche constitue la représentation logique d'une hypothèse apprise pour le concept étiquetant la feuille de la branche. Ainsi, une branche peut être vue comme une règle du type $h \rightarrow C$ où h est une hypothèse et C un concept. Lorsqu'une instance satisfait la partie prémisse d'une telle règle, la partie conclusion spécifie le concept auquel appartient l'instance. L'instance satisfait alors la règle. L'ensemble des règles ainsi construit est exclusif (une instance ne peut satisfaire qu'une seule règle) et exhaustif (toute instance satisfait au moins une des règles).

1.2.2.6 Simplification des règles

La probabilité d'erreurs des hypothèses apprises peut être réduite (ayant ou non élagué l'arbre au préalable) en supprimant des littéraux dans leur représentation logique, autrement dit en supprimant des tests dans la partie prémisse des règles construites. Soit R une règle du type $H \rightarrow C$ et R^- une règle plus générale du type $H^- \rightarrow C$, où H^- est obtenu en supprimant le test X de H . Une instance de l'ensemble d'apprentissage satisfaisant H^- peut ou non satisfaire X et peut ou non appartenir au concept C . Ces instances peuvent être dénombrées de la manière suivante :

	$\in C$	$\notin C$
Satisfait X	Y_1	E_1
\neg Satisfait X	Y_2	E_2

La probabilité d'erreur de la partie prémisse de R peut alors être estimée par $U_R = U(CF, E_1, Y_1 + E_1, 0)$ et la probabilité d'erreur de celle de R^- par $U_{R^-} = U(CF, E_1 + E_2, Y_1 + Y_2 + E_1 + E_2, 0)$. Le processus de simplification d'une règle consiste à supprimer de la partie prémisse de la règle, le littéral dont la suppression produit la règle de plus faible probabilité d'erreur. Dans notre cas, si $U_{R^-} < U_R$, alors R peut être remplacé par R^- . Ce processus est répété tant qu'une telle suppression est possible.

L'intérêt majeur de ce procédé de simplification de règles est qu'il ne requiert pas, contrairement à l'élagage, d'échantillon de test. La probabilité d'erreur des

¹Fonction d'estimation de la probabilité d'erreur d'une hypothèse.

règles est en effet évaluée au moyen de l'ensemble d'apprentissage seulement. De l'autre côté, son inconvénient principal réside dans le fait que l'ensemble des règles simplifiées ne constitue pas un ensemble mutuellement exclusif et exhaustif, contrairement à l'ensemble des règles non simplifiées. Une instance peut en effet satisfaire les parties prémisses de plusieurs règles. Le problème se pose alors de déterminer la règle à considérer pour prédire le concept auquel appartient l'instance. La solution adoptée dans C4.5 consiste à ordonner les règles par probabilité d'erreur croissante et à considérer seulement la première règle satisfaite.

1.3 Espaces des versions

1.3.1 Principe

[Mitchell, 1997] a défini l'espace des versions d'un concept c induit par l'ensemble d'apprentissage E_a comme l'ensemble des hypothèses cohérentes avec E_a . Si E_a^+ est l'ensemble des exemples de E_a et E_a^- l'ensemble des contre-exemples, l'espace des versions de c , noté VS , est défini par :

$$VS = \{h \in \mathbf{H}, h \text{ cohérent avec } E_a\} = \{h \in \mathbf{H}, E_a^+ \subseteq h \wedge E_a^- \cap h = \emptyset\}.$$

Bien évidemment, l'espace des versions contient c . Une solution à l'apprentissage inductif d'un concept consiste donc à déterminer son espace des versions d'une part, et à identifier le concept dans cet espace d'autre part. Cependant l'espace des hypothèses \mathbf{H} est bien souvent très grand ou infini et il est impossible d'en énumérer tous les éléments pour déterminer les hypothèses qui sont cohérentes avec E_a et celles qui ne le sont pas. L'ordre partiel de spécialisation/généralisation existant entre les hypothèses permet de représenter l'espace des versions au moyen de deux sous-ensemble : l'ensemble \mathbf{S} de ses éléments les plus spécifiques et l'ensemble \mathbf{G} de ses éléments les plus généraux.

$$\mathbf{S} = \{s \in \mathbf{H}, s \text{ cohérent avec } E_a \wedge \nexists st \in \mathbf{H}, [st \prec s \text{ et } st \text{ cohérent avec } E_a]\}.$$

$$\mathbf{G} = \{g \in \mathbf{H}, g \text{ cohérent avec } E_a \wedge \nexists gt \in \mathbf{H}, [g \prec gt \text{ et } gt \text{ cohérent avec } E_a]\}.$$

Il a été démontré que $VS = \{h \in \mathbf{H}, (\exists s \in \mathbf{S})(\exists g \in \mathbf{G})(s \prec h \prec g)\}$, où \mathbf{S} est la frontière inférieure (généralisation la plus spécifique des exemples) de VS , et \mathbf{G} comme sa frontière supérieure (généralisation maximale des exemples). Cette représentation sera notée $[\mathbf{S}, \mathbf{G}]$.

1.3.2 Algorithme CEA

La représentation $[\mathbf{S}, \mathbf{G}]$ a permis d'élaborer un algorithme incrémental, connu sous le nom de CEA (Candidate Elimination Algorithm), pour la construction de l'espace des versions induit par un ensemble d'apprentissage. Au départ, \mathbf{S} et \mathbf{G} sont initialisés par l'hypothèse la plus spécifique et l'hypothèse la plus générale (ensemble ils représentent l'espace des hypothèses tout entier). Lorsqu'une instance est traitée, \mathbf{S} et \mathbf{G} sont mis à jour pour éliminer de l'espace des versions toutes les

hypothèses qui n'en font pas partie. Si l'instance est un exemple, les éléments de \mathbf{S} sont généralisés aussi peu que possible pour couvrir l'exemple tout en restant cohérent avec l'ensemble des instances déjà traitées, et les éléments de \mathbf{G} qui ne couvrent pas l'exemple sont supprimés. Si l'exemple est un contre-exemple, les éléments de \mathbf{G} sont spécialisés suffisamment (aussi peu que possible) pour ne pas couvrir le contre-exemple tout en restant cohérent avec l'ensemble des instances déjà traitées, et les éléments de \mathbf{S} qui couvrent le contre-exemple sont supprimés. L'espace des versions ainsi construit est indépendant de l'ordre de traitement des instances et contient toutes les hypothèses cohérentes avec l'ensemble d'apprentissage et celles-là seulement. L'apprentissage terminé, les ensembles \mathbf{S} et \mathbf{G} peuvent être utilisés pour prédire si une instance est un exemple ou un contre-exemple. Si celle-ci satisfait toutes les hypothèses de \mathbf{S} , elle satisfait donc le concept et est un exemple. Si elle ne satisfait que certaines des hypothèses de \mathbf{S} , il est probable que ce soit un exemple. À l'inverse, si elle ne satisfait aucune des hypothèses de \mathbf{G} , elle ne satisfait donc pas le concept appris et est un contre-exemple. Si elle satisfait certaines hypothèses de \mathbf{G} mais aucune hypothèse de \mathbf{S} , il n'est pas possible de décider si c'est un exemple ou un contre-exemple.

Malheureusement l'algorithme CEA originel possède plusieurs limitations pratiques dont la plus importante est l'incapacité de gérer des données bruitées. En effet, si les valeurs d'attributs de certaines des instances sont inconnues, l'algorithme est incapable de déterminer si une hypothèse de \mathbf{S} ou \mathbf{G} couvre cette instance et donc de construire ces ensembles. De même, l'espace des versions induit par un ensemble d'apprentissage incohérent est vide puisque dans ce cas, aucune hypothèse ne pourra être cohérente avec cet ensemble : elle couvrira les vrais exemples mais aussi les exemples incorrectement pris pour des contre-exemples.

1.3.3 Traitement du bruit

Plusieurs algorithmes ont été proposés pour le traitement de données bruitées. Parmi eux CALM, qui explore l'espace des hypothèses (l'espace des conjonctions des littéraux) de l'hypothèse la plus générale vers les hypothèses les plus spécifiques. La longueur (en nombre de littéraux) l des hypothèses est un paramètre de l'algorithme d'apprentissage et doit être fixée par l'utilisateur. L'algorithme recherche les hypothèses qui couvrent **beaucoup** d'exemples et **peu** de contre-exemples. Les notions de **beaucoup** et **peu** se traduisent par des seuils entiers, notés respectivement α et β , dont la valeur est fixée par l'utilisateur. Une hypothèse est dite *forte* si elle couvre plus de α exemples et plus de β contre-exemples, elle est dite *satisfaisante* si elle couvre plus de α exemples et moins de β contre-exemples, enfin, elle dite *faible* si elle couvre moins de α exemples. Le but de l'algorithme est de construire toutes les hypothèses satisfaisantes de l'espace des hypothèses. Les hypothèses sont générées par additions successives de littéraux à l'hypothèse la plus générale (forte par définition). Elles sont construites et représentées sous la forme d'un arbre. La racine de l'arbre est étiquetée par l'hypothèse la plus générale de l'espace des hypothèses et chaque noeud de niveau k par k littéraux. La figure 1.1 présente l'arbre le plus grand qui peut être construit pour une longueur maximale de trois littéraux pour le langage de description $\{a, b, c, d\}$. L'arbre est construit de manière incrémentale en ajoutant un à un tous les littéraux

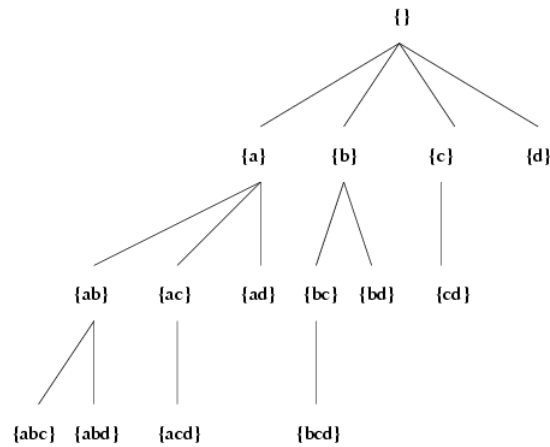


FIG. 1.1 – Arbre construit par CALM pour l'ensemble des littéraux $\{a, b, c, d\}$

du langage de description. Etant donné que l'addition d'un littéral à une hypothèse faible ne peut produire qu'une hypothèse faible (car plus spécifique), l'addition des littéraux à une hypothèse se poursuit aussi longtemps que l'hypothèse reste forte. Dès lors que l'addition d'un littéral produit une hypothèse satisfaisante, aucun littéral n'est plus ajouté à l'hypothèse.

Comme les algorithmes basés sur les arbres de décision, CALM construit et représente les hypothèses sous la forme d'un arbre. Cependant, contrairement à ces algorithmes, il ne cherche pas les hypothèses permettant de partitionner au mieux l'ensemble d'apprentissage, mais toutes les hypothèses satisfaisantes, et se rapproche en cela des algorithmes basés sur les espaces des versions. CALM a été conçu pour l'apprentissage d'un seul concept mais construit deux ensembles d'hypothèses, un pour le concept, un autre pour sa négation. La procédure de prédiction est la suivante : étant donné un ensemble d'hypothèses, une instance est considérée comme un exemple du concept pour lequel ces hypothèses ont été apprises, si elle satisfait **beaucoup** d'hypothèses et un contre-exemple si elle en satisfait **peu**. Dans les autres cas, il n'est pas possible de prédire si l'instance appartient ou non au concept. Encore une fois, les notions **peu** et **beaucoup** sont à définir par l'utilisateur. Cette procédure de prédiction est opérée pour le concept et pour sa négation. Si ces prédictions sont contradictoires, l'appartenance de l'instance au concept ne peut pas être déterminée.

Outre sa mauvaise complexité théorique au pire cas, l'inconvénient de CALM est le nombre élevé de seuils à régler, réglage souvent bien difficile à réaliser. Cependant, ces seuils permettent de relaxer l'exigence de cohérence de l'algorithme CEA originel. Les hypothèses recherchées ne sont pas uniquement celles qui sont cohérentes avec l'ensemble d'apprentissage mais les plus satisfaisantes (éventuellement cohérentes) qui peuvent être construites. CALM est donc capable d'apprendre à partir de données bruitées (contenant des incohérences).

1.4 Raisonnement à base des cas

Traditionnellement l'apprentissage en induction basée sur des cas (ou des instances¹), appelé **raisonnement à base de cas** (*case-based reasoning* ou encore CBR), s'appuie sur un ensemble d'apprentissage constitué des expériences antérieures décrites dans des formats complètement structurés tels que des objets ou des enregistrements des bases de données.

1.4.1 Principe

Le raisonnement à base des cas est une approche de résolution de problèmes qui utilise des expériences passées pour résoudre de nouveaux problèmes. L'ensemble d'apprentissage forme une base de cas. Typiquement un cas contient au moins deux parties : une description de situation représentant un problème - l'instance, et une solution pour remédier à cette situation - le concept auquel cette instance appartient. Parfois, le cas décrit également les conséquences résultant de l'application de la solution. Les techniques CBR permettent de produire de nouvelles solutions en extrapolant sur les situations similaires au problème à résoudre. Cette approche est adéquate pour les domaines où la similarité entre les descriptions de problèmes nous donne une indication sur l'utilité des solutions antécédantes.

L'approche CBR offre de nombreux avantages, car elle est très souvent plus facile à mettre en oeuvre que les autres démarches basées sur un modèle du domaine (base de règles). La démarche CBR permet d'éviter les problèmes d'acquisition de connaissance (*knowledge bottleneck*) qui rendent difficile la conception de bases de connaissances de taille importante.

1.4.2 Systèmes CBR

Un système CBR est une combinaison de processus et de connaissances (*knowledge containers*) qui permettent de préserver et d'exploiter les expériences passées. Le modèle générique présenté sur la figure 1.2 permet de simplifier la présentation du système. Il montre comme principaux processus la recherche (*retrieval*), l'adaptation (*reuse*), la maintenance (*retain*) et la construction (*authoring*) et comme structures de connaissances le vocabulaire d'indexation, la base de cas, les métriques de similarité et les connaissances d'adaptation.

1.4.2.1 Processus

La recherche est la phase qui permet de déterminer les cas de la base les plus similaires au problème à résoudre. La procédure de recherche est habituellement implantée par une sélection des plus proches voisins, *k-nearest neighbors*, ou par la construction d'une structure de partitionnement obtenue par induction. L'approche des plus proches voisins utilise des métriques de similarité pour mesurer la correspondance entre chaque cas et le nouveau problème à résoudre. On

¹Il s'agit de l'induction basée sur les instances

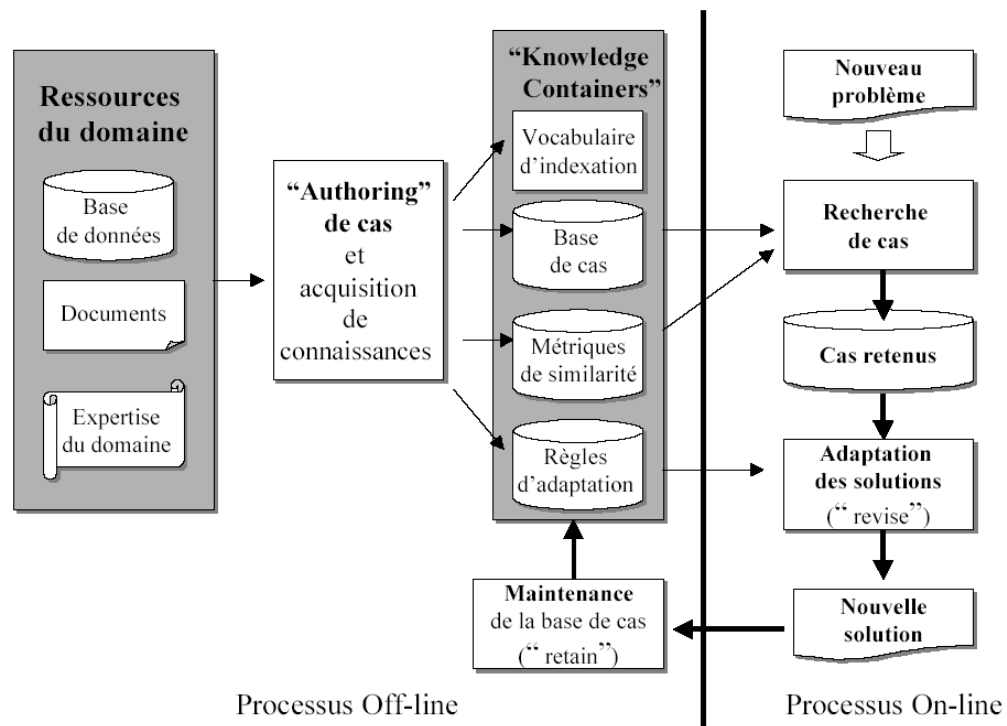


FIG. 1.2 – Modèle générique d'un système CBR

détermine les k instances les plus proches à la nouvelle instance, et on retient comme prédiction le concept auquel appartient la majorité des k instances analysées. Les performances de tels algorithmes dépendent, bien entendu, de la mesure de similarité employée. L'approche par induction génère un arbre qui répartit les cas selon différents attributs et qui permet de guider le processus de recherche.

L'adaptation consiste à modifier les cas sélectionnés lors de la phase de recherche et à réutiliser les solutions de ces cas pour résoudre le problème courant. En général, on retrouve deux approches pour l'adaptation de cas. Par l'approche transformationnelle (ou structurelle), on obtient une nouvelle solution modifiant des solutions antécédentes et en les réorientant afin de satisfaire le nouveau problème, alors que par l'approche générative (ou dérivationnelle), on garde pour chaque cas passé, une trace des étapes qui ont permis de générer la solution. Pour un nouveau problème, une nouvelle solution est générée en appliquant l'une de ces suites d'étapes. Le processus d'adaptation est très rarement automatique dans des systèmes CBR. Pour la plupart des systèmes, une intervention humaine est nécessaire pour générer une solution à partir d'exemples.

Durant le cycle de vie d'un système CBR, certaines stratégies doivent être préconisées pour intégrer de nouvelles solutions dans la base de cas et pour modifier les structures du système pour en optimiser les performances. Il s'agit du processus de *maintenance*. Une stratégie simple pour mettre en oeuvre ce pro-

cessus est d'insérer tout nouveau cas dans la base. Mais d'autres stratégies, plus sophistiquées, visent à apporter des modifications à la structuration de la base des cas - l'indexation - pour en faciliter l'exploitation.

La construction est un processus, en amont des activités de résolution de problèmes du système CBR, qui sous-tend la structuration initiale de la base de cas et des autres connaissances du système à partir de différentes ressources tels des documents, bases de données ou transcriptions d'interviews avec des praticiens du domaine. Ce processus, souvent effectué manuellement par le concepteur du système, se prête moins bien à l'automatisation car il nécessite une connaissance du cadre applicatif pour guider, entre autre, la sélection du vocabulaire d'indexation et la définition des métriques de similarités.

1.4.2.2 Connaissances

Les différentes connaissances utilisées par un système CBR sont regroupées en quatre catégories, appelées *knowledge containers* :

- *Vocabulaire d'indexation*. Un ensemble d'attributs ou de traits (*features*) qui caractérisent la description de problèmes et de solutions du domaine. Ces attributs sont utilisés pour construire la base de cas et jouent un rôle important lors de la phase de recherche.
- *Base de cas*. L'ensemble des expériences structurées qui seront exploitées par les phases de recherche, d'adaptation et de maintenance.
- *Mesures de similarité*. Des fonctions pour évaluer la similarité entre deux ou plusieurs cas. Ces mesures sont définies en fonction des traits et sont utilisées pour la recherche dans la base des cas.
- *Connaissance d'adaptation*. Des heuristiques du domaine, représentées habituellement sous forme de règles, permettant de modifier les solutions et d'évaluer leur applicabilité à de nouvelles situations.

1.4.3 Modèles CBR

Il existe un grand nombre de modèles pour le raisonnement à base des cas regroupés en trois grandes familles : structurelle, conversationnelle et textuelle.

1.4.3.1 Modèle structurel

Le modèle structurel a émergé des premières vagues applicatives de systèmes CBR. Dans ce modèle toutes les caractéristiques importantes, pour décrire un cas, sont déterminées à l'avance. Ainsi, le concepteur élabore un modèle de données du domaine applicatif. Les cas sont complètement structurés et sont représentés par des paires $\langle \text{attribut}, \text{valeur} \rangle$. Un attribut représente alors une caractéristique importante du domaine d'application. La similarité entre deux cas est mesurée en fonction de la distance entre les valeurs de mêmes attributs fréquemment estimée par *les mesures euclidiennes* et de *Hamming*. La similarité globale entre deux cas est habituellement évaluée par une somme pondérée de la similarité de chacun des attributs. Comme les attributs d'un cas n'ont pas tous la même importance et que cette importance varie d'une situation à l'autre, un poids est attribué à

chaque attribut de chaque cas. Ces poids permettent de pondérer la similarité globale entre deux cas en accordant un vote plus important aux attributs les plus méritant.

1.4.3.2 Modèle conversationnel

Dans l'approche traditionnelle (le modèle structurel) un problème doit être complètement décrit avant que ne débute la recherche dans la base de cas. Cette exigence présuppose une expertise du domaine d'application permettant de bien caractériser une situation précise et de sélectionner les principaux facteurs pouvant influencer la résolution des problèmes futurs. Toutefois pour certains domaines ces aspects sont difficiles à déterminer à l'avance et c'est dans le but de surmonter ces difficultés que le modèle conversationnel a été proposé. Actuellement, ce modèle est le plus répandu parmi les applications commerciales du CBR.

Le modèle CBR conversationnel mise sur l'interaction entre l'utilisateur et le système pour définir progressivement le problème à résoudre et pour sélectionner les solutions les plus appropriées.¹ Un cas conversationnel consiste en trois parties : un problème P (une brève description textuelle de quelques lignes) ; une série de questions et de réponses Q_A représentés par des index et exprimés sous forme de questions, permettant d'obtenir plus d'information sur la description du problème (chaque question a un poids représentant son importance par rapport au cas) ; et, une action A qui est une description textuelle *free-text* de la solution à mettre en oeuvre pour ce problème. Cette représentation des cas est donc une extension du modèle structurel avec des attributs de trois types bien précis : *descriptions*, *questions* et *actions*. L'interaction entre le système et l'utilisateur se fait comme suit : l'utilisateur fournit au système une brève description textuelle du problème à résoudre et le système calcule la similarité entre cette description et la section **problème** des cas. Le système propose alors une série de questions, parmi lesquelles l'utilisateur choisit celles auxquelles il souhaite répondre et pour chaque réponse fournie, le système réévalue la similarité de chacun des cas. Les questions n'ayant pas reçu de réponse sont présentées par ordre décroissant de priorité. Lorsqu'un des cas atteint un niveau de similarité suffisamment élevé (supérieur à un certain seuil), le système propose ce cas comme solution. Si aucun cas n'atteint un degré de similarité suffisant et que le système n'a plus de questions à poser, le problème est stocké comme étant non-résolu.

Remarque. Les systèmes CBR conversationnels n'effectuent pas d'adaptation des solutions passées. Une des raisons est que la section **solutions** des cas est *free-text*, ce qui rend relativement difficile la formulation de connaissances d'adaptation.

1.4.3.3 Modèle textuel

Les travaux sur le raisonnement à base des cas textuels portent sur la résolution de problèmes à partir d'expériences dont la description est contenue dans des documents textuels. Dans cette approche, les cas textuels sont soit *non-structurés* ou *semi-structurés*. Ils sont non-structurés si leur description est complètement *free-text* et semi-structurés lorsque le texte est découpé en plusieurs portions étiquetées

¹Voir [Aha D.W. and H., 2000]

par des descripteurs tels que **problème**, **solution** ou autre. Un cas textuel non-structuré est un cas dont le seul attribut est textuel, tandis qu'un cas textuel semi-structuré est un cas a un sous-ensemble d'attributs textuels. Pour ce modèle, la représentation textuelle des cas joue habituellement un rôle important dans la résolution du problème. Elle peut être une finalité en soit, mais peut aussi décrire une situation et une solution qui ne peuvent être facilement codifiées selon un schéma de représentation de connaissance. Cette voie de recherche est relativement récente, et à ce jour aucune représentation standard ne s'est dégagée pour le modèle textuel. Les approches actuelles misent leurs effort principalement sur la phase de recherche sur la base de cas et ne proposent pas d'avenue pour l'adaptation des solutions textuelles.

Remarque. Le CBR textuel diffère de l'approche structurelle dans laquelle les textes sont tout simplement des chaînes de caractères sans syntaxe ni sémantique précises, d'autant plus que cette dernière impose une structuration complète des attributs d'un cas. De l'autre côté, le modèle conversationnel, lui non plus, ne fait pas partie des approches textuelles, car sa phase préliminaire se limite à une comparaison par mots-clés de caractères, de courtes descriptions textuelles de problèmes. Durant la phase suivante, l'interaction avec l'utilisateur est guidée par une suite de questions et de réponses sans que les échanges lors de l'interaction ne fassent l'objet d'un traitement textuel quelconque. La langue y est donc utilisée uniquement dans le but de rendre les questions plus intelligibles aux usagers du système.

1.4.4 Performances

L'avantage des algorithmes CBR est le faible temps de calcul de la procédure d'apprentissage (il n'y a pas d'apprentissage à proprement parler) et, dans le cas des k plus proches voisins, la résistance à la présence d'incohérences dans les données. Les inconvénients sont les temps de prédictions qui peuvent être élevés (on doit parcourir tout l'ensemble d'apprentissage), le fait qu'ils nécessitent un ensemble d'apprentissage assez grand pour garantir une probabilité d'erreur faible, et, qu'ils sont assez inefficaces si les instances sont décrites par un grand nombre d'attributs dont peu sont significatifs, car ces attributs faussent les résultats.

1.5 Induction de règles

1.5.1 Principe

Les algorithmes basés sur l'induction des règles procèdent par une approche "diviser pour régner". Cette stratégie doit son nom au fait qu'elle construit la meilleure hypothèse (partie prémisse d'une règle) couvrant le plus possible d'exemples positifs de l'ensemble d'apprentissage et le moins possible (voire aucun) d'exemples négatifs, et ensuite, itère ce processus sur les mauvais exemples. Un ensemble de règles est construit ainsi pour chacun des concepts à apprendre. Le choix de la meilleure hypothèse est guidé par une évaluation numérique de la qua-

lité des hypothèses. En général la qualité est maximale pour l'hypothèse couvrant le maximum de contre-exemples. Une hypothèse est une conjonction de littéraux du langage de description des instances (dans le cas des attributs continus, un littéral n'est pas un couple $a = \nu$ mais $a \leq \nu$ où a est un attribut et ν une de ses valeurs possibles). La procédure de construction de la meilleure hypothèse est donc basée sur une exploration de l'espace des hypothèses dirigée du plus général vers le plus spécifique. Elle consiste à spécialiser progressivement l'hypothèse la plus générale par ajouts de littéraux. Le littéral ajouté à chaque spécialisation est celui qui maximise la qualité de l'hypothèse obtenue. La procédure se termine lorsque plus aucune spécialisation ne permet d'augmenter la qualité de l'hypothèse courante. Cette procédure peut être vue comme la construction d'un arbre de décision dont seule la branche la plus prometteuse (de meilleure qualité) serait effectivement construite. Le plus populaire des algorithmes d'induction des règles, CN2 [Clark and Niblett, 1989] retient à chaque étape les k meilleures spécialisations et produit k hypothèses. L'hypothèse finalement choisie est celle de meilleure qualité.

La procédure de prédiction du concept auquel appartient une instance consiste à déterminer les règles dont l'instance satisfait la partie prémisse. Si une seule règle est ainsi satisfaite, l'instance est considérée comme un exemple de la partie conclusion de la règle. Si aucune règle n'est satisfaite, le concept prédit par défaut est généralement celui auquel appartient la majorité des instances de l'ensemble d'apprentissage. Enfin, si plusieurs règles sont satisfaites, l'algorithme peut prédire le concept de la règle dont la partie prémisse couvre le plus d'exemples, ou dont la qualité de la partie prémisse est la plus grande.

1.5.2 Performances

Les algorithmes d'induction de règles ont une complexité théorique voisine de celle des algorithmes de construction d'arbres de décision (quadratique dans le nombre d'instances et de littéraux du langage de description), mais leur supériorité réside dans le fait qu'ils peuvent être très facilement étendus pour apprendre des hypothèses (donc des règles) représentées dans une logique du premier ordre. Ils sont en outre assez résistants à la présence d'incohérences dans les données d'apprentissage. Un inconvénient de ces algorithmes est que le nombre d'instances de l'ensemble d'apprentissage diminue au fur et à mesure que les règles sont apprises. Les dernières règles sont donc apprises sur des ensembles d'apprentissage qui peuvent être très réduits. Les hypothèses couvrant peu d'exemples (cinq ou moins) ont tendance à avoir une probabilité d'erreur élevée, mais la suppression de ces hypothèses augmente davantage la probabilité d'erreur globale de l'ensemble des hypothèses apprises.

1.6 Approche combinatoire

La comparaison empirique des différentes approches d'apprentissage a montré qu'aucune de ces approches n'est supérieure aux autres en termes de performances

dans tous les domaines d'application. Dès lors, lorsqu'on traite une certaine tâche, la question du choix de l'approche la plus appropriée se pose. Une manière de procéder dans ce cas serait d'essayer toutes les approches possibles à tour de rôle et de retenir la meilleure d'entre elles. Mais il est bien sûr évident que cette approche est beaucoup trop lourde et lente à mettre en oeuvre, vu le nombre élevé de solutions potentielles, d'autant plus que les algorithmes utilisés sont souvent assez difficiles à "régler". Une autre manière d'obtenir les meilleurs résultats possibles est d'utiliser des algorithmes qui combinent plusieurs méthodes d'apprentissage. Idéalement ces algorithmes devraient être capables d'égaliser les meilleures performances de leurs parents dans différents domaines. Et même si, malheureusement, ceci n'est souvent pas le cas, il existe tout de même un certain nombre d'algorithmes "combinatoires" qui arrivent à hériter des meilleures caractéristiques de leurs parents. L'un des résultats majeurs dans ce domaine est RISE¹.

1.6.1 Description de RISE

L'algorithme RISE [Domingos, 1996] a pour but de dépasser les limites de l'induction basée sur les instances et l'induction des règles, en unifiant ces deux approches. Comme les algorithmes d'induction des règles il construit un ensemble de règles, mais comme l'induction basée sur les instances, il fait appel à une mesure de similarité pour prédire le concept auquel appartient une instance. La procédure d'apprentissage procède par généralisation progressive d'un ensemble initial de règles. Cet ensemble de règles est simplement l'ensemble d'apprentissage lui-même. Le principe clé de cet algorithme est qu'il ne fait pas de différence entre les règles et les instances. Les instances ne sont que des règles particulières et les règles sont manipulées de la même manière que les instances.

1.6.2 Les règles et les instances

La partie prémisse d'une règle est une conjonction de littéraux (ou une hypothèse), et la conclusion, un concept. Un littéral peut être vu comme étant un test portant sur un attribut. Dans le cas d'un attribut discret c'est un test d'égalité du type $a = \nu$, et dans le cas d'un attribut continu c'est un test d'appartenance à un intervalle du type $\nu \leq a \leq \nu'$. Une instance est simplement une règle dont la partie prémisse est la conjonction des littéraux décrivant l'instance (les littéraux décrivant des attributs continus sont dégénérés en intervalles, i.e. $a = \nu$ est dégénéré en $\nu \leq a \leq \nu$), la partie conclusion est le concept auquel appartient l'instance. Dans la suite nous dirons qu'une règle r couvre une règle r' lorsque la partie prémisse de r couvre celle de r' et que les deux règles ont la même partie conclusion. Généraliser une règle consiste à généraliser les tests (ou littéraux) de sa partie prémisse ; généraliser un test portant sur un attribut discret consiste à remplacer la valeur de cet attribut par une valeur plus générale de la hiérarchie de l'attribut et, finalement ; généraliser un test portant sur un attribut continu

¹Les performances de RISE (Rule Induction from a Set of Exemplars) sont une fois sur deux meilleures que celles de la meilleure approche parmi PEBLS, CN2 (ses parents) et C4.5. Elles sont également presque toujours au moins équivalentes à celles de la moins performante. Les résultats ont été obtenus par [Domingos, 1994] sur 30 jeux de données standards.

consiste à élargir l'intervalle correspondant.

1.6.3 Mesure de similarité

L'évaluation de la similarité d'une règle et d'une instance est basée sur une mesure de la distance qui les sépare. Soit r une règle dont la partie prémisses est constituée des tests $t_1, t_2 \dots t_n$. Soit e une instance décrite par les valeurs $e_1, e_2 \dots e_n$. La distance $\delta(r, e)$ entre r et e est définie par :

$$\Delta(r, e) = \sum_{i=1}^n \delta^s(i),$$

où s est un paramètre ($s = 1, 2, 3 \dots$) de l'algorithme. La distance $\delta^s(i)$ entre le test de r portant sur l'attribut i et la valeur de l'attribut i dans e est définie par :

$$\delta^s(i) = \begin{cases} 0 & \text{si } t_i \text{ est vrai pour } e_i \\ SVDM(t_i, e_i) & \text{si } t_i \text{ porte sur un attribut discret} \\ \delta_{num}(i) & \text{si } t_i \text{ porte sur un attribut continu} \end{cases}$$

Si t_i est un test portant sur un attribut discret de la forme $a_i = \nu_i$, p est le nombre de concepts à apprendre, et q un paramètre entier de l'algorithme, alors :

$$SVDM(t_i, e_i) = \sum_{h=1}^p |P(c_h|\nu_i) - P(c_h|e_i)|^q.$$

$P(c_h|\nu_i)$ peut être approchée par le pourcentage d'instances de l'ensemble d'apprentissage possédant la valeur ν_i qui appartiennent au concept c_h . Si t_i est un test sur un attribut continu de la forme $\nu_{min} \leq a_i \leq \nu_{supp}$, $\delta_{num}(i)$ est définie par :

$$\delta_{num}(i) = \begin{cases} \frac{e_i - \nu_{supp}}{\nu_{i,max} - e_{i,min}} & \text{si } e_i \geq \nu_{max} \\ \frac{\nu_{min} - e_i}{e_{i,max} - \nu_{min}} & \text{si } e_i \leq \nu_{min} \end{cases}$$

La mesure de la distance entre une règle et une instance est telle que si la règle couvre l'instance, cette distance est nulle. Ainsi cette mesure unifie les notions de couverture des algorithmes d'induction des règles et de similarité de l'induction basée sur les instances.

1.6.4 Prédiction et apprentissage

Après avoir construit l'ensemble des règles, le but de la procédure de prédiction est, étant donné une instance e , de déterminer la règle r à appliquer pour prédire le concept auquel appartient cette instance. Pour cela, l'algorithme utilise la mesure de similarité $\Delta(r, e)$. Lorsque plusieurs règles sont similaires RISE utilise la règle de plus grande qualité de Laplace. Si n_e est le nombre d'exemples que la règle couvre, n_c le nombre de contre-exemples qu'elle couvre et p le nombre de concepts à apprendre, la qualité de Laplace est définie par :

$$Q_L = \frac{n_e + 1}{n_e + n_c + p}.$$

1.7 Classifieur Naïf de Bayes

Le classifieur naïf de Bayes¹, ou NBC, est capable de simplifier de manière significative la tâche d'apprentissage en considérant les attributs des différents concepts indépendants. Même si cette hypothèse simplificatrice est rarement vérifiée dans la réalité, NBC permet néanmoins de rester très souvent compétitif par rapport aux classifieurs plus sophistiqués.

1.7.1 Théorie

Soit $C = \{c_1, c_2, \dots, c_m\}$ l'ensemble des concepts à apprendre et $\mathbf{X} = (a_1, a_2, \dots, a_n)$ le vecteur décrivant les instances par leurs littéraux a_i . Le classifieur, de manière générale, va associer à chaque concept c_i une fonction discriminante $f_i(\mathbf{x})$, $i = 1, 2, \dots, m$, et choisir comme résultat de prédiction le concept qui maximise la fonction discriminante pour l'instance \mathbf{x} donnée. Le classifieur de Bayes, appelé également classifieur optimal, utilise comme fonction discriminante la probabilité conditionnelle d'avoir comme prédiction le concept c_i , étant donné un vecteur instance \mathbf{x} , noté $P(C = c_i | \mathbf{X} = \mathbf{x})$. Le théorème de Bayes² permet de réécrire cette expression de la manière suivante :

$$P(C = c_i | \mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x} | C = c_i)P(C = c_i)}{P(\mathbf{X} = \mathbf{x})}.$$

Le problème de cette approche est qu'il est assez difficile d'estimer la valeur de $P(\mathbf{X} = \mathbf{x} | C = c_i)$ sur base de l'ensemble d'apprentissage. Ainsi pour des raisons de simplification, on va considérer que les attributs sont conditionnellement indépendants étant donné un concept, autrement dit que :

$$P(\mathbf{X} = \mathbf{x} | C = c_i) = \prod_{j=1}^n P(X_j = a_j | C = c_i).$$

La procédure d'apprentissage NBC consiste donc à calculer et à mémoriser l'ensemble des probabilités $P(a_j | C)$. Elle est incrémentale et peu coûteuse en temps de calcul. La procédure de prédiction est elle aussi très facile à mettre en oeuvre et permet de déterminer le concept C_{max} étant donnée une instance décrite par des attributs a_j :

$$C_{max} = \underset{i}{\operatorname{argmax}} \prod_{j=1}^n P(X_j = a_j | C = c_i)P(C = c_i).$$

Nous avons ignoré le dénominateur $P(\mathbf{X} = \mathbf{x})$ de la fraction, puisqu'il est identique pour tous les concepts et qu'il n'a dès lors aucune influence sur la recherche de la valeur maximale.

¹Proposé par [Duda and Hart, 1973]

² $P(A|B)P(B) = P(B|A)P(A)$

1.7.2 Performances

Les performances de NBC sont particulièrement intéressantes lorsque les attributs des instances à apprendre sont complètement non-corrélés (ce qui est tout à fait logique compte tenu des hypothèses simplificatrices propre à l'algorithme) mais, plus étonnamment, l'algorithme arrive à produire des résultats comparables à ceux de C4.5, CN2 ou PEBLS même pour des données corrélées. En effet, [Rich, 2001] montre que les performances de NBC ne dépendent pas directement du degré de corrélation entre les attributs des instances.

1.8 Apprentissage des habitudes

1.8.1 Définitions

Définition 9 (Action) *Une action est une intervention quelconque de l'utilisateur sur l'environnement (la pression d'un bouton, la saisie d'un texte, etc.).*

La séquence de toutes les actions accomplies au cours d'une session de travail est appelée *la trace*, une séquence d'actions étant considérée comme un ensemble chronologiquement ordonné d'actions effectuées de manière consécutive.

Définition 10 (Tâche répétitive) *Lorsqu'une séquence d'actions quelconque se répète dans la trace, nous parlerons de différentes occurrences d'une tâche répétitive.*

Ces occurrences ne sont pas toujours strictement identiques (définition très restrictive). Lors d'une session de travail l'utilisateur peut soit faire des erreurs de manipulation, soit encore appliquer un même traitement à des objets différents. . . Dans ce cas les tâches répétitives seront composées des mêmes actions $A_i(\nu_1, \nu_2 \dots \nu_p)$, mais avec des attributs aux valeurs différentes.

Définition 11 (Situation) *Une situation S est la séquence des s actions (s est un paramètre de l'algorithme d'apprentissage) précédant une action A donnée.*

Nous dirons que S est la situation dans laquelle l'utilisateur a accompli A (SA). S peut être représentée par la formule logique $A_1 \wedge A_2 \wedge \dots \wedge A_n$, noté $S(A_1, A_2 \dots A_n)$. Les s dernières actions de la trace représentent *la situation courante*.

Définition 12 (Modèle de situation) *Un modèle de situation est une séquence ordonnée ou non d'actions.*

Un modèle de situation est défini par rapport aux situations dans lesquelles l'utilisateur a accompli une tâche répétitive particulière. Intuitivement, un modèle de situation exprime des similarités entre ces situations et donc les *caractérise*. Un modèle de situation ordonné est noté $[A_1, A_2 \dots A_i]$. Un modèle de situation non ordonné est noté sous forme ensembliste. Ces modèles permettent d'exprimer le fait que les situations ont en commun des actions ou la valeur de certains attributs mais dans un ordre différent. Un tel modèle est noté $\{A_1, A_2 \dots A_i\}$.

Exemple. Soit la trace suivante : $\dots \overbrace{A_1 a A_2(x, y) b c A_3}^{\text{Situation}_1} T^1 \dots \overbrace{A_1 d A_2(x, z) e f A_3}^{\text{Situation}_2} T^2 \dots$
 Le modèle dans lequel la tâche T a été accomplie est $[A_1 * A_2(x, *) ** A_3]$.¹

Définition 13 (Habitue) Une habitude est une paire (M, T) , où T est une tâche répétitive et M un modèle de situation (abstrait ou non ordonné abstrait) qui couvre au moins une des situations dans lesquelles T a été accomplie.

1.8.2 Principe

Une interface hybride basée sur l'apprentissage des habitudes de l'utilisateur fait appel à deux procédures : la *procédure d'apprentissage des habitudes* et la *procédure de prédiction*. La procédure d'apprentissage comporte deux phases. La première consiste à détecter les tâches répétitives de la trace (les séquences d'actions identiques à elles-mêmes peuvent être aisément détectées au moyen d'un algorithme de recherche de répétitions) et la seconde à construire, pour chacune de ces tâches, un ou plusieurs modèles de situation qui caractérisent les situations dans lesquelles l'utilisateur les a accomplies. La phase de prédiction consiste à détecter, au moyen des habitudes apprises, les situations dans lesquelles l'utilisateur est sur le point d'accomplir une tâche répétitive pour lui proposer de rejouer automatiquement la tâche ainsi prédite (illustré sur la figure 1.3).

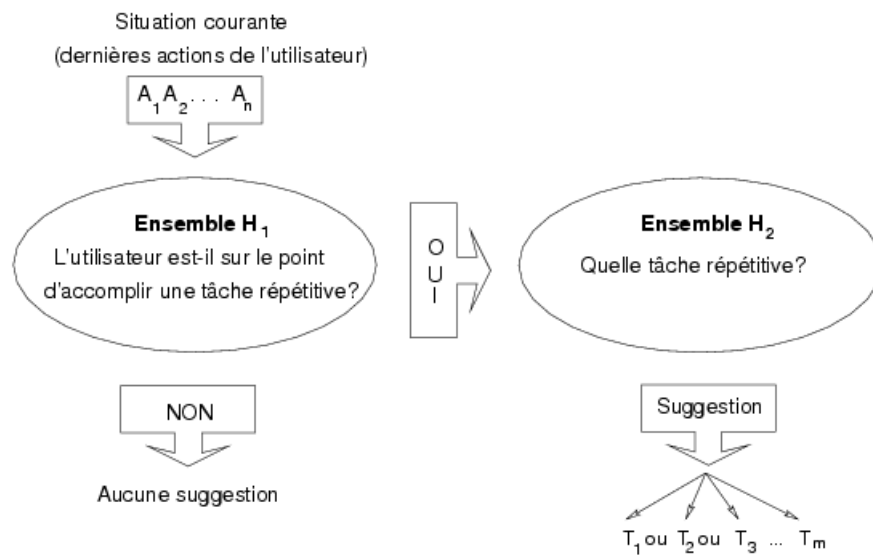


FIG. 1.3 – Processus de prédiction

¹Les lettres de A_1 à A_4 ainsi que les lettres minuscules symbolisent des actions. Les lettres T^1 et T^2 représentent des occurrences d'une tâche répétitive T .

1.9 Conclusion

Les algorithmes d'apprentissage inductif visent à généraliser un concept sur base d'exemples et de contre-exemples de ce concept, de manière à pouvoir ultérieurement classer des observations décrites par un vecteur de valeurs d'attributs à l'aide d'un ensemble d'apprentissage constitué par des observations antérieures.

La plupart des algorithmes d'apprentissage inductif procèdent à une exploration partielle d'un espace de concepts hypothétiques, appelés hypothèses. L'algorithme C4.5 est l'un des plus répandus parmi ces algorithmes grâce en grande partie à son faible coût en temps de calcul. Il procède à une exploration des hypothèses dirigée des hypothèses les plus générales vers les hypothèses les plus spécifiques (de la racine aux feuilles). Un autre algorithme explorant de manière similaire l'espace des hypothèses est CALM. Ses inconvénients principaux sont sa complexité théorique élevée et la difficulté de régler correctement ses nombreux seuils (paramètres de l'algorithme). Néanmoins, il a l'avantage important de tolérer des données bruitées, ce qui le rend nettement préférable à son ancêtre CEA tout aussi lent et incapable de gérer du bruit. Les algorithmes d'induction basés sur les instances (CBR) ne construisent pas d'hypothèses du tout et ont de ce fait une complexité d'apprentissage quasi nulle et une complexité de prédiction assez élevée (parcours linéaire des exemples mémorisés en les comparant à l'instance cible). De l'autre côté, les algorithmes d'induction de règles, dont le plus populaire est CN2, ont une complexité voisine de celle des arbres de décision et sont assez résistants au bruit. Un inconvénient de ces algorithmes est que le nombre d'instances de l'ensemble d'apprentissage diminue (et donc la qualité des hypothèses aussi) au fur et à mesure que les règles sont apprises. L'algorithme RISE dépasse les limites de ces deux approches, en les unifiant. Lui aussi construit un ensemble de règles, mais fait appel ensuite à une mesure de similarité pour prédire le concept auquel appartient l'instance cible. RISE est relativement peu coûteux en temps de calcul et possède une faible probabilité d'erreur. Le classifieur naïf de Bayes n'explore pas d'espace d'hypothèses. Cet algorithme probabiliste peu coûteux en temps de calcul fait l'hypothèse naïve sur l'indépendance conditionnelle des valeurs des attributs des instances, mais garde néanmoins une faible probabilité d'erreur.

La dernière section présente une manière très élégante de résoudre un problème auquel à première vue les algorithmes d'apprentissage inductif de concepts n'apportent pas de solutions. En effet, dans le cadre des interfaces humain-machine, on considère généralement une observation comme étant une suite d'actions, et logiquement le concept comme une seule action - l'action suivante. L'idée de l'apprentissage des habitudes est de décomposer le problème en plusieurs sous-problèmes : effectuer un pré-traitement de l'historique, afin de déceler les tâches répétitives, qui ensuite, seront considérées comme étant des concepts ; apprendre les corrélations entre ces tâches répétitives et les situations dans lesquelles elles ont été effectuées ; prédire des suites d'actions sophistiquées sur base de la situation courante. Cette approche combine plusieurs algorithmes d'apprentissage inductif.

Chapitre 2

BIGRE

Ce mémoire s'inscrit dans le cadre d'une étude appliquée au projet bioinformatique BIGRE; un projet de conception et de modélisation d'une architecture intégrée basée sur les concepts GRID¹. La phase de conception du projet étant actuellement quasi terminée, nous en profitons dans la suite de notre travail, d'abord pour présenter une vue globale sur le futur système et son fonctionnement et, ensuite pour intégrer nos propres idées et concepts à l'ensemble de ceux déjà adoptés.

Le Workflow² est un outil permettant la construction à un haut niveau d'abstraction de diagrammes représentant des enchaînements graphiques de tâches biologiques. L'élément central de ce chapitre est la conception du Workflow Intelligent. Il s'agit là d'une extension du simple Workflow consistant essentiellement en l'ajout *d'une barre des prévisions* à celui-ci. Cette barre des prévisions constitue une sorte d'assistant intelligent susceptible de faciliter considérablement la tâche aux utilisateurs en devinant leurs intentions lors des expérimentations dans le laboratoire biologique virtuel qu'est le Workflow. Pour atteindre notre but, nous proposons un système *d'agents* indépendants capables de communiquer entre eux au travers de différentes sources d'informations et mettant en oeuvre les différents algorithmes d'apprentissage introduits précédemment. Dans notre analyse nous plaçons un accent particulier sur la justification des différents choix algorithmiques effectués.

L'analyse de la conception de la couche intelligente de BIGRE est réalisée en six sections : la première et la deuxième présentent les objectifs officiels du projet et procèdent à une première description des différentes composantes du système ; la troisième section lance la réflexion sur la place de l'intelligence au sein du système et introduit la notion du Workflow Intelligent ; la quatrième décrit de manière détaillée son fonctionnement ; et finalement la cinquième section formalise sous forme de cahier des charges les différentes tâches à effectuer pour rendre le Workflow Intelligent opérationnel. La conclusion fournit un bref résumé du chapitre.

¹General Repository for Interactions Datasets.

²Le terme générique "workflow" (flux des tâches, *fr.*) s'applique à toute modélisation de processus d'activité. Pour plus de renseignements sur la théorie de la gestion des workflows se référer au livre de [van der Aalst and van Hee, 2002].

2.1 Le projet BIGRE

Le projet BIGRE étant au coeur même de ce mémoire, une description globale de celui-ci, aussi brève soit-elle, s'impose. Ainsi, dans cette section nous commencerons par fournir sa présentation et ses objectifs officiels. Notons que les informations concernant la description et le fonctionnement du système BIGRE présentées dans la suite de ce chapitre proviennent des sources internes du projet et que, pour de raisons évidentes, présenter toutes ces informations de manière détaillée dans le cadre de notre travail serait tout simplement irréalisable et tout compte fait inutile.

2.1.1 Présentation

Le projet BIGRE vise à concevoir, développer et déployer une architecture distribuée de médiateurs sur Internet dans le but de fédérer des ressources bio-informatiques, telles que des bases de données ou des services informatiques, et de les proposer de manière intégrée au travers d'une interface homogène à différents profils d'utilisateurs¹. Ce projet se situe à la confluence de deux mouvements technologiques très actuels. D'abord, propre à la biologie, le recourt de plus en plus important à des applications logicielles, malheureusement extrêmement hétérogènes (pour stocker, lire et interpréter les données biologiques), ensuite, propre à l'informatique, le développement par les constructeurs de plate-formes d'intégrations² facilitant l'interopérabilité des applications logicielles hétérogènes existantes (processeurs, systèmes d'exploitation, bibliothèques logicielles et langages de programmation). La biologie offre ainsi un contexte d'expérimentation et d'application idéal pour les récentes solutions intégratives proposées par les différents constructeurs de logiciels et les organismes de standardisation.

2.1.2 Objectifs

De manière plus détaillée, les objectifs du projet sont :

- Fournir un outil permettant de mailler automatiquement et de manière dynamique un vaste nombre de services informatiques (dans le domaine de la bio-informatique) au sein d'une fédération intégrée. Celle-ci permettra l'accès transparent à un nombre accru de services (*legacy* et *futurs*), hétérogènes, distribués, complémentaires et/ou concurrents. Ces services intégrés seront accessibles via des interfaces utilisateur intelligentes produites automatiquement à partir de métadonnées de description des services telles que, par exemple, UML (Unified Modeling Language) et XML (eXtensible Markup Language).
- Répondre à la demande des laboratoires (publiques et privés) concernant la dissémination et l'accès à l'information génomique.

¹Les différents profils d'utilisateurs restent à définir.

²Citons comme exemples de telles plate-formes CORBA (Common Object Request Broker Architecture) et les Services Web, devenus plus des standards que des produits.

- Réaliser l'état de l'art en matière de plate-formes intégratives et, une fois le choix effectué, développer un modèle d'architecture compatible avec les standards actuels¹ et extrapolable à d'autres domaines.
- Mettre au point une interface d'aide à la décision permettant aux utilisateurs, et ceci de la manière la moins contraignante qui soit, de découvrir parmi les multiples applications logicielles répondant à leurs besoins celles qui leur conviennent le mieux.
- Procéder à un déploiement partiel de l'architecture dans des laboratoires européens.

2.2 Le système BIGRE

L'objectif de cette section est de présenter une brève description fonctionnelle du système BIGRE en termes techniques². Cette description est un premier pas vers l'analyse d'une éventuelle introduction de l'intelligence artificielle dans le système (le but ultime de notre travail), car elle permet d'introduire et de situer les différentes composantes du système au sein de celui-ci en montrant clairement leurs interactions mutuelles. En effet, c'est seulement lorsqu'une telle vue globale du système sera dressée, qu'il sera possible de réfléchir à la place de *l'intelligence* au sein du système ainsi qu'aux conséquences que cette intervention aura sur son fonctionnement. Puisque l'intelligence du système se situera au niveau de l'Interface Humain-Machine, c'est clairement sur cet aspect-là du système qu'il faut insister dans notre étude. C'est ainsi très logiquement qu'on va commencer par présenter l'interface du système (encore inexistante en réalité) et introduire la notion de *plug-ins*. Ceci dit, la vue générale sur l'architecture du système reste tout de même le point de départ de notre description, car celle-ci est indispensable à une bonne compréhension de la suite de l'analyse.

2.2.1 Architecture générale

La meilleure manière de présenter l'architecture du système est d'introduire d'abord les différents acteurs impliqués dans son fonctionnement et d'illustrer ensuite leurs rôles respectifs par un exemple réel de leur utilisation.

2.2.1.1 Découpe en sous-systèmes

L'architecture de BIGRE comprend trois types de sous-systèmes³ : les *clients*, les *médiateurs* et les *services*. Un *client* est l'application utilisateur principale. C'est via un *client* qu'un utilisateur interagit avec le système, qu'il utilise des *services*. Un *service* est une des applications de traitements de données mis à dispositions des utilisateurs. Pour utiliser un *service*, un *client* interagit directement

¹Définis par : W3C (WWW Consortium), OMG (Object Management Group), etc.

²Rappelons encore une fois qu'une description exhaustive du système dans le cadre de cette étude est impossible et inutile. Dès lors, nous ne décrirons, en détails ou pas (selon les cas), que les parties du système qui concernent notre travail.

³L'architecture BIGRE est basée sur le mémoire de fin d'études de [Dallons and Buyle, 2003].

avec lui et toujours à son initiative. Pour permettre ces interactions, BIGRE doit fournir certaines fonctionnalités telles que la découverte dynamique de *services*, la sécurisation des communications entre *clients* et *services*, l'“accounting” des utilisations d'un *service* ou encore la gestion de la QoS attendue lors de l'utilisation de *services*. Ce sont les *médiateurs* qui assurent ces fonctionnalités.

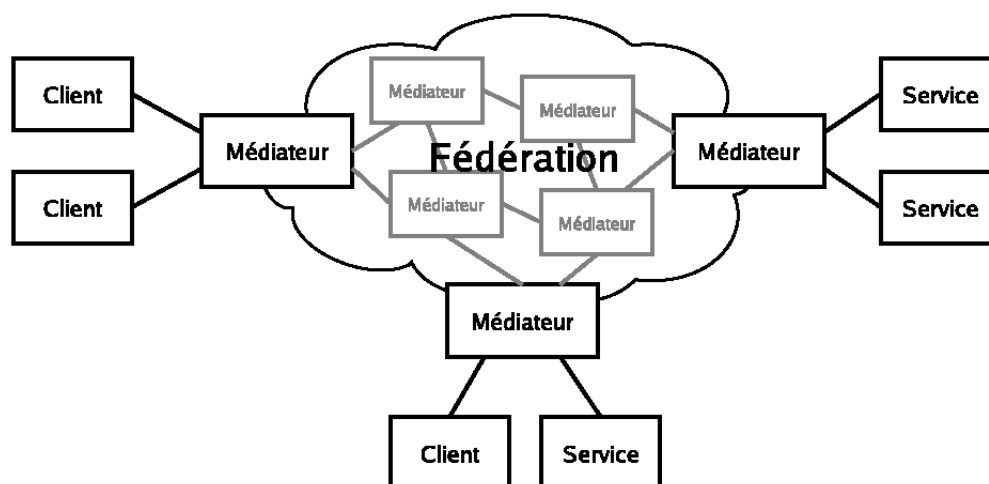


FIG. 2.1 – Architecture générale de BIGRE.

Les *médiateurs* sont organisés en une fédération et forment un système distribué. Pour assurer l'ouverture du système, la gestion de la fédération est assurée collectivement par les *médiateurs* eux-mêmes. Il s'agit d'un système distribué autogéré plutôt que reposant sur une forme d'administration centralisée et/ou fortement contrôlée. Pour les *clients* et les *services* l'organisation des *médiateurs* en fédération reste transparente, ils ne communiquent qu'avec un seul *médiateur*. Si chaque *médiateur* communique directement avec tous les autres il faut qu'en permanence chaque *médiateur* ait connaissance de tous les autres ce qui est difficile à garantir dans un système sans gestion centralisée. Dans BIGRE, chaque *médiateur* ne communique directement qu'avec un nombre restreint d'autres *médiateurs*. Pour assurer les communications indirectes entre *médiateurs* nous aurons recours à un système de communications distribué tel que ceux rencontrés dans les systèmes Peer-to-Peer. Notons que chaque médiateur est composé des éléments suivants :

Services Directory

Le *Services Directory* est l'annuaire des *services* présents au sein de la fédération et est accessible par les *clients*. Le *Services Directory* fournit deux interfaces ; celle permettant la gestion de l'information qu'il contient et une interface de recherche de *services* et de consultation des informations attachées aux *services* qu'elle contient.

Ontologies Repository

L'*Ontologies Repository* assure le maintien et la propagation des ontologies dans la fédération. Il offre à tous composants qui pourraient en avoir besoin

une interface d'accès aux ontologies connues et acceptées par le *médiateur*.

Security Manager

Les *médiateurs* assurent le rôle d'autorité de certification dans le cadre de la sécurisation de communications entre *clients* et *services*. A cette fin le *Security Manager* présente la ou les interfaces nécessaires aux fonctionnalités que ce rôle impose aux *médiateurs*.

Accounting Manager

L'*Accounting Manager* permet une gestion à posteriori de l'utilisation des *services*, principalement à des fins de refacturation. Pour ce faire, il offre aux *services* une interface leur permettant de remettre des rapports d'utilisations après qu'ils aient été utilisés par un *client*. Il offre également, aux fournisseurs de *service*, une interface permettant l'accès aux informations concernant l'utilisation de leurs *services* par les *clients*.

QoS Evaluator

Assure la mise à jour de la QoS attendue pour un *service*.

Federation Messenger

Certaines fonctionnalités des autres composants du *médiateur* nécessitent la collaboration de plusieurs *médiateurs* (de leurs composants) dans la fédération, il est donc nécessaire de permettre la communication entre *médiateurs*. C'est ce que permet le *Federation Messenger* en offrant aux composants d'un *médiateur* la possibilité d'échanger des messages avec les composants des autres *médiateurs* de la fédération.

La nature des fonctionnalités assurées par les *médiateurs* impliquent qu'ils assurent également une part de la gestion des *clients* et des *services*. Pour faciliter cette gestion, un *client* ou un *service* est rattaché à un seul *médiateur*, il n'interagit qu'avec ce *médiateur* et aucun autre. Pour sécuriser les communications entre *clients* et *services*, BIGRE utilise une infrastructure à clefs publiques dans lequel les *médiateurs* assurent vis à vis de leurs *clients* et *services* le rôle d'autorité de certification. Une partie de ces caractéristiques du système sont représentées sur la figure 2.1. On peut y voir que les *médiateurs* sont organisés en fédération où chaque *médiateur* n'est "lié" qu'à une fraction de l'ensemble de tous les *médiateurs*. On voit aussi qu'un *client* ou un *service* n'est "lié" qu'à un seul *médiateur* et qu'il n'a pas connaissance des autres. Pour plus de clarté, le fait que chaque *client* a la capacité d'interagir avec chaque *service* n'est pas représenté.

2.2.1.2 Dynamique des sous-systèmes

Les figures 2.2 et 2.3 présentent les principales interactions entre sous-systèmes lorsqu'un *client* utilise un *service*¹. Pour simplifier la représentation, la fédération de *médiateurs* y est représentée comme un tout, lorsque le *client* ou le *service* interagit avec son *médiateur* on le représente par une interaction entre lui et la fédération. De même, le diagramme ne présente qu'une seule interaction "appel d'opération" entre le client et le service. Il est cependant envisageable d'avoir plus

¹La deuxième figure détaille les composantes impliquées dans les médiateurs.

d'une seule interaction de ce type. Par exemple, lorsque le traitement rendu par le service nécessite un contrôle de son exécution le client appelle plusieurs opérations du service ce qui donne lieu à plusieurs interactions "appel d'opération".

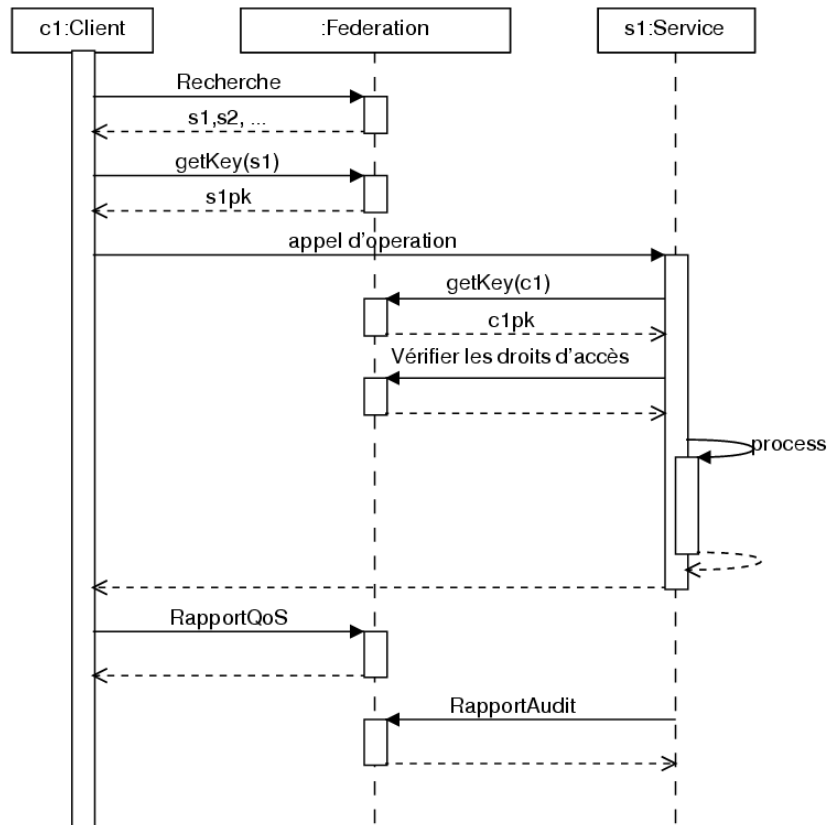


FIG. 2.2 – Diagramme de séquence général pour l'utilisation d'un service.

Le *client* commence par effectuer une recherche de *services*. Il reçoit en réponse un ensemble de références de *services* dont celle de s_1 . Le *client* fait appel à s_1 via une communication sécurisée. Si il ne la possède pas encore, il doit donc récupérer auprès de son *médiateur* l'information nécessaire à cette sécurisation, la clef publique de s_1 . Il effectue ensuite l'appel d'une ou plusieurs opérations de s_1 pour réaliser le traitement offert par s_1 . Pour permettre la sécurisation de leurs communications, s_1 a lui aussi besoin de la clef publique de c_1 . Si il ne la possède pas il doit donc la récupérer auprès de son *médiateur* dès que c_1 entre en interaction avec lui. s_1 vérifie ensuite, toujours auprès de son *médiateur*, que c_1 a bien le droit de faire appel à lui. Une fois ces interactions réalisées, s_1 peut exécuter l'opération et en retourner le résultat à c_1 . Après la réalisation du traitement, soit après l'exécution d'une ou plusieurs opérations, c_1 fait un rapport de la qualité de service rencontrée lors de l'utilisation de s_1 auprès de son *médiateur*. De même s_1 fait un rapport sur la consommation de ressources engendrée suite à son utilisation par c_1 .

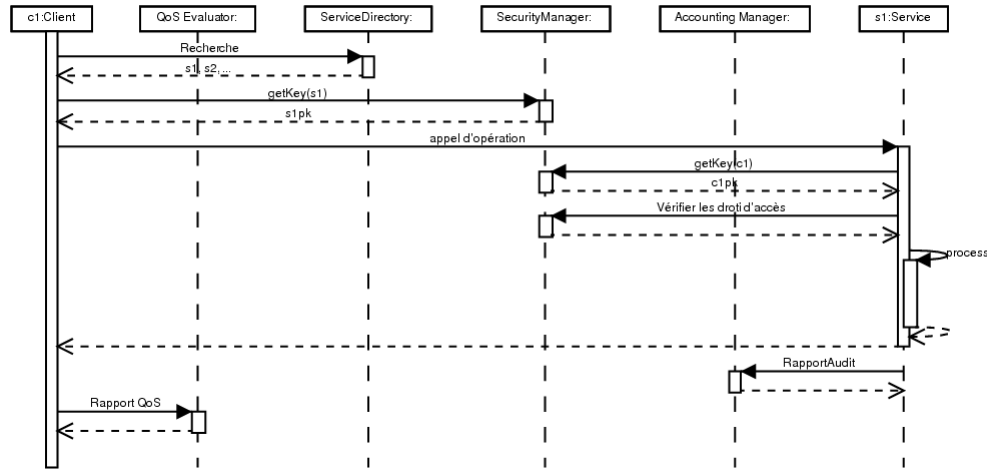


FIG. 2.3 – Diagramme de séquence détaillé pour l'utilisation d'un service.

2.2.2 Notion de *Plug-ins*

Le système BIGRE se veut être un système entièrement générique. Nous entendons par là que l'interface utilisateur doit être totalement adaptative. Le système BIGRE est conçu pour permettre au client de dialoguer avec des services quelconques de manière transparente. Pour cela le système doit pouvoir fournir au client une interface d'appel aux services et ce quels qu'ils soient. Pour atteindre ce but l'ensemble des données manipulées par un service¹ seront définies dans *une ontologie de données*. Pour chaque type de données de l'ontologie au moins un *visualiseur*² sera toujours disponible de manière à assurer que le client pourra toujours visualiser tout type de données. Cependant certains types de données peuvent nécessiter un rendu plus travaillé de manière à améliorer l'utilisabilité du service; ces types de données-là auront plusieurs visualiseurs. Le client peut alors choisir le visualiseur qu'il préfère parmi ceux dont il dispose pour un même type de données. Pour qu'un tel système puisse être mis en place, son architecture doit être *modulable*. C'est à dire que le système doit être évolutif au travers des téléchargements de visualiseurs.

La suite des types de données associée à l'entrée d'un service, définit l'IHM qui sera l'interface du service. L'IHM pourra être décrite par un document listant l'ordonnancement des visualiseurs (donc des types de données). Cependant une telle IHM doit aussi pouvoir être accompagné d'un *code mobile* pouvant décrire le comportement dynamique d'une IHM (les réactions de l'interface face aux différents événements utilisateurs) et permettant de doter éventuellement l'IHM d'un *module d'intelligence artificiel*. Nous définissons donc comme *plug-in* le document associé à un service, décrivant l'IHM, et éventuellement, accompagné d'un code mobile.

¹Il s'agit des données fournies en entrée aux services et de ceux renvoyées en sortie.

²Interface visualisant des données et intégrant éventuellement une zone de saisie.

C'est au travers du téléchargement de *plug-ins* que l'interface cliente s'adaptera automatiquement aux futurs services. Le système (coté client) peut alors être vu comme étant composé d'un noyau de base intégrant :

- Le coeur du système graphique.
- L'API de dialogue avec les services.
- Le module de téléchargement des plug-ins.

Le noyau du système n'intègre de base aucun visualiseur, ce qui revient à dire qu'il ne connaît aucun type de données. De l'autre côté celui-ci est capable de se connecter à un annuaire d'ontologie de la fédération BIGRE, découvrir de nouvelles données et de télécharger pour chaque donnée inconnue le visualiseur nécessaire. Dans le même ordre d'idées, le noyau du système ne connaît lui non plus a priori aucun service (donc aucune IHM de service), mais il est capable de se connecter à un annuaire de services disponibles au sein de la fédération BIGRE et de découvrir de nouveaux services (et leurs IHM). C'est ce scénario de découvertes dynamiques de services et de téléchargements dynamiques de visualiseurs de données qui confère au système BIGRE toute sa généralité.

2.2.3 IHM du système

L'IHM de la partie cliente du système sera constituée principalement par les différents plug-ins. C'est en effet l'ensemble des services disponibles au sein de la fédération qui définira l'aspect extérieur final du système. Bien entendu, pour le moment, compte tenu de l'état d'avancement du développement de BIGRE, il est encore trop tôt pour entamer la discussion concernant les futures IHM de la plupart des services. Néanmoins il est assez utile de présenter les IHM des services actuellement disponibles séparément (par exemple à travers le Web), puisque les futures IHM du client auront très probablement beaucoup de points communs avec elles. Ces services Web bio-informatiques actuellement disponibles sont représentés typiquement par des formulaires html acceptant les données biologiques en entrée sous forme de texte ASCII (introduit directement ou par le biais d'un fichier) et faisant ensuite appel aux scripts nécessaires pour effectuer la tâche concernée avec les paramètres d'exécution sélectionnés. Un exemple d'un tel service est présenté sur la figure 2.4¹. Nous nous limitons donc, sans rentrer dans les détails, à supposer qu'un grand nombre de plug-ins BIGRE auront une IHM assez similaire à celle-ci.

2.2.4 Introduction du Workflow

2.2.4.1 Exemple réel de workflow

Un *workflow*² est typiquement une suite de traitements informatiques appliqués aux données biologiques. A chaque étape, une application informatique correspondante à la tâche à effectuer, reçoit des données en entrée et produit des résultats

¹Blast (Basic Local Alignment Search Tool) est une méthode heuristique destinée à trouver les alignements optimaux locaux de meilleurs scores entre la séquence requête et la banque.

²Le Workflow est une application. Le workflow est un diagramme.

Blast2 at GeneStream

Choose program to use and database to search:

Program Database

Enter sequence below in [FASTA](#) format

Or load it from disk

The query sequence is [filtered](#) for low complexity regions by default.

[Filter](#) Low complexity Mask for lookup table only

[Expect](#) [Matrix](#) Perform ungapped alignment

[Query Genetic Codes \(blastx only\)](#)

[Other advanced options:](#)

[Graphical Overview](#) [Alignment view](#)

[Descriptions](#) [Alignments](#) [Color schema](#)

FIG. 2.4 – L’IHM du service Web Blast.

en sortie. Ces résultats sont passés en entrée à l’application suivante ou bien stockés comme résultats définitifs s’il n’y a plus de traitements à effectuer. Dans un simple exemple réel de workflow, nous présentons un protocole constitué d’une série d’analyses allant d’une séquence nucléotidique inconnue à la caractérisation de la protéine codée, notamment sa famille et les significations biologiques. Il s’agit d’un workflow basique constitué de la séquence linéaire suivante d’actions :

- Traduction de la séquence.

Site	INFOBIOGEN
URL	http://www.infobiogen.fr/services/analyseq/cgi-bin/traduc.in.pl
Paramètres	Séquence au format FASTA > seq1 CGCCCTGCAGGCGGACGGACGCTCCTGCGGGCTGCCGGCCGAGCACCCGTG CCACCAACTGCGAGCACTTCTGCCACCTCCACGGGCTGGCAACTACACGT GCATCTGCGAGGACAGCTACCAGCTGGCCGCCGCCGACC Traitement Phase Ouverte Potentielle (Met : codon STOP) Mode Verbeux Actif Région de la séquence à traiter : 1 à 0 (séquence entière) Code Génétique Standard Edition des Acides Aminés : Code à une lettre

- **Tri de séquences.**

Les séquences contenant des codons STOP (notés) ne sont pas conservées.

- **Recherche de motifs.**

Site	
URL	http://bioinf.man.ac.uk/dbbrowser/OWL/QuizOWLSQ.html
Paramètres	Qualifier : /Info Sequence fragment (successivement) : ALQADGR RPAGGRT PCRRTDA

- **Recherche d'empreintes de famille.**

Site	EXPASY - ScanProsite
URL	http://us.expasy.org/cgi-bin/scanprosite
Paramètres	Sequence : ALQADGRSCGLPAEHPCHQLCEHFCHLHGLGNYTCICEAGYQLAAD Scan Patterns - Profiles - Rules : Checked

- **Recherche des caractéristiques physicochimiques.**

Site	PBIL - Network Protein Sequence Analysis
URL	http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=/NPSA/npsa_pcprof.html
Paramètres	Windows Size : 7 Sequence : ALQADGRSCGLPAEHPCHQLCEHFCHLHGLGNYTCICEAGYQLAAD

- **Prédiction de la structure secondaire.**

Site	PBIL - Network Protein Sequence Analysis
URL	http://npsa-pbil.ibcp.fr/cgi-bin/npsa_automat.pl?page=/NPSA/npsa_mlrc.html
Paramètres	Output Wide : 70 Sequence : ALQADGRSCGLPAEHPCHQLCEHFCHLHGLGNYTCICEAGYQLAAD

2.3 Interfaces Intelligentes

Après avoir passé en revue les différentes interfaces rencontrées dans l'environnement BIGRE, nous allons maintenant nous concentrer sur les méthodes à mettre en oeuvre afin de rendre l'utilisation de ces interfaces encore plus efficace pour les utilisateurs. Nous commençons pour cela par introduire la notion d'interfaces dites "intelligentes" et nous appliquons ensuite cette notion au cas précis du Workflow.

2.3.1 Définitions et objectifs

Les Interfaces Intelligentes (*Intelligent User Interfaces*, IUI) sont issues du domaine de l'Interaction entre l'Homme et la Machine (IHM) et dans la littérature sont également connues sous le nom des *Interfaces Adaptatives*¹. La définition du terme *Interfaces Intelligentes* consiste principalement en la définition du mot-clé *Intelligence*. Il existe une grande multitude de définitions de ce mot, et la plupart de ces définitions mentionnent la capacité d'adaptation (apprendre et être capable de gérer de nouvelles situations inconnues auparavant), la capacité de communiquer et la capacité de résoudre des problèmes. De l'autre côté, une interface *normale* est définie comme une méthode de communication entre un utilisateur humain et la machine. Ainsi, en étendant cette définition, nous pouvons dire qu'une interface *intelligente* utilise une sorte de technologie intelligente qui permet la mise en oeuvre de cette communication entre l'humain et la machine. Les IUI sont donc des interfaces qui s'adaptent à l'utilisateur, qui communiquent avec lui et qui sont capable de résoudre des problèmes à sa place. La propriété principale des IUI est le fait qu'elles sont conçues pour *améliorer la communication* entre l'utilisateur et la machine. La technique utilisée pour atteindre cet objectif importe peu du moment qu'elle correspond à la définition d'une technique intelligente. Les principales techniques intelligentes peuvent être regroupées en quelques grandes catégories :

- **Modélisation des utilisateurs.** Il s'agit des techniques qui permettent au système de générer et de maintenir *une base de connaissance* relative aux utilisateurs sur base de leurs *inputs* (actions).
- **Adaptation aux utilisateurs.** Ces techniques permettent d'adapter l'interaction Homme-Machine aux différents utilisateurs et aux différentes situations rencontrées lors de l'utilisation du système.
- **Explication des résultats.** Le but des techniques de cette catégorie est de présenter à l'utilisateur les résultats obtenus par le système de la manière la plus compréhensible (naturelle) possible.

La mise en oeuvre d'une (ou de plusieurs) des techniques citées plus haut permet la *personnalisation* du système, ce qui rend son utilisation beaucoup plus flexible. Pour pouvoir personnaliser l'interaction avec un utilisateur donné, les IUI ont souvent recours à la représentation de cet utilisateur. Des modèles d'utilisateurs enregistrent les comportements des utilisateurs, leurs performances et capacités. Ainsi, de nouvelles connaissances concernant un utilisateur précis peuvent

¹Les ouvrages de [Cypher, 1993] et [Lieberman, 2000] constituent une bonne source de renseignements sur les interfaces adaptatives.

être dérivées de ces enregistrements sur base de son input et de son interaction antérieure avec le système, son historique. De manière générale, la mise en oeuvre des IUI permet de résoudre certains problèmes propres aux interfaces habituelles (dites de manipulation directe) en utilisant différentes méthodes de l'intelligence artificielle. Parmi les plus importants de ces problèmes, nous pouvons citer :

La création de systèmes personnalisés

Comme nous l'avons décrit plus haut, chaque utilisateur a ses propres habitudes, préférences et méthodes de travail. Une interface intelligente qui prend en compte les particularités de différents utilisateurs peut fournir des méthodes personnalisées d'interaction. L'interface "connaît" l'utilisateur, et peut dès lors utiliser cette connaissance pour mieux communiquer avec lui.

La gestion de l'*overflow* d'informations et de problèmes de filtrage

Les interfaces intelligentes peuvent faciliter considérablement la tâche de recherche des données dans de grandes bases d'informations ou dans des systèmes complexes (en utilisant les techniques de data mining par exemple). En filtrant tout ce qui n'a pas d'importance, on arrive à diminuer la charge cognitive sur l'utilisateur. . . Ceci permet de cibler les solutions qu'on lui propose et, dans certains cas, les IUI peuvent même proposer de nouvelles solutions inconnues jusque-là.

Aide à l'utilisation de programmes complexes

Certaines applications peuvent être assez difficiles à manipuler et souvent la plupart des utilisateurs ne connaissent tout simplement pas toutes les possibilités qui leur sont offertes. Pour contrer ce problème, on peut utiliser des systèmes d'aide intelligentes qui vont faire des remarques sur les actions courantes, expliquer les concepts qui se cachent derrière et fournir des informations qui vont faciliter les tâches futures.

Soulager l'utilisateur en effectuant le travail à sa place

Les IUI peuvent observer les actions de l'utilisateur, comprendre et reconnaître ses intentions et, donc, éventuellement, faire le travail à sa place, ce qui lui permettra de se concentrer sur des choses plus importantes. *Ce concept marche bien surtout dans le contexte des actions répétitives.*

La liste des problèmes cités n'est évidemment pas exhaustive, mais correspond bien aux principaux problèmes des futures interfaces de différents services BIGRE. Dans la suite, nous allons montrer à quel niveau ces problèmes se situent dans le système et quelles sont les différentes approches qui vont permettre de les résoudre.

2.3.2 Workflow Intelligent

2.3.2.1 Motivation

Le Workflow représente pour les différents utilisateurs bio-informaticiens un fabuleux outil de travail qui leur permet d'accomplir de complexes expérimentations biologiques *in silico*¹ et ce à un très haut niveau d'abstraction [Goble *et al.*, 2003].

¹Par opposition aux expériences de laboratoires.

Les expérimentations *in silico* sont des procédures informatiques, dont le but est de tester des hypothèses et de prouver des faits biologiques connus sur base des informations contenues dans de larges bases de données en utilisant pour cela la puissance de calcul qu'offre la fédération BIGRE. Nous pouvons donc dire que le Workflow est l'équivalent virtuel d'un laboratoire biologique, et qu'il peut de ce fait être considéré comme étant l'élément principal du processus de réalisation des expérimentations par les utilisateurs.

En mettant à la disposition du bio-informaticien (tous profils confondus) un outil aussi puissant qu'un laboratoire virtuel, nous allons inévitablement nous heurter à plusieurs problèmes d'interaction Homme-Machine propres aux interfaces de manipulation directe. C'est ainsi très naturellement que nous retrouvons dans le cas du Workflow la plupart des problèmes ayant fait l'objet de discussion dans la section précédente. Par exemple, le caractère par définition distribué des bases d'informations utilisées au sein de la fédération BIGRE ainsi que leurs tailles considérables rendent très logiquement difficile la tâche de la création d'une interface de Workflow intelligible. Cette constatation implique obligatoirement la mise en oeuvre d'un système d'aide à la navigation dans l'interface du workflow. Ce système doit permettre aux utilisateurs de retrouver facilement les différentes informations qui les intéressent (par exemple les services actuellement disponibles à l'utilisation au sein de la fédération) et ce, de la manière la plus transparente possible.

Normalement, l'invocation des différents services de la fédération résulte en un filtrage de l'ensemble des options (services) disponibles pour la suite de la manipulation (sur base des données obtenues en sortie du service invoqué), car les appels aux services sont réalisés à partir de leurs descriptions (issues des ontologies des services), des descriptions de leurs données (issues des ontologies des données) et des informations sur les instances de services (issues de l'annuaire). Mais même si ce filtrage facilite considérablement la recherche des options disponibles pour les utilisateurs dans une situation précise, il s'agit dans ce cas plus d'un mécanisme indispensable à la cohérence du système que d'une réelle amélioration *intelligente* de l'interface (au sens "intelligence artificielle"). C'est la raison pour laquelle des approches supplémentaires d'aide aux utilisateurs doivent être envisagées. Les nouvelles approches devront accélérer encore plus le processus du choix des composantes du Workflow et de son élaboration tout en facilitant le plus possible le travail aux utilisateurs. Les deux aspects, à savoir l'efficacité temporelle et l'intelligibilité de la représentation de l'aide aux utilisateurs, sont également importants. Notons finalement qu'à part le gain temporel, un autre intérêt particulier du Workflow intelligent est l'assistance à l'utilisateur. Le système doit être capable de remplacer l'utilisateur dès que possible en lui proposant de nouvelles solutions dont il ignorait pour une raison ou une autre l'existence et en les exécutant à sa place si tel est son désir.

2.3.2.2 Analyse de l'IHM

La particularité de l'analyse que nous entamons ici consiste dans le fait que le projet BIGRE n'est pour l'instant qu'à ses débuts et qu'aucun composant du système n'a pour le moment encore été implémenté. Pire encore : certaines idées

ne sont qu'à moitié réfléchies et on ne connaît souvent que leur objectif final. C'est précisément le cas du Workflow. Nous connaissons ce que cet outil devra faire en définitif, mais nous n'avons pour l'instant qu'une très vague idée de ce à quoi ressemblera son aspect extérieur. Heureusement dans le cas du Workflow, il y a moyen de procéder à une analyse théorique de l'IHM sur base d'objectifs généraux sans pour autant être obligé d'implémenter le service. Nous allons baser notre analyse sur une simple maquette de workflow présentée sur la figure 2.5¹.

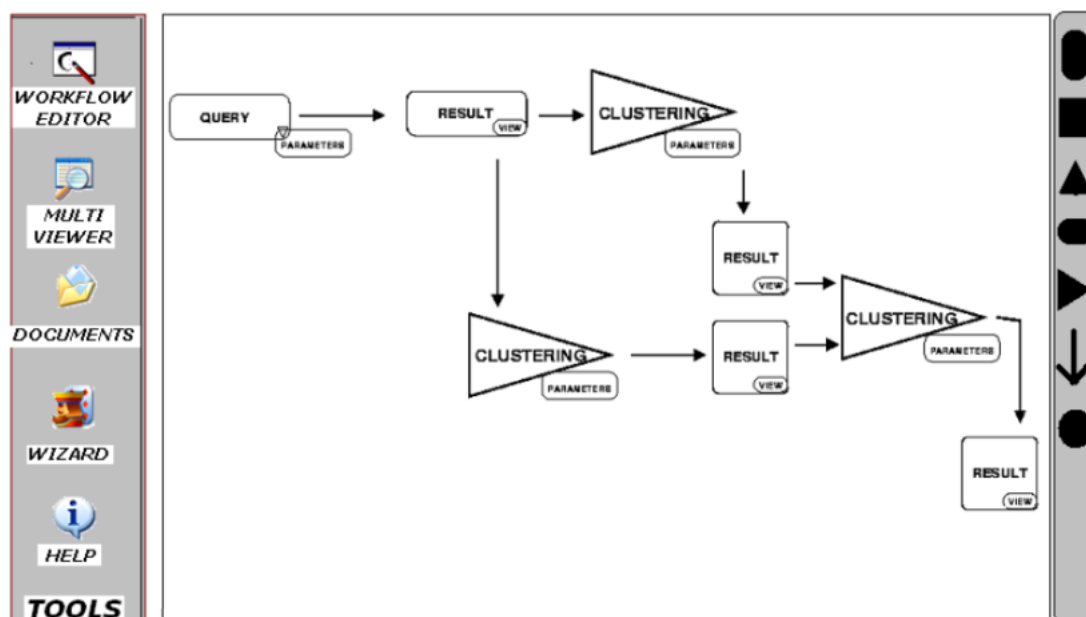


FIG. 2.5 – IHM schématique du Workflow

Le schéma montre clairement les principales parties de la future IHM :

- **La barre des outils.** Son rôle est de configurer les différentes propriétés liées au workflow, telle que la présentation de son code XML, le lancement de l'exécution ou encore l'obtention de l'aide en ce qui concerne son utilisation.
- **L'aire de travail** constitue le workflow à proprement parler. C'est là qu'on pose les différentes icônes-services en les chaînant sous forme de diagramme pour constituer de tâches complexes.
- **Les icônes** représentent l'ensemble des services disponibles à l'utilisation au sein de la fédération. Elles peuvent être triées et regroupées de manière quelconque. Ce sont ces icônes qu'on place sur l'aire de travail.

L'IHM proposée nous semble très bien convenir aux besoins du Workflow. En effet, cette approche permet de trouver des solutions intéressantes pour tous les problèmes dont on a parlé dans la motivation. Le regroupement des icônes dans une seule barre selon certains critères abstraits (des critères fonctionnels par exemple) résout de manière très élégante les problèmes de la transparence des services dispo-

¹L'exemple ne correspond pas à la réalité biologique.

nibles au sein du système et celui de la navigation parmi ces services. Le bioinformaticien désireux d'utiliser un certain service disponible dans sa liste des icônes ne doit rien savoir sur l'emplacement réel du service en question (pas d'adresse réseau à remplir, pas de codes à retenir). L'aspect distribué lui est complètement caché. Un autre problème qu'on pourrait résoudre facilement avec une telle configuration du Workflow est le problème de filtrage des options disponibles à l'utilisateur à la suite d'une certaine *action*¹ précise (action suivante). Lorsqu'on décide d'effectuer une certaine action et de mettre sur l'aire de travail le service correspondant, il est alors très facile de mettre à jour toutes les listes des icônes concernées de manière à rester cohérent avec la suite que l'utilisateur peut entreprendre à partir de cet état-là du diagramme. Néanmoins, une telle approche peut paraître très restrictive aux yeux de certains, puisque dans ce cas le choix des actions est limité (par le service précédent). Or ce n'est pas parce qu'on a sélectionné une certaine icône qu'on est forcément sur le point d'utiliser sa sortie (de continuer cette branche-là de l'arborescence). On pourrait ainsi être induit en erreur (ne pas trouver des services utiles) par une simple maladresse (un mauvais clic de souris par exemple). La solution à ce problème est de combiner deux approches ; mettre à jour les listes des services disponibles lorsqu'on sélectionne une certaine icône dans l'air de travail, mais sans pour autant cacher complètement les autres services. L'idée est de souligner de manière claire à l'utilisateur qu'il y a deux sortes de services à sa disposition : ceux qui peuvent continuer l'arborescence et les autres. Une bonne manière de réaliser cette opération est de marquer d'une manière ou d'une autre les services dans la liste qui les contient. Notons également qu'en plus de ce mécanisme de filtrage de base, on peut faciliter encore plus la navigation parmi les services disponibles à un moment donné, en construisant un modèle de l'utilisateur et en adaptant le contenu des listes à cet utilisateur précis. L'exemple le plus simple d'utilisation de cette méthode consiste à afficher en premier lieu les n services les plus utilisés.

La construction d'un modèle d'utilisateur est une notion tout à fait primordiale. Elle est à la base de l'utilisation des méthodes d'intelligence artificielle dans le Workflow. C'est en utilisant ces modèles d'utilisateurs qu'on pourra créer un Workflow réellement personnalisé, capable d'assister efficacement les utilisateurs et de prendre leur relève si besoin est. Une assistance efficace et intelligente sous-entend l'élaboration d'un système capable non seulement de faciliter la tâche à ses utilisateurs, même si bien évidemment l'intelligibilité de l'IHM et les améliorations de base ont beaucoup d'importance, mais aussi (et surtout) capable de proposer à l'utilisateur des solutions intelligentes qu'il ne connaissait pas auparavant et qu'il ne pouvait généralement pas connaître. Il s'agit des solutions que l'utilisateur ne pouvait pas connaître lorsque l'analyse se basait sur les modèles des autres utilisateurs (des utilisateurs experts). Lorsque l'analyse est basée sur son propre modèle d'utilisation, l'intelligence consiste à déceler des patterns (motifs) répétitifs dans son comportement et de proposer à l'utilisateur de les rejouer au moment opportun. Ces deux cas de figures d'utilisation de l'intelligence artificielle peuvent être appliqués au Workflow. Néanmoins la question de savoir si l'environnement du workflow est répétitif, et si par conséquent déceler des actions répétitives chez un utilisateur précis a un sens, reste ouverte². C'est la raison pour laquelle, dans un

¹Il s'agit d'une action dans le sens défini dans la section 1.8.1

²Ce problème nécessite une analyse poussée dans de conditions d'utilisation réelles.

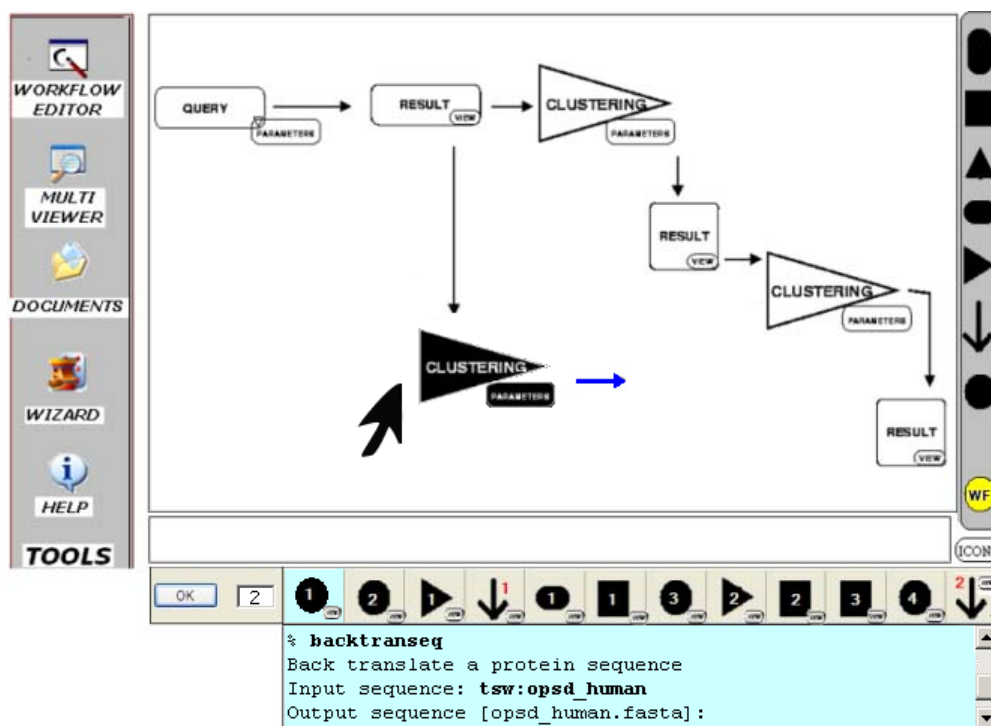


FIG. 2.6 – IHM schématique amélioré du Workflow.

premier temps nous allons nous concentrer sur l'analyse des modèles des autres utilisateurs, l'idée étant de retrouver dans leurs modèles des motifs en relation avec le travail actuel de l'utilisateur. Il sera ainsi possible de *prédire* les futures actions de l'utilisateur en question.

La figure 2.6 montre les améliorations à apporter à l'IHM pour mettre en oeuvre le mécanisme de prévisions. La modification principale à apporter au schéma précédent consiste en une barre de prévision qui va fournir à l'utilisateur le choix des prochaines actions à effectuer selon ce qu'il vient de faire à l'instant. Ainsi nous pourrons choisir le nombre n d'actions à prédire (plus n est petit, plus la prédiction est probable), configurer les actions directement dans la barre et exécuter cette suite d'actions ainsi constituée ou choisir directement une autre suite (celle qui nous convient le mieux) dans la liste des suites triées par probabilité et l'exécuter. Deux autres améliorations de la nouvelle IHM sont une barre d'aide, dont le but est de tenir informé les utilisateurs des conséquences des actions qu'ils entreprennent (simple description des services sélectionnés par exemple) et un nouveau bouton particulier WF dans la barre des icônes qui permet d'invoquer directement des workflows sans être obligé de reconstituer le diagramme correspondant à la main.

2.3.2.3 Contraintes de la mise en oeuvre

Le fonctionnement du Workflow (ou plus précisément celui de sa barre de prévision), introduit dans la section précédente, implique une série de contraintes d'ordre général sur la réalisation de l'interface présentée sur la figure 2.6 et, par conséquent, sur le choix de différentes méthodes techniques qui conviennent à la réalisation de cette tâche. En effet, l'efficacité du Workflow dépendra en grande partie de paramètres tels que l'indépendance, la souplesse et la transparence.

La contrainte de l'indépendance fait référence au développement du programme principal (le Workflow) qui doit être séparé de celui de *la couche IA* (la barre de prévisions). L'interaction entre les deux va se faire uniquement via *des points d'accès* (techniquement représentés par des zones de stockage communes par exemple) bien précis. Puisque les deux tâches sont effectuées simultanément (la construction du Workflow est effectuée en parallèle avec l'analyse et la gestion du modèle d'utilisateur courant), nous voyons apparaître le concept de *multithreading*. Ce concept est d'autant plus actuel que la couche IA elle-même consistera en plusieurs¹ processus appelés *agents* et exécutés en même temps en *background*. Parmi ces processus nous trouverons entre autre : l'agent responsable de l'affi-

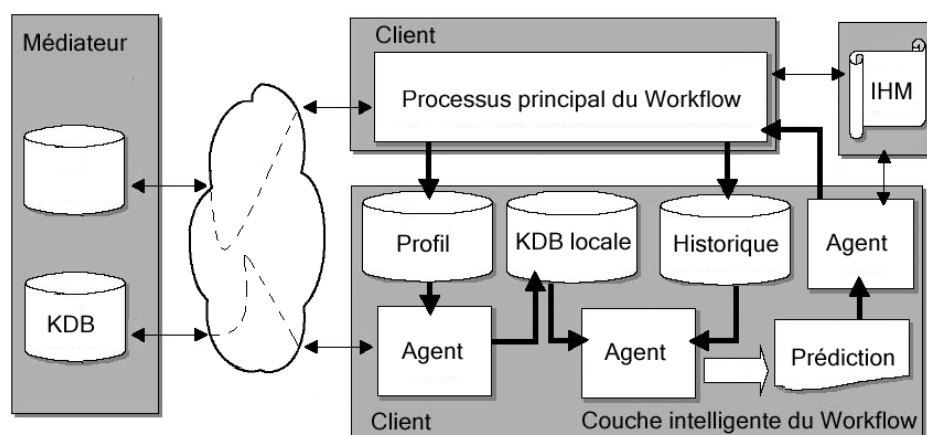


FIG. 2.7 – Interaction entre les agents et le processus principal

chage des prévisions (son rôle est donc de peupler la barre des prévisions), celui responsable du traitement de l'historique des actions de l'utilisateur (dans le but de calculer les prochaines prévisions) et celui responsable de la collecte des connaissances indispensables au calcul des prévisions. La figure 2.7 montre l'interaction entre ces trois agents, les différentes sources de connaissances et le processus principal :

- Le premier agent gère l'affichage des résultats de l'analyse et est également responsable de l'exécution des actions prédites à l'étape précédente (affichées dans la barre des prévisions) au cas où l'utilisateur le demande. Les actions choisies sont dans ce cas passées au processus principal du Workflow qui les exécute alors comme s'il s'agissait des actions normales.

¹Dans la suite nous analyserons le cas de trois processus.

- Le second agent traite l'historique de l'utilisateur (l'ensemble de ses actions). Il reçoit une nouvelle action de l'utilisateur de la part du processus principal chaque fois que celle-ci est effectuée. Il procède ensuite à l'analyse de la situation courante sur base des données ainsi recueillies et des modèles des utilisateurs experts gérés par le troisième agent.
- Et, finalement, le troisième agent, lui, s'adresse via le réseau au répertoire contenant les descriptions des workflows de référence pour construire les modèles des utilisateurs experts (base de connaissances). La réponse de la fédération est bien entendu fonction du profil actuel de l'utilisateur.

La contrainte sur la souplesse concerne l'entretien technique de la future application. La couche IA doit pouvoir facilement être mise à jour. Ceci est relativement facile à mettre en oeuvre si nous tenons compte du fait que le code source du Workflow et donc celui de la barre des prévisions se trouvent dans la définition XML du service workflow dans le *Service Directory*. Il suffit donc de modifier *le code mobile* dans la description du service pour modifier le comportement de la barre des prévisions - que ce soit la modification des algorithmes de prévisions, ou la modification des structures de données utilisées par les agents de la couche IA, ou encore la modification des protocoles de communication (propres à la couche IA) ou tout autre modification du fonctionnement de la couche IA. La dernière contrainte importante à respecter est la transparence de l'application. Ce point concerne plutôt les utilisateurs, mais n'en est pas moins important. Le fonctionnement de la couche IA doit faire le moins possible appel à l'utilisateur et ne doit surtout pas le gêner dans son travail (pas de demandes de confirmations inutiles, pas d'arrêt de travail pour cause, par exemple, de mise à jour du profil, etc.). Cette contrainte est bien respectée de par la nature-même de la barre des prévisions ; la barre est peuplée (si possible) sans une intervention directe quelconque de l'utilisateur. Les actions ne sont effectuées que lorsque l'utilisateur exprime ce désir de manière explicite.

2.4 Réalisation du WI

Le principal mérite de l'analyse des contraintes de la mise en oeuvre du Workflow est la découpe en sous étapes du processus de développement qui en découle. En effet, cette découpe facilite considérablement la conception des différentes composantes du système. La seule condition à remplir avant de procéder à la conception des agents (l'implémentation technique et les tests plus tard) est de définir clairement les notions encore floues pour l'instant, telles que le contenu des différentes sources de connaissances auxquelles les agents font appel, et d'analyser de manière plus algorithmique leur fonctionnement (il faut situer l'emplacement des différents algorithmes intelligents au sein de la couche IA tout en précisant leur rôle global exact). L'analyse de la réalisation fait l'objet de cette section.

2.4.1 Profil de l'utilisateur

Nous commençons l'analyse de la problématique concernant la réalisation du workflow intelligent par l'introduction d'une claire définition du profil de l'utilisateur. En effet le profil d'un certain utilisateur, définit lorsque celui-ci travaille sur un certain projet bio-informatique bien précis, est l'élément central de toute notre stratégie de prédictions intelligentes que nous nous apprêtons à mettre en oeuvre. Le profil de l'utilisateur constitue le point de départ de la chaîne des manipulations censées nous permettre cerner la logique de *ses actions*¹. C'est le profil de l'utilisateur qui détermine de manière univoque ce dont le système sera capable une fois mis en marche. De ce fait, nous pouvons affirmer que le profil de l'utilisateur a une importance tout à fait capitale au sein du Workflow (sa partie intelligente) puisque l'utilité de tout le système dépendra presque exclusivement de la cohérence du profil, la suite n'étant qu'une simple question de technique purement mécanique.

Intuitivement ces affirmations semblent très logiques ; le fait de savoir si on pourra deviner les prochaines idées de l'utilisateur dépend directement de notre degré de connaissance du domaine de travail de cet utilisateur. Notre pari est de se dire que généralement les utilisateurs qui travaillent dans un certain domaine sur un certain sujet auront tendance à avoir recours aux mêmes méthodes et techniques de travail. Ainsi, si on arrive à déterminer le domaine de travail de l'utilisateur (décrit par son profil), on peut alors espérer aligner son comportement sur celui de ses collègues et donc déduire des conclusions intéressantes sur base de son comportement actuel. Nos conclusions seront dans ce cas d'autant plus correctes que notre connaissance du domaine de travail de l'utilisateur est meilleure. Notons que nous n'essayons pas de modéliser l'utilisateur. Notre approche ne consiste pas à essayer d'examiner les habitudes de l'utilisateur pour ensuite déceler des répétitivités dans son comportement. Cette approche proposée par [Ruvini, 2000] et analysée dans le premier chapitre n'est pas très appropriée à notre situation², car elle suppose que nous avons affaire à un environnement *hautement* répétitif. Cette hypothèse (assez forte) n'est pas complètement insensée, mais nécessite des preuves solides obtenues suite à des analyses poussées sur un grand nombre d'exemples réels. Malheureusement il n'est pas encore possible d'obtenir ces preuves, et combien même ça aurait été possible, l'idée de retrouver beaucoup de répétitions chez un utilisateur du workflow précis semble assez peu probable vu la nature des manipulations qu'il sera amené à effectuer ; en effet dans le cas du Workflow les actions de l'utilisateur sont plus corrélées à son sujet de travail, plutôt qu'à sa manière de travailler.

Lorsque nous parlons du profil de l'utilisateur, il faut faire attention à la signification de ces deux mots, car les apparences dans notre cas peuvent s'avérer trompeuses. Si le mot *profil* ne pose pas de questions quant à son interprétation, le fait de préciser qu'il s'agit d'un profil associé à un utilisateur n'est pas tout à fait correct. Comme nous venons de le préciser à l'instant, nous ne modélisons pas les utilisateurs et donc les utilisateurs n'ont pas de profil par définition. Ainsi le mot *profil* doit être vu dans le cas du Workflow comme étant associé à un *projet* précis d'un utilisateur, plutôt qu'à cet utilisateur. Ceci veut dire que d'un travail

¹La notion définitive d'*action* en termes techniques sera définie lors de l'implémentation. Pour l'instant notons que chaque manipulation dont le but est de gérer (la création, la modification ou la destruction) un élément du workflow quelconque peut être vue comme étant une action.

²Nous utiliserons plus tard le principe de l'approche répétitive dans un autre but.

à l'autre le profil de l'utilisateur peut changer, ce qui d'ailleurs est très logique compte tenu des objectifs de la mise en oeuvre de ce système des profils.

Au départ les profils sont créés par les utilisateurs souhaitant recourir à l'aide de la barre des prévisions. Les utilisateurs sont alors amenés à activer l'option qui leur permet d'avoir accès à l'intelligence de manière explicite. La justification de ce passage un peu forcé s'explique par le fait qu'il est indispensable d'avoir un profil cohérent lorsqu'on souhaite obtenir des résultats ne serait-ce que plus ou moins convaincant de la part de la barre des prévisions. Lorsque la définition du sujet de travail est trop générale, les chances de se faire aider décevantement par l'application sont bien évidemment assez maigres. Ainsi, nous avons mis au point trois manières principales de définir le profil qui correspond au travail en cours :

- *Spécification manuelle du sujet de travail.* Il s'agit ici du cas de figure probablement le plus basique. L'utilisateur introduit un texte *libre* (ASCII) qui sera par la suite analysé par le système de manière à en extraire l'image la plus fidèle possible du travail en cours de réalisation. Cette manière d'agir est assez simple et ne demande de l'utilisateur qu'un minimum d'efforts - un abstract d'un article quelconque en rapport avec le sujet en question suffit.
- *Spécification manuelle des mots-clés.* La difficulté principale de la première méthode est qu'il est assez compliqué de pondérer correctement les différents mots présentés dans un texte libre, car un grand nombre de ceux-ci ne représentent généralement que du bruit. Les algorithmes de comparaison de textes produisent alors dans ce cas souvent des solutions biaisées. Une solution possible pour se débarrasser de cet inconvénient est de donner aux utilisateurs un moyen d'affiner leurs définitions. Pour ce faire, un choix de différents mots-clés leur est proposé. Il ne leur reste alors plus qu'à préciser les mots-clés qui correspondent à leur profil parmi ceux proposés.
- *Spécification de titres des articles.* L'idée ici est de préciser les titres des articles ayant un rapport avec le sujet de travail (éventuellement avec une pondération arbitraire des différents articles). Le système s'occupera alors tout seul d'aller retrouver l'abstract et les mots-clés qui correspondent à cet article sur le réseau. L'avantage de cette méthode par rapport à la première est le fait qu'on a des mots-clés en plus du texte et qu'on ne risque pas de se tromper en introduisant la description. L'avantage par rapport à la deuxième méthode est la transparence, la facilité d'utilisation et le gain de temps.

L'implémentation de la saisie initiale du profil par les trois méthodes définies ci-dessus reste bien évidemment à réaliser ainsi que d'ailleurs le choix des algorithmes convenant à cette tâche. L'analyse de ce problème sort du cadre de notre travail sur la prédiction des workflows et constitue un sujet de mémoire en soi. Notons simplement qu'un nombre de plus en plus considérable de travaux traitent des problèmes issus du domaine de *literature mining*¹ ces dernières années (surtout appliqué au domaine de la biologie). Ainsi, la solution à notre problème pourra être trouvée ultérieurement dans des travaux tels que [de Bruijn and Martin, 2002].

¹Il s'agit d'un processus d'extraction et de recombinaison de faits à partir des publications scientifiques. Plusieurs résultats intéressants en ce qui concerne des outils informatiques capables d'extraire des données à partir des abstracts (ou textes complets) contenus dans des bases de données, telles Medline ou PubMed ont été proposés dans le domaine de la biologie moléculaire.

Une amélioration possible de la description statique du profil de l'utilisateur serait une approche plus dynamique. Au lieu de forcer l'utilisateur à préciser le sujet de son travail de manière explicite, nous envisageons la possibilité de mettre en place un mécanisme de surveillance qui nous permettrait d'essayer d'élaborer son profil automatiquement. En gros cette méthode devrait essayer d'approximer le deuxième type de saisie explicite, à savoir essayer de retrouver les mots clés définissant le travail en cours. Ces mots-clés pourraient par exemple être déduits en fonction des services invoqués. L'idée est d'associer à chaque service un ensemble de mots-clés typiques. Ainsi, après un certain nombre d'actions de l'utilisateur nous pourrions former un profil plus ou moins réaliste de son travail en superposant les descriptions de ces actions. La justification de la raison d'être de ce mécanisme est évidente : il permet d'éliminer la rigidité du système statique, car offre beaucoup plus de souplesse. Dans le cas statique, la description est supposée être connue depuis le départ, ce qui peut s'avérer être faux dans le cas des utilisateurs non expérimentés ou novices par exemple, et quand bien même ce ne serait pas le cas, la nécessité de passer par des étapes obligatoires (perte de temps) sans aucune certitude quant aux résultats pourrait diminuer considérablement l'attrait de l'intelligence pour les futurs utilisateurs potentiels. L'adaptation dynamique, elle, serait complètement transparente à l'utilisateur. De l'autre côté la gestion dynamique des profils pose un certain nombre de problèmes qu'on n'est pas sûr de pouvoir résoudre de manière positive, du moins pour l'instant, sans avoir recours aux tests à grande échelle dans des conditions d'exploitation réelle du système. L'hypothèse qui consiste à se dire qu'on sera en mesure d'extrapoler un profil fidèle à l'utilisateur après seulement un certain nombre restreint d'actions de sa part est purement spéculative pour l'instant, même si théoriquement rien ne dit le contraire. En effet, il est assez difficile de construire une bonne description "universelle" de mots clés pour chaque service disponible. Ainsi, il est clair que la question concernant la manière de définir le profil de l'utilisateur est "une question à double tranchant", puisque *de toute façon* en gagnant en transparence nous perdons en efficacité, et vice versa. La proposition finale est donc de combiner les deux approches tout en essayant d'optimiser l'approche dynamique le plus possible.

2.4.2 Knowledge Data Base

Notre idée de départ est de déduire les prochaines actions de l'utilisateur sur base de son profil. Plus précisément, pour ce faire, nous comptons nous baser sur le contenu des workflows effectués antérieurement au sein du système. En effet, puisqu'un profil ne contient que la description textuelle du projet auquel il est associé, l'utiliser en tant que tel pour la prédiction n'aurait tout simplement aucun sens. Ainsi, nous voyons apparaître une nouvelle notion dans notre réflexion, celle de la définition du workflow. Il s'agit là d'une notion complètement différente de celle du profil, bien qu'étroitement liée à celle-ci. En fait, les deux notions sont quasiment indissociables et vont toujours de paire. La définition du workflow correspond au travail effectué, alors que le profil de l'utilisateur décrit ce travail. Dans la suite nous appellerons *Workflow Expert* ou *Expert Workflow* (EW) l'entité logique constituée du double lien $\{\text{profil de l'utilisateur} \Leftrightarrow \text{définition du workflow}\}$.

Lorsque l'utilisateur aura terminé de travailler sur son workflow, il aura l'occa-

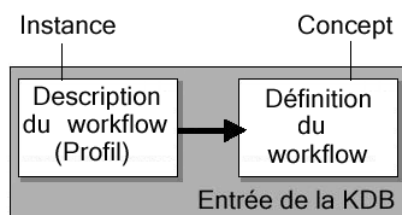


FIG. 2.8 – Expert Workflow

sion de rendre son travail public en l’envoyant sur l’un des médiateurs du système. Néanmoins avant qu’un workflow ne devienne réellement accessible au grand public en tant que modèle de conception (EW), il est indispensable de le faire valider par des experts, en lui attribuant éventuellement un certain *score*. L’ensemble des workflows validés sera stocké dans une base de données située sur l’un des médiateurs. Cette base de données constituée d’un ensemble de workflows experts est appelée *Knowledge Data Base*, ou encore *KDB*. La base de connaissances s’ajoute à la liste des éléments composant les médiateurs¹ et a une fonction bien plus élaborée qu’un simple stockage de données. En effet, la KDB devra offrir au même titre que par exemple le *Services Directory* deux interfaces à tous ceux qui en auraient besoin : celle permettant la consultation des workflows experts et des informations attachées à ces workflows et celle permettant la gestion des informations qu’elle contient (l’ajout, la suppression et la modification des données). L’une de ces interfaces sera réservée aux gestionnaires du médiateur et l’autre aux utilisateurs désireux de se constituer leur propre base de connaissances locale.

La procédure de construction de la base locale de connaissances (*local KDB*) est illustrée sur la figure 2.9. L’agent responsable de la construction de la base sur la machine cliente s’adresse à l’interface utilisateur de la KDB sur le médiateur en lui envoyant le contenu du profil de l’utilisateur courant. Dès que le profil est reçu de l’autre côté du réseau, l’interface utilisateur commence la recherche des workflows experts de la KDB ayant un certain degré de similitude σ (paramètre à déterminer de manière expérimentale) avec la définition du profil cible. Les entrées ainsi trouvées sont alors classées dans une liste en fonction de leur σ combiné à leur *score* et sont ensuite renvoyées au client en tant que sa nouvelle base de connaissances locale. Le nombre d’éléments η de la liste à renvoyer dépend bien évidemment du contenu de la liste. La seule limite à déterminer en ce qui concerne η est sa limite supérieure (la taille maximale de la *IKDB*). Il s’agit là d’un paramètre à trouver expérimentalement. Quant au problème du tri de la liste des entrées susceptibles d’être renvoyés au client, encore une fois, la formule exacte $f(\sigma, score)$ qui permettrait de maximiser l’efficacité de la classification reste à déterminer, même si pour l’instant nous pouvons prendre comme hypothèse une classification qui se déroule en deux étapes : d’abord en fonction de σ et ensuite en fonction du *score* pour les entrées ayant le même σ . Dans ce cas nous commençons par prendre les entrées les plus en rapport avec le sujet du workflow en cours de réalisation sur la machine cliente et c’est seulement après que va intervenir la qualité du workflow expert. Autrement dit, un workflow expert de moyenne qualité bien en

¹Voir 2.2.1.1

rapport avec le travail qui est effectué a plus de valeur qu'un workflow expert de qualité parfaite mais qui n'est lié que de très loin au travail courant de l'utilisateur.

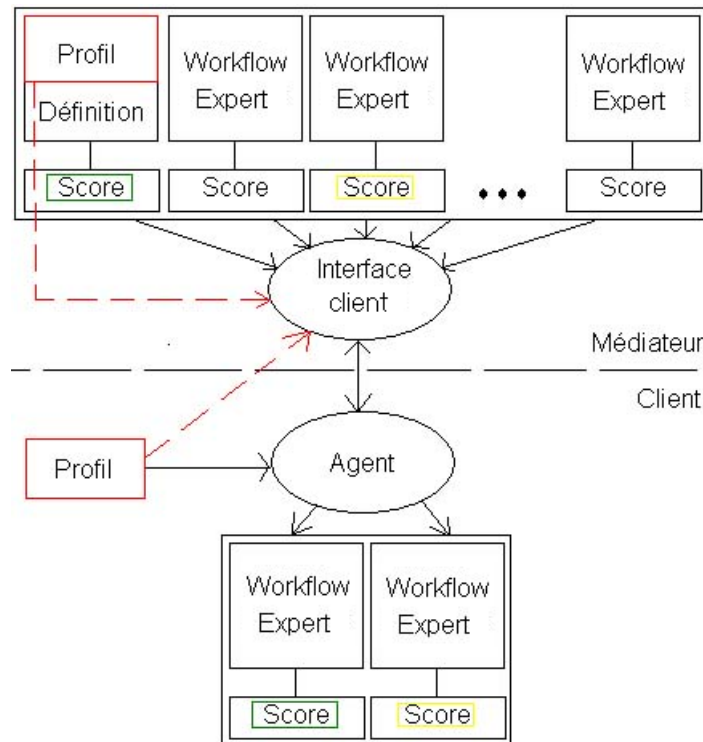


FIG. 2.9 – Construction de la IKDB

2.4.3 Historique des actions

L'utilisation de la base de connaissances locale nécessite l'observation des actions courantes de l'utilisateur puisque c'est, en effet, la combinaison de ces deux sources d'informations qui va nous permettre de prédire son comportement futur. En considérant la IKDB comme étant acquise, il nous reste alors à analyser le cas de *l'historique des actions* de l'utilisateur. Une manière de définir cette notion de l'historique est de dire qu'il s'agit d'un ensemble contenant toutes les actions que l'utilisateur a effectué tout au long de son travail sur son workflow. Néanmoins une telle définition reste très floue, car ne nous donne aucune idée précise quant à la construction (et donc quant au contenu exact) de cet ensemble. L'approche la plus simple et la plus naturelle de construire l'historique serait d'y rajouter comme telle chaque action effectuée par l'utilisateur. Ainsi, nous pourrions à chaque instant consulter les pas précédents de l'utilisateur et donc, par conséquent, nous serons normalement en mesure de faire nos prédictions. Seulement le problème de cette approche consiste dans le fait qu'une telle liste d'actions ne correspondrait pas au contenu de la IKDB. En effet, celle-ci nous renseigne sur l'agencement *final*

des composants des workflows et donc des actions correspondantes effectuées par les experts ; un agencement qui n'a généralement *rien* à voir avec la suite réelle des actions qui ont été effectuées lors de la création de ces workflows et qui n'a donc pas grand chose à voir avec un historique purement *séquentiel*, puisque l'historique séquentiel contient *toutes* les actions à effectuer pour obtenir un résultat final (avec toutes les erreurs et corrections possibles), et non seulement les actions qu'il faut effectuer pour obtenir un workflow opérationnel. Notons que la raison qui nous pousse à ne mettre dans les workflows des experts que les définitions finales de leurs travaux est évidente : nous ne sommes pas intéressés par leur manière de travailler (leurs habitudes) - nous sommes intéressés seulement par leurs résultats.

L'approche que nous allons retenir pour la construction de l'historique est donc celle qui ne prend en compte que la définition finale (à chaque instant) du workflow actuel. Ainsi, les actions sans impact sur le workflow actuel (par exemple la création d'un service suivie par son effacement) ne seront pas mémorisées et le format du contenu de l'historique sera identique à celui des définitions des workflows de la KDB des médiateurs. La construction de l'historique est prise en charge par

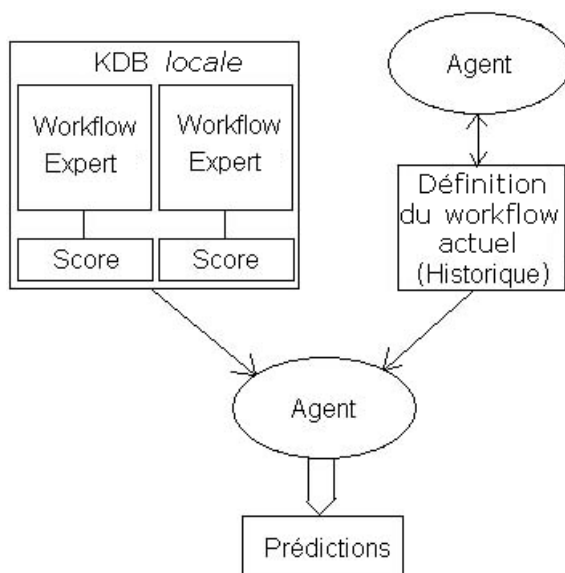


FIG. 2.10 – Utilisation de l'historique des actions de l'utilisateur

un des agents (celui responsable de l'interaction avec l'IHM). Comme illustré sur la figure 2.10, l'agent mettra à jour le contenu de l'historique chaque fois qu'une certaine action est effectuée. En même temps, un processus parallèle (un autre agent) analysera le contenu de l'historique ainsi créé et proposera sur base de la KDB les différentes prévisions possibles quant aux actions futures de l'utilisateur.

2.4.4 Choix algorithmiques

La vue globale sur le fonctionnement de la barre des prévisions que nous avons dressée dans ce chapitre permet de situer très précisément l'emplacement des différents algorithmes au sein du système tout en indiquant clairement leurs rôles respectifs. Ainsi, compte tenu de notre intérêt particulier en ce qui concerne les algorithmes de l'intelligence artificielle, nous comptons analyser dans la suite les différents cas de figure dans lesquels ces algorithmes apparaissent afin d'essayer d'en tirer des conclusions pratiques quant aux choix algorithmiques à effectuer.

Les algorithmes "intelligents" interviennent à deux niveaux dans notre schéma de fonctionnement : lors de la construction de la base des connaissances locale et lors de la prédiction du comportement de l'utilisateur (voir les figures 2.9 et 2.10). Dans le cas de la *IKDB* la problématique est celle de la recherche parmi un ensemble d'expériences passées de nouvelles solutions à des problèmes donnés sur base de similitudes entre les descriptions des problèmes actuels et celles des problèmes antérieures. Nous considérons dans ce cas que *le problème* c'est le profil cible, *les solutions* possibles sont les définitions des workflows ayant un profil semblable à celui de la "cible" et *les expériences passées* - l'ensemble des workflows des experts. Nous voyons donc qu'il s'agit ici de la définition même du **raisonnement à base des cas**. La *KDB* sur le médiateur forme l'ensemble d'apprentissage, ou encore la base des cas, où chaque workflow expert est un cas constitué de deux parties : *l'instance* qui décrit la nature du problème et *le concept* qui représente la solution à ce problème (voir la figure 2.8). L'approche CBR est d'autant plus appropriée dans le cas de la *IKDB* qu'elle est assez facile à mettre en oeuvre en pratique (par rapport aux autres approches d'apprentissage) et que nous savons qu'elle est particulièrement performante dans les domaines où la similarité entre les descriptions de problèmes nous donne une indication sur l'utilité des solutions antérieures, ce qui justement est notre hypothèse principale depuis le départ.

2.4.4.1 Prochaine Action

En ce qui concerne les prédictions des actions de l'utilisateur, le choix de l'algorithme d'apprentissage n'est pas aussi évident que précédemment. Nous savons que généralement l'efficacité d'un algorithme particulier dépend plus de la formulation et de la représentation adéquates du problème que de l'algorithme lui-même. Ainsi, une manière logique de procéder est de d'abord déterminer l'ensemble des algorithmes potentiellement utilisables compte tenu de la particularité de notre ensemble d'apprentissage (la *IKDB*), et ensuite d'implémenter ceux d'entre eux qui ont des performances théoriques intéressantes en commençant par le plus facile à mettre en oeuvre. Pour l'instant le format exact de l'ensemble d'apprentissage n'est pas encore défini, mais il pourra être mis sous n'importe quelle forme ultérieurement (traduit à partir de la définition du workflow initial). Par contre nous savons déjà que le contenu de cet ensemble peut être biaisé, puisque clairement nous pouvons nous retrouver dans une situation d'inconsistance des données, lorsque plusieurs instances identiques ont un concept différent¹ (plusieurs chemins

¹Il s'agit ici du premier type de bruit rencontré dans des ensembles d'apprentissages (voir 1.1). Le deuxième type ne pourra jamais se produire dans le cas du workflow, puisque l'ensemble d'apprentissage est complet (nous connaissons tous les noeuds de chaque workflow).

identiques dans le diagramme représentant le workflow ont des suites différentes). Ainsi nous devons éliminer de notre analyse tous les algorithmes incapables de gérer du bruit ainsi que ceux dont la complexité théorique est très élevée. Finalement nous avons retenu les algorithmes suivants : **C4.5** (avec/sans élagage) et le **Classifieur Naïf de Bayes**. Certains algorithmes, tel que **CALM** pourtant intéressant au niveau du temps de calcul et de la gestion du bruit, n'ont pas été retenus pour diverses raisons, dont, entre autre, la complexité de leur utilisation (typiquement le nombre de paramètres à régler). Il serait peut-être intéressant de les comparer à ceux retenus pour les tests lorsque les performances de ces derniers seront établies.

Notons que lorsque nous appliquons l'un des algorithmes choisis tel quel sur notre ensemble d'apprentissage, nous ne sommes en mesure de prédire qu'un seul concept à la fois. Autrement dit, nous pourrions prédire une seule prochaine action de l'utilisateur, puisque dans notre cas (implicitement) le concept est une action. Néanmoins, il est à remarquer que l'instance, elle, pourra avoir une taille supérieure à un, puisque nos algorithmes d'apprentissage travaillent sur base d'un vecteur d'attributs. Ainsi, nous pourrions prédire une seule prochaine action, mais en fonction d'un nombre arbitraire d'actions précédentes.

2.4.4.2 Prochaines Actions

Pour pouvoir prédire un ensemble de prochaines actions, nous proposons d'adapter la méthode d'apprentissage des habitudes des utilisateurs à notre situation. En effet, comme nous l'avons montré dans le premier chapitre cette méthode est capable de retrouver des répétitions de blocs d'actions dans un ensemble d'apprentissage et de construire sur base de ces blocs un nouvel ensemble d'apprentissage dont les instances restent les mêmes que précédemment, mais dont les concepts, eux, changent. Les concepts deviennent alors ces blocs répétitifs. Ainsi, avec un pré-traitement adéquat de la *IKDB*, nous pourrions la modifier de manière à prendre en compte les suites de manipulations typiques que font les utilisateurs lorsque le travail est effectué dans un certain domaine particulier de la bio-informatique.

Le pari dans ce cas reste le même que celui énoncé depuis le départ, à savoir que nous espérons retrouver des corrélations propres à un domaine bien précis de la bio-informatique entre une ou plusieurs actions des utilisateurs avec les prochaines actions qu'ils seront amenés à effectuer. Seulement dans ce cas, la suite sera plus "longue" et donc les chances que ce soit effectivement le cas, plus minces.

2.5 Cahier des charges

La réalisation du Workflow Intelligent décrite dans les sections précédentes et que nous nous apprêtons à effectuer dans la suite de notre travail nécessite une approche concise et méthodique du problème. Dans le tableau 2.1¹ nous proposons un cahier des charges reprenant les tâches techniques principales à effectuer pour

¹Les modules sont marqués par **M**, les priorités par **P** et les tâches facultatives par **F**

M	P	Description de la tâche	F
I		Construire <i>à la main</i> une base de connaissances expérimentale respectant la réalité biologique et contenant un nombre suffisamment élevé de workflows experts. Procéder ensuite aux tests et à l'analyse des performances en appliquant pour cela aux données expérimentales recueillies précédemment les différents algorithmes implémentés et les différentes méthodes de tests possibles.	
II	1 2 3 4	1 Elaboration du format de représentation des workflows sous forme d'ensemble d'apprentissage et du format des définitions des workflows experts stockées dans les bases de connaissances. 2 Elaboration du format de représentation de l'historique des actions. 3 <i>Conception d'un traducteur du format initial des workflows en format de représentation sous forme d'ensembles d'apprentissage.</i> 4 Implémentation des différents algorithmes d'apprentissage retenus pour la procédure de prédiction (en commençant par C4.5). Ces algorithmes doivent être capables de gérer nos formats de représentation des workflows sous forme d'ensembles d'apprentissage et de l'historique des actions en entrée.	*
III	1 2 3 4 5 6 7	1 Elaboration du format définitif du profil d'utilisateur. 2 <i>Conception d'une application recevant en entrée un abstract (article) ASCII ou une référence web vers cet abstract (article) et renvoyant en sortie le profil d'utilisateur correspondant.</i> 3 Définition de la structure et du format technique de la KDB. 4 Implémentation de la procédure CBR de comparaison de différents profils d'utilisateur accompagnée d'un jeu de tests pertinents. 5 Conception de l'interface utilisateur de la KDB (côté médiateur). Il s'agit d'une application basée sur la procédure CBR acceptant un profil d'utilisateur en entrée et qui renvoie une liste de définitions de workflows experts. Le contenu de chaque définition est transparent pour l'application et pour la KDB. 6 <i>Mise en réseau de l'interface utilisateur de la KDB. Jeu de tests avec un agent (basique) distant chargé de lancer des requêtes contenant un profil cible et recevant en réponse de la part de la KDB sur le médiateur la liste de définitions des workflows experts.</i> 7 <i>Conception de l'interface administrateur de la KDB.</i>	* * *
IV		<i>Assembler les différentes composantes du Workflow Intelligent et les intégrer ensuite dans l'application principale du Workflow.</i>	*

TAB. 2.1 – Le cahier des charges

mettre en oeuvre l'ensemble des idées (abstraites) que nous avons proposé tout au long de notre description de la barre des prévisions. Les tâches sont regroupées en quatre modules, dont les deux premiers peuvent être réalisés indépendamment l'un de l'autre. Dans chaque module les tâches sont classées selon leur priorité. Les tâches marquées par une étoile sont des tâches "facultatives" qui sortent du cadre de ce mémoire, car pour diverses raisons ne peuvent être effectuées pour l'instant.

Notons que notre cahier des charges ne reprend que les tâches les plus essentielles au fonctionnement du système. L'accent est placé principalement sur l'aspect algorithmique du problème. D'autres tâches, telle l'introduction d'une surveillance des actions de l'utilisateur dans le but de former son profil de manière dynamique, pourront être abordées ultérieurement lorsque l'environnement du Workflow principal sera disponible. C'est en effet la raison pour laquelle le module IV ne fait que décrire son objectif principal sans détailler les différentes tâches qui le composent.

2.6 Conclusion

Le projet BIGRE vise à concevoir, développer et déployer une architecture distribuée de médiateurs sur Internet dans le but de fédérer des ressources bio-informatiques. La description de l'architecture, composée de trois sous-systèmes principaux, à savoir *les clients*, *les médiateurs* et *les services* nous sert de point de départ dans notre analyse. Une attention tout à fait particulière est consacrée aux services bio-informatiques. C'est en effet grâce au système des *plug-ins* à la base du fonctionnement des services, qu'on voit apparaître la notion de Workflow.

Le Workflow est décrit comme étant un laboratoire biologique virtuel. Il s'agit d'un service spécial permettant aux utilisateurs d'effectuer à un très haut niveau d'abstraction des expériences biologiques *in silico*. Son avantage principal consiste en son IHM hautement intelligible constituée d'une aire de construction de diagrammes représentant des enchaînements graphiques de tâches biologiques - *les workflows*. L'étape suivante dans notre réflexion a été celle de rajouter au Workflow *une barre des prévisions* capable d'aider de manière *intelligente* les utilisateurs en leur suggérant de nouvelles idées dont ils ignorent *souvent* l'existence. La barre des prévisions se base sur l'analyse des workflows construits antérieurement par des experts et stockés dans *la base de connaissances* KDB située sur le Médiateur. Le mécanisme de la barre des prévisions constitue la couche intelligente du Workflow qui du côté Client est composée par trois processus indépendants appelés agents et d'un ensemble de sources de données parmi lesquelles nous retrouvons *le profil*, *la base des connaissances locale* lKDB et *l'historique*. De l'autre côté du réseau, sur le Médiateur, se trouve la KDB avec ses deux interfaces d'accès : l'interface utilisateur et l'interface administration. Comme illustré sur la figure 2.7 le processus de prédiction se déroule en cascade : l'interface utilisateur de la KDB renvoie l'ensemble d'apprentissage au Client suite à une requête concernant un profil précis, la base des connaissances locale est alors construite par un des agents du Client qui passe ensuite la main à l'agent chargé de produire les futures prédictions sur base de l'historique et de la lKDB fraîchement construite, qui lui à son tour passe la main à l'agent chargé d'afficher le résultat dans la barre des prévisions.

La dernière partie de notre analyse de la conception du Workflow Intelligent se penche sur l'aspect algorithmique de la mise en oeuvre du fonctionnement décrit plus haut. Compte tenu du rôle et du contenu des différentes sources de données de la couche intelligente, nous avons pris la décision d'implémenter l'algorithme **CBR** dans le processus de l'interface utilisateur de la KDB sur le Médiateur, et l'un des trois algorithmes **C4.5** (sans/avec élagage) et **NBC** pour la prédiction des intentions (la prochaine action) de l'utilisateur sur base de la *IKDB* et de son historique.

Le cahier des charges présenté dans le tableau 2.1 contient les différentes tâches à réaliser dans le chapitre suivant. Il s'agit principalement de la définition technique du format des sources de données de la couche intelligente et l'implémentation des algorithmes d'apprentissage retenus accompagnée par des jeux de tests pertinents.

Chapitre 3

Etude des performances

La réalisation pratique du Workflow Intelligent ne peut être entamée sans une “validation” concluante préalable de l’idée principale derrière le concept de prédiction des futures actions d’un utilisateur donné sur base des actions ultérieures des autres utilisateurs. Par “validation” nous entendons une mise en évidence indiscutable de l’utilité d’un tel système. La notion d’utilité étant elle-même assez floue, nous nous contentons de la mesurer en fonction des performances objectives observées ; si le système est capable de prédire des actions avec un taux de réussite convenable, alors il est utile. Il s’agit du premier point de notre cahier des charges.

L’étude des performances potentielles du système se divise en trois grandes étapes. La première a pour objectifs la collecte de données bio-informatiques (dont le contenu doit être représentatif des futurs workflows) et l’adaptation de leur format de représentation à celui accepté par les algorithmes d’apprentissage utilisés¹. Le choix de la méthodologie d’analyse des données recueillies constitue la deuxième étape de l’étude. L’objectif des différentes configurations d’analyse présentées à cette étape est de mettre en évidence l’influence de la plupart des paramètres susceptibles d’avoir un impact sur les performances de prédiction. La dernière étape de l’étude consiste très naturellement en l’analyse des résultats obtenus suite à l’application des méthodes de test sur les données et, plus important, en l’apport d’une réponse quant à la question de l’utilité globale du Workflow Intelligent.

Ce chapitre est composé de trois sections reprenant chacune l’une des trois grandes étapes de l’étude des possibles performances de notre système de prédiction.

¹Il s’agit des algorithmes Java fournis par le projet Weka.

3.1 Données

3.1.1 Description des données

Toutes les expérimentations présentées dans ce chapitre ont été conduites sur des traces collectées à l'issue de l'utilisation de l'application web **Emboss**¹. Les différents comptes utilisateurs représentés dans nos données (288 comptes au total) ont par définition des profils fortement diversifiés allant de l'utilisation purement académique propre aux étudiants novices jusqu'à une utilisation professionnelle dans les milieux de recherche et industriels. Dans le tableau 3.1 nous présentons un

```
biofor07 biofor08 biofor09 biofor10 biofor11 biofor12 biofor13
bioinfo3 bioinfo4 bioinfo5 bioinfo6 bioinfo7 bioinfo8  bjouris
dbaup dchristo dcoorna ddeforce ddoucet dea1 dea12 dea13 dea14
dea15 dea16 dea17 dea18 dea19 dea2 dea20 dea22 dea3 dea5 dea6
ipmb15 ipmb16 ipmb17 ipmb18 ipmb19 ipmb2 ipmb20 lgrobet lleyns
mattimon mawaelbr mcolet mdenayer nmoguil odelgran odugas
oduterge pafontey pdreze pdujardi umhulb5 umhulb6 umhulb7
```

TAB. 3.1 – Echantillon des comptes utilisateurs repris dans les *logs* Emboss

échantillon de l'ensemble des comptes utilisateurs dont les traces ont été analysées qui illustre bien l'étendue des différents profils. Effectivement, nous voyons bien que la liste présentée contient des comptes “étudiants”, “chercheurs” et “privés”.

La collecte des traces s'est étendue sur une période d'environ un an (entre les années académiques 2003 et 2004) et contient la liste des applications Emboss exécutées par les utilisateurs de différents comptes. C'est l'ensemble de telles listes propres à tous les comptes utilisateurs - le fichiers des *logs* - qui constitue la base de notre analyse. Chaque action d'une trace quelconque contenue dans le fichier des *logs* est définie par une ligne contenant l'application exécutée, le compte utilisateur auquel elle appartient ainsi qu'une marque temporelle détaillée. Les lignes du fichier des *logs* sont rajoutées au fur et à mesure que les actions correspondantes sont exécutées et sont donc triées en ordre chronologique comme illustré dans le tableau 3.2 (nous ne présentons ici que les 20 premières lignes de ce fichier²).

Une analyse attentive du contenu des *logs* nous permet de conclure que les données brutes contiennent une grande quantité d'information que nous pourrions qualifier d'“inutile”. Parmi les données jugées inutiles dans notre cas nous trouvons en premier lieu les lignes contenant des utilisateurs système, tel **gbottu** et **embadmin**. En effet, le comportement de tels utilisateurs ne nous serait d'aucun intérêt, puisqu'ils ne créent rien et que personne ne sera jamais amené à répéter leur actions³. Les autres lignes à éliminer des *logs* sont celles contenant des actions

¹Il s'agit d'une application on-line d'analyse de séquences (www.emboss.org).

²La taille du fichier des *logs* est de 6,864,371 bytes.

³Il s'agit des comptes d'administration du système.

très répandues n'ayant pas de rapport direct avec le but principal des traitements en cours. L'exécution de l'application `wossname`¹ par exemple ne nous donne aucun renseignement sur le travail précis de l'utilisateur, car sert par définition d'une sorte d'aide mémoire et peut être consulté à n'importe quel endroit de la chaîne

<code>showdb</code>	<code>gbottu</code>	Sun Feb 15 18:18:00 2004
<code>entret</code>	<code>gbottu</code>	Sun Feb 15 18:18:14 2004
<code>dotmatcher</code>	<code>gbottu</code>	Sun Feb 15 18:19:15 2004
<code>dotmatcher</code>	<code>gbottu</code>	Sun Feb 15 18:20:06 2004
<code>fasta</code>	<code>gbottu</code>	Sun Feb 15 18:22:37 2004
<code>meme</code>	<code>embadmin</code>	Sun Feb 15 20:19:20 2004
<code>showdb</code>	<code>gbottu</code>	Sun Feb 15 20:21:39 2004
<code>entret</code>	<code>gbottu</code>	Sun Feb 15 20:21:56 2004
<code>tfm</code>	<code>embadmin</code>	Sun Feb 15 20:28:27 2004
<code>pfmake</code>	<code>gbottu</code>	Sun Feb 15 20:36:54 2004
<code>mkdom</code>	<code>gbottu</code>	Sun Feb 15 20:39:07 2004
<code>pfmake</code>	<code>gbottu</code>	Sun Feb 15 20:40:50 2004
<code>mkdom</code>	<code>gbottu</code>	Sun Feb 15 20:41:38 2004
<code>showdb</code>	<code>gbottu</code>	Sun Feb 15 20:50:25 2004
<code>entret</code>	<code>gbottu</code>	Sun Feb 15 20:51:22 2004
<code>dotpath</code>	<code>gbottu</code>	Sun Feb 15 20:56:56 2004
<code>dotpath</code>	<code>gbottu</code>	Sun Feb 15 20:57:25 2004
<code>seqret</code>	<code>ipmb14</code>	Sun Feb 15 21:16:28 2004
<code>wossname</code>	<code>pvdhaegh</code>	Mon Feb 16 09:02:55 2004
<code>wossname</code>	<code>pvdhaegh</code>	Mon Feb 16 09:02:56 2004

TAB. 3.2 – Extrait du fichier des *logs* Emboss

des actions effectuées par l'utilisateur. Il s'agit là donc typiquement d'un élément qui risque de perturber les futures prédictions au même titre que d'autres applications du même genre telles `showdb`², `infoseq`³, etc. (la prédiction proposera de temps en temps aux utilisateurs des applications étrangères à leur sujet de travail). Notons, néanmoins que l'élimination de la plupart des lignes contenant certaines de ces applications relève d'un choix "stratégique" et ne représente aucun danger pour la consistance du système de prédiction (n'y apporte pas de bruit).

En revenant sur l'exemple du petit extrait des *logs* présenté ci-dessus, nous remarquons qu'il n'en restera qu'une seule ligne valable après son "épuration"⁴.

¹Retrouve les applications sur base des mots-clés dans leur documentation on-line.

²Affiche les informations sur les bases de données actuellement disponibles.

³Affiche les informations basiques sur les sequences.

⁴La taille du fichier des *logs* épuré est de 2,220,100 bytes (un rapport $\sim \frac{1}{3}$).

3.1.2 Description des workflows

La spécification du langage BPML¹ fournit un modèle formel capable de décrire des processus abstraits et exécutables de complexité variable, ainsi que les transactions, la gestion de données et la concurrence au sein de ceux-ci. BPML nous fournit également une grammaire XML permettant la réalisation de cette description [BPML, 2002]. C'est ainsi très naturellement que le choix du formalisme de description des workflows s'est arrêté sur ce langage. Bien entendu le cadre d'application du langage initial étant de manière générale beaucoup plus vaste que de "simples" workflows, nous serons obligés de n'utiliser qu'un sous-ensemble restreint des possibilités qui nous sont offertes. Notons que cette partie du travail reste à effectuer et sera très probablement entamé lors de la conception du Workflow, puisque ce n'est qu'à ce moment-là qu'on aura vraiment besoin de décrire les workflows - au moment de leur construction et de leur utilisation (interprétation).

Dans le cadre de ce mémoire la représentation initiale des workflows nous intéresse peu, puisque de toute façon nous n'en avons pas pour l'instant. Le seul point de départ que nous avons c'est le fichier des *logs* qui contient toutes les actions effectuées par tout le monde de manière confondue. Il est donc de notre ressort de commencer par extraire et reconstituer les workflows à partir des *logs* (obtenir l'équivalent des futurs workflows BPML de départ) pour les traduire ensuite dans notre propre représentation de workflows au sein du Workflow Intelligent (la partie intelligente du système utilisera sa propre représentation des données). Un exemple de workflow reconstitué est présenté dans le tableau 3.3². Nous avons

```

<workflow>
  <author> uname <\author>
  <date> 2003/11/21 <\date>
  <contents type=linear>
    compseq          :16:26:01
    backtranseq      :16:41:43
    backtranseq      :16:45:56
    helixturnhelix   :17:05:03
    needle           :17:52:50
    backtranseq      :17:53:01
    charge           :18:27:45
    octanol          :18:28:02
  <\contents>
<\workflow>

```

TAB. 3.3 – Exemple de workflow extrait des *logs* en format pseudo-BPML

essayé de nous rapprocher le plus possible du futur format BPML même si bien

¹Business Process Modeling Language.

²Pour toutes les analyses des *logs* nous avons utilisé le langage de scripting Perl, choisi essentiellement pour sa facilité de fonctionnement ainsi que pour sa vitesse de traitement des données. Pour les différents scripts utilisés dans ce chapitre - consulter la partie Annexe A du mémoire.

évidemment cela n'est aucunement indispensable. Le code de l'exemple est très clair et nous renseigne sur plusieurs aspects du workflow qu'il décrit ; il s'agit d'un workflow effectué sous le compte `uname`, le 21 novembre 2003 et chaque application lancée est marquée par le moment du début de son exécution. Sur ce nous pouvons conclure que les données contenues dans les *logs* sont assez maigres, mais qu'elles peuvent quand même être utilisées pour un certain nombre d'analyses.

3.1.3 Ensembles d'apprentissage

Les algorithmes d'apprentissage retenus précédemment pour la mise en oeuvre de la procédure de prédiction des futures applications qui vont être exécutées par l'utilisateur ne sont pas en mesure de gérer les informations de départ - l'ensemble d'apprentissage - telles que présentées plus haut dans la section `<contents ... >` du code XML des workflows (une liste d'actions). Nous devons donc adapter cette liste d'actions de manière à ce qu'elle soit acceptable comme ensemble d'apprentissage par les différents algorithmes d'apprentissage en question. Pour ce faire nous nous sommes inspirés de l'approche largement utilisée dans les travaux d'analyses et de prédiction des séries temporelles qui consiste à exploiter la relation d'ordonnement chronologique intrinsèque aux données analysées. L'idée de cette approche peut être résumée par la formule suivante :

$$\underbrace{var_k, var_{k+1}, var_{k+2} \dots var_{k+m-1}}_{\text{Une suite de } m \text{ variables consécutives}} \longrightarrow \underbrace{var_{k+m}, var_{k+m+1} \dots var_{k+n}}_{\text{Les } n+1-m \text{ variables consécutives suivantes}}$$

En faisant prendre à $k \in [0; \text{nombreDeVariablesTotal}-n-1]$ toutes ses valeurs possibles, nous obtenons un ensemble d'apprentissage de type *instance* \longrightarrow *concept*. Il est à remarquer que dans le cas des algorithmes d'apprentissage inductif retenus, la suite dans la partie droite de la formule ne peut avoir qu'une seule variable¹.

```

compseq, backtranseq, backtranseq
backtranseq, backtranseq, helixturnhelix
backtranseq, helixturnhelix, needle
helixturnhelix, needle, backtranseq
needle, backtranseq, charge
backtranseq, charge, octanol

```

TAB. 3.4 – Traduction du workflow en ensemble d'apprentissage.

Nous reviendrons ultérieurement sur le format définitif de représentation des workflows traduits en ensembles d'apprentissage (acceptables par nos algorithmes).

3.1.4 Hypothèses

Même si les traitements proposés dans les sections précédentes nous permettent de construire une solide base indispensable à la future validation de notre système

¹Il y a moyen de surmonter cette limite à l'aide de l'apprentissage des habitudes.

de prédiction, force est tout de même de constater que tout au long de notre réflexion nous avons ignoré un certain nombre de circonstances qui pourraient plus tard s'avérer décisives au bon fonctionnement du système. Ainsi, l'objectif de cette section est de reprendre chacune des hypothèses effectuées, afin d'essayer d'en évaluer l'impact potentiel sur la qualité des prédictions du système :

- **Comptes utilisateurs.** Nous supposons depuis le départ que les comptes repris dans le fichiers des *logs* appartiennent à une seule personne physique, ce qui n'est vrai que très rarement. En effet, plusieurs comptes ont été créés pour être partagés par des groupes d'utilisateurs, ce qui résulte en un mélange de workflows des utilisateurs concernés. En considérant que les suites d'actions des utilisateurs d'un même compte ne se chevauchent pas (ce qui généralement devrait être le cas, chaque utilisateur ayant sa session), l'impact de ce mélange devrait se traduire en un rajout d'un ensemble de fausses règles dans l'ensemble d'apprentissage final lors de sa construction.
- **Découpe en workflows.** La découpe de la suite des actions extraites des *logs* en workflows représente la tâche principale du processus de construction de l'ensemble d'apprentissage. Même lorsqu'il s'agit d'une suite d'actions effectuées par un même utilisateur, il est extrêmement difficile d'identifier de manière précise les limites des différents workflows. En effet, comment déterminer qu'une application donnée est une action faisant partie d'une chaîne de traitement bien précise et non une action isolée n'ayant rien à voir avec les actions précédentes ? Dans ce cas, il n'existe pas de solutions fiables sûres à 100%. La solution qui nous a semblé être la plus appropriée dans cette situation est une découpe en workflows sur base des dates ; nous considérons que toutes les actions effectuées un jours donné constituent un workflow. La seule manière possible d'améliorer cette approche serait de se baser sur l'heure. Malheureusement, l'heure nous semble être encore plus trompeuse que la date, car rien n'indique qu'une grande pause entre deux actions (le séparateur de workflows le plus probable) signifie le début d'une autre chaîne de traitement. Ici, encore une fois, une mauvaise découpe implique l'ajout à l'ensemble d'apprentissage de fausses règles.
- **Perte de données.** La traduction des suites d'actions en ensemble d'apprentissage implique une perte de certaines suites, lorsque le nombre d'actions de celles-ci n'est pas suffisant pour construire une règle.
- **Eléments parasites.** Il s'agit d'actions isolées effectuées de manière "aléatoire" à l'intérieur d'une chaîne d'actions constituant un traitement complexe. De manière générale, nous ne pouvons rien faire pour empêcher ce genre de bruit, si ce n'est enlever de la suite des actions les applications susceptibles d'être de tels parasites. C'est d'ailleurs ce que nous avons fait en éliminant un certain nombre d'actions des *logs*.
- **Actions séquentielles.** Dans le même ordre d'idées que la remarque précédente, notons que l'approche très restrictive qui consiste à considérer les suites d'actions analysées comme étant les seules possibles pour atteindre le but fixé par l'utilisateur (ce qui est implicitement le cas), diminue considérablement les capacités prédictives des algorithmes utilisant ces suites comme ensemble d'apprentissage. Une construction de l'ensemble d'apprentissage qui prendrait en compte les différentes permutations possibles de certaines

actions des workflows permettrait au système d'acquérir une souplesse additionnelle, indispensable au fonctionnement correct dans des conditions réelles.

- **Structure arborescente.** Une autre hypothèse majeure d'ordre général - dépassant largement le cadre de l'analyse des *logs* Emboss - que nous avons appliqué jusqu'ici consiste à considérer les workflows comme étant de simples suites linéaires d'actions. Or, la vraie puissance d'expression des workflows réside justement dans leur capacité de décrire des processus beaucoup plus complexes qu'un simple enchaînement linéaire d'actions, et c'est d'ailleurs dans ce contexte que leur définition a été introduite. Ainsi, une description arborescente des workflows correspond à un formalisme général, au sein duquel les suites d'actions linéaires constituent des cas particuliers. Plus précisément, les descriptions de workflows exécutés correspondent à des DAG¹. En effet il s'agit de graphes, car nous pouvons leur appliquer les notions de noeuds (les services exécutés) et celle des arcs dirigés (enchaînements de services). Les workflows exécutés sont acycliques par définition : s'ils ne l'étaient pas, ils ne se termineraient jamais (boucle infinie dans les cycles).

-
- *Rajouter l'origine aux sommets visités.*
 - *Effectuer un traitement de l'origine*
 \hookrightarrow *Procédure de parcours en profondeur dans le sens inverse*
Passer en paramètre la liste vide des prédécesseurs.
 - \forall *sommet adjacent*
 \hookrightarrow *Procédure de parcours en profondeur de manière récursive.*
-

TAB. 3.5 – Algorithme récursif de parcours en profondeur de graphes

Dans le cas des workflows représentés sous forme de DAG, nous pouvons facilement appliquer le même format de description des ensembles d'apprentissage que précédemment. L'idée de base reste la même, à savoir qu'un ensemble d'actions séquentielles impliquent l'action suivante, mais l'algorithme de construction de l'ensemble d'apprentissage change. Comme on n'a pas la description BPML initiale des workflows nous ne pouvons pas encore

-
- *Rajouter l'origine aux sommets visités.*
 - *Rajouter l'origine dans la liste des prédécesseurs*
 \hookrightarrow *(tailleListe == nombreVariablesRègle)*
{imprimer la liste en ordre inverse comme étant la règle apprise}
 - \forall *sommet adjacent (les pères)*
 \hookrightarrow *Procédure de parcours en profondeur dans le sens inverse*
Passer en paramètre la liste des prédécesseurs.
-

TAB. 3.6 – Parcours inverse de graphes avec impression des règles

¹Directed Acyclic Graph (Graphe Acyclique Dirigé)

implémenter la traduction du code BPML en graphe comme structure de données (en un langage de programmation quelconque), mais nous pouvons par contre proposer un algorithme de construction de l'ensemble d'apprentissage en supposant que le graphe existe déjà. Dans ce cas, il suffit de parcourir tous les éléments du graphe en effectuant pour chaque noeud une procédure inverse qui consiste à remonter ses parents en ordre chronologique (ordre implicite à la construction du graphe) jusqu'à un certain niveau précis (le nombre de variables des règles) en mémorisant à chaque étape le chemin parcouru. En imprimant l'inverse du chemin parcouru lorsque l'algorithme atteint les derniers ancêtres en remontant - nous obtenons un ensemble de règles ayant comme concept le noeud de départ. Les tableaux 3.5 et 3.6 présentent respectivement les pseudo-codes de la procédure "normale" de parcours des graphes en profondeurs et de celle du parcours des ancêtres.

3.1.5 Améliorations possibles

L'ensemble des hypothèses invoquées peut se subdiviser en deux grandes catégories : les hypothèses propres aux données recueillies auprès des *logs* Emboss et celles, plus générales, concernant le fonctionnement futur du système de prédiction de manière globale¹. Si les hypothèses de la première catégorie ne posent pas de problèmes à long terme, la situation est toute autre dans le deuxième cas. En effet, il serait intéressant de pouvoir prendre en compte la gestion des actions "accidentelles", ainsi que celle des traitements "semblables" effectués en désordre.

Pour résoudre ces problèmes, nous proposons une approche semblable à celle employée dans le cadre de l'apprentissage des habitudes. Une bonne manière d'aborder le sujet serait d'appliquer les notions de situation et de modèle² à notre ensemble d'apprentissage : dans chacune des règles apprises, le concept serait la tâche répétitive et l'ensemble des actions qui le précèdent - la situation dans laquelle cette tâche a été effectuée. La formule suivante résume cette approche :

$$\underbrace{(A_1 a A_2 b c A_3)}_{\text{Situation}_1} \longrightarrow C^1, \underbrace{(A_1 d A_2 e f A_3)}_{\text{Situation}_2} \longrightarrow C^2 \implies \underbrace{([A_1 * A_2 * * A_3])}_{\text{Modele}} \longrightarrow C$$

Ayant deux règles avec un concept commun (exemples de ce concept), nous pouvons par la suite déduire le modèle de situation (ordonnés ou non) dans lequel l'action correspondant à ce concept est effectuée. Il ne s'agit plus de C4.5.

Nous pouvons considérer que l'ensemble des modèles de situations correspond à l'ensemble **S** de la généralisation la plus spécifique des exemples comme défini par [Mitchell, 1997]. Rappelons que l'espace des versions d'un concept peut être vu comme l'ensemble des hypothèses cohérentes avec l'ensemble d'apprentissage (c'est-à-dire qui couvrent tous les exemples et aucun contre-exemple) et que cet espace peut être représenté soit par les ensembles **S** et **G** (la généralisation des éléments les plus généraux), soit par l'ensemble **S** et l'ensemble des contre-exemples E_c (le reste de l'ensemble d'apprentissage) [Hirsh, 1992]. L'ensemble $[S, E_c]$ (comme $[S, G]$) permet de prédire le concept auquel appartient une instance donnée (tableau 3.7). Malheureusement lorsque l'ensemble d'apprentissage

¹Elles sont marquées par un cercle vide.

²Voir la section 1.8.1

-
- Instance couverte par les hypothèses de \mathbf{S} \longrightarrow exemple.
 - Sinon utiliser l'instance pour mettre \mathbf{S} à jour
 \hookrightarrow Si le résultat est un ensemble vide \longrightarrow contre-exemple
 Sinon \longrightarrow impossible de dire.
-

TAB. 3.7 – Algorithme de Hirsh.

contient du bruit il est problématique de construire \mathbf{S} . C'est la raison pour laquelle d'autres approches sont envisagées dans ce cas. Notons par exemple IDHYS, l'algorithme proposé par [Ruvini, 2000] qui procède par construction d'un ensemble \mathbf{S}_c approximatif pour chaque concept c de l'ensemble d'apprentissage et dont les hypothèses sont retenues sur base de la mesure de qualité de Laplace. Une fois les ensembles \mathbf{S}_c calculés la prédiction se fait à l'aide de l'algorithme présenté dans le tableau 3.8 (l'exemple basique de deux concepts se généralise très facilement).

-
- Instance couverte par des hypothèses de \mathbf{S}_1 seulement
 \hookrightarrow Probablement un exemple de c_1 .
 - Instance couverte par des hypothèses de \mathbf{S}_2 seulement
 \hookrightarrow Probablement un exemple de c_2 .
 - Instance couverte par aucune hypothèse
 \hookrightarrow Impossible de prédire le concept.
 - Instance couverte par des hypothèses de \mathbf{S}_1 et de \mathbf{S}_2
 \hookrightarrow Exemple de c_1 si qualité et nombre d'hypothèses plus important
 \hookrightarrow Exemple de c_2 sinon.
-

TAB. 3.8 – Algorithme de prédiction IDHYS pour deux concepts c_1 et c_2 .

Il est évident que l'approche de modélisation des situations "décrivant" le contexte dans lequel les différentes applications sont exécutées exploite beaucoup plus les informations contenues dans les futurs workflows experts de la IKDB (l'ensemble d'apprentissage) que, *grosso modo*, une simple recherche de cas identiques. Mais elle exige également un certain nombre de pré-requis indispensables à sa mise en oeuvre tels que, par exemple, la hiérarchie sous entendue tout au long de nos explications sur les actions des utilisateurs (les applications bio-informatiques)¹.

3.2 Méthodologie

L'ambition finale de notre travail étant la conception et la réalisation pratique du Workflow Intelligent, nous ne pouvons bien évidemment pas envisager une quelconque étude sérieuse de celui-ci sans avoir recours à de réels tests de

¹La classification du consortium Gene Ontology GO pourrait éventuellement convenir.

fonctionnement en “grandeur nature”. Et c’est justement dans cette optique-là que nous avons abordé cette dernière partie du travail ; nous avons commencé par répondre à un certain nombre de questions posées dans le Cahier des charges lors de la conception théorique de notre système intelligent tout en introduisant un ensemble de données réelles dont nous comptons nous servir par la suite pour évaluer expérimentalement les performances potentielles de notre système de prédiction.

3.2.1 Expérimentations

L’objectif principal de nos expérimentations est de donner l’approximation la plus fidèle possible de la qualité des prédictions que le système envisagé sera capable de fournir. Pour que cette évaluation de performances soit représentative de l’utilisation réelle du système, nous sommes obligés de gérer d’une manière ou d’une autre l’extrême hétérogénéité des données de tests disponibles. Ultérieurement l’équivalent de cette pré-sélection sera effectué par les routines CBR situées sur les médiateurs et chargées de construire la *IKDB* sur demande des machines clientes.

3.2.1.1 Pré-traitement des données

Ainsi, les données de test recueillies auprès des *logs* Emboss constituent le point de départ de nos expérimentations. Pour essayer de tenir compte le plus possible de l’ensemble des hypothèses simplificatrices effectuées dès le départ sur la (faible) qualité des données de test, il est indispensable de définir un moyen efficace de construction des ensembles d’apprentissage valides de bonne qualité. Ces ensembles serviront à tester les performances de nos différents algorithmes de prédiction et nous permettrons, donc, d’obtenir des résultats représentatifs du fonctionnement global du système sur nos données de test. Notons que théoriquement les résultats obtenus sur nos données de test présentent une sorte de limite inférieure des performances du système, car compte tenu de leur mauvaise qualité et de la taille relativement restreinte de notre base de données, nous sommes en droit d’attendre des taux de performance nettement meilleurs lors de l’utilisation réelle du système.

L’idée principale de la méthode adoptée s’inscrit de manière générale dans la même logique que celle consistant à choisir parmi l’énorme quantité de workflows stockés au sein des bases de données des médiateurs ceux qui répondent le mieux à une certaine description donnée. Nous effectuons une sorte de regroupement des workflows mis à notre disposition afin de former des *groupes* représentatifs consistants en workflows “semblables”. Il est en effet évident que les performances des algorithmes de prédiction seront meilleures lorsqu’ils sont exécutés sur des groupes composés des workflows ayant le plus d’éléments possible en commun. Le seul problème majeur de cette approche consiste dans le fait qu’il est assez compliqué de déceler les workflows “semblables” au sein de notre base de données. La notion de similarité même est très difficilement mesurable dans notre cas, car une comparaison de deux (ou plus) workflows suppose par définition une description extérieure intelligente de chacun des workflows en question. Ainsi, à défaut d’avoir les descriptions nécessaires à l’utilisation des méthodes bien adaptées à la tâche de regroupement (l’ensemble des algorithmes de classification non-supervisés - *le clustering*), nous avons choisi une approximation de l’algorithme des *k* plus proches voisins qui permet d’obtenir un résultat semblable sur base des critères bien définis.

Notre méthode consistant à construire les groupes diffère de l'algorithme k - nn de base dans le sens où la comparaison n'est pas basée sur les attributs de chaque cas (workflow) puisqu'ils n'en ont pas, mais reste bien similaire à celui-ci en ce qui concerne la fonction de mesure de similarité et la construction de l'ensemble des k voisins les plus proches. Plus précisément, nous construisons l'ensemble des k plus proches comptes pour chaque compte utilisateur¹. Le regroupement des différents comptes typiques se fait sur base d'une mesure de similarité calculée comme suit :

$$f_{compte} = \begin{cases} f_{compte} + 1, & \text{si application commune} \\ f_{compte} - 1, & \text{sinon.} \end{cases}$$

Ceci suppose bien évidemment l'extraction au préalable pour chaque compte de la liste des applications effectuées sous ce compte. Lorsqu'un compte est comparé au compte cible (pour lequel on construit l'ensemble des voisins) la formule présentée ci-dessus est appliquée pour chaque application des deux comptes. La valeur finale de la fonction est retenue comme mesure de similarité. Sa valeur initiale est zéro.

Intuitivement, pour un compte cible donné la fonction favorise les comptes ayant le plus d'applications identiques à la cible et le moins possible d'applications différentes. Notons que cette approche n'est de loin pas la seule possible², et n'est peut-être même pas la plus efficace parmi toutes les méthodes de classement possibles. Son inconvénient principal est le fait qu'elle défavorise les comptes ayant beaucoup d'applications différentes alors qu'ils pourraient contenir un nombre important de situations intéressantes constituées par les applications communes. Néanmoins elle a le mérite de regrouper les comptes qui ont tendance à limiter leur champs d'action à un ensemble restreint de mêmes applications. Cette situation est typique aux environnements académiques, lorsque plusieurs novices travaillent sur des sujets données. Ce qui justement est le cas de certains comptes Emboss³.

3.2.1.2 Analyse des groupes

Ces analyses ont pour but la mesure des performances (le taux de prédiction) des algorithmes potentiellement utilisables pour prédire les actions de l'utilisateur. La méthode utilisée est analogue à la validation croisée⁴. La validation croisée est utilisée en apprentissage lorsque l'on ne dispose que d'un seul échantillon que l'on doit utiliser à la fois pour apprendre et pour mesurer le taux d'erreur des hypothèses apprises. Elle consiste à diviser l'échantillon en k sous-échantillons, ou blocs⁵, à écarter un des blocs, à apprendre sur les $k - 1$ restants (qui constituent donc l'ensemble d'apprentissage), pour finalement mesurer le taux d'erreur des hypothèses apprises sur le bloc écarté (ensemble de test). Le processus est répété k fois en excluant tour à tour chacun des blocs. La moyenne des k taux d'erreur mesurés donne alors une estimation de la probabilité d'erreur de l'algorithme.

¹Il s'agit d'un choix dicté par le fait que les comptes utilisateurs se situent à un niveau d'abstraction plus élevé que celui des workflows et qu'ils sont dès lors plus résistants contre les incohérences dans les données. Néanmoins, les deux approches sont intéressantes à tester.

²Nous pourrions même nous contenter d'un choix aléatoire.

³Les résultats pratiques de regroupement semblent confirmer cette constatation.

⁴Connue également sous le nom de *k-fold Cross Validation*.

⁵Les valeurs de k les plus utilisées dans la littérature sont $k = 3$ et $k = 10$. Lorsque k est égal à la taille de l'échantillon n , il s'agit alors de la méthode qu'on appelle *Leave One Out*.

En considérant les groupes obtenus à l'étape de pré-traitement des données comme étant les échantillons de départ, la première approche d'analyse des performances consiste à se placer dans le contexte du fonctionnement typique du Workflow Intelligent sur les clients; dans ce cas nous pouvons considérer que les blocs à écarter sont tout simplement les workflows effectués par les utilisateurs des différents comptes représentés dans le groupe analysé et que l'ensemble des autres workflows du groupe constituent l'ensemble d'apprentissage - la *IKDB*. La validation croisée nous renseigne donc dans ce cas sur le pourcentage de prédictions correctes pour chaque workflow sur base de ses k plus proches voisins (workflows ou comptes) et nous fournit une mesure similaire pour tout le groupe à savoir la moyenne sur tout les workflows. Il est à remarquer qu'en utilisant une telle approche, nous passons sous silence un nombre de caractéristiques importantes propres à nos données. En effet, le caractère temporelle des données implique un certain ordre sur les liens logiques entre les différents ensembles de tests et les ensembles d'apprentissage correspondant; nous ne pouvons pas nous baser sur un workflow exécuté aujourd'hui pour prédire les actions d'un workflow effectué il y a deux mois. Néanmoins, pour simplifier nous faisons l'hypothèse qui consiste à ignorer les dates. Cette hypothèse est d'autant plus justifiée que la qualité de nos données est extrêmement pauvre et que nous ne pouvons dès lors pas nous permettre d'ignorer le peu de données (peut-être) correctes disponibles.

Une autre limite à l'obtention de résultats fiables par la validation croisée au niveau des workflows provient encore une fois des caractéristiques mêmes des données mises à notre disposition; la moyenne sur un groupe de workflows ne reflète une image fidèle de l'ensemble de ces workflows que lorsque les blocs écartés de l'échantillon sont de tailles égales ou du moins comparables. Or dans notre cas, nous n'avons aucune garantie sur les tailles des workflows écartés. L'idée qui consisterait à pondérer les taux des différents workflows (par la taille du bloc par exemple) ne convient pas, puisqu'elle fausserait bien évidemment le résultat final.

La deuxième approche que nous avons testé propose une solution à ce problème de la moyenne. L'approche consiste à descendre d'un pas du niveau d'abstraction correspondant aux workflows pour s'intéresser aux règles qui les composent. Les groupes restent toujours les échantillons de départ, mais cette fois-ci au lieu d'écarter des workflows en tant que blocs de test, nous écartons des ensembles de règles obtenus par une découpe préalable de l'échantillon en blocs de tailles égales. Le problème dans ce cas est que plusieurs morceaux de différents workflows se retrouvent dans un même bloc et ne sont jamais utilisés pour prédire l'un à partir des autres alors que cela pourrait très certainement être intéressant parfois. Notons que ceci n'est pas très grave puisque nous ne sommes pas intéressés par des chiffres exacts (qui d'ailleurs sont impossibles à obtenir sur nos données), mais plutôt par l'ordre de ces chiffres. Or, puisqu'il est clair que cette hypothèse ne peut que baisser les performances observées, nous pouvons nous contenter de la limite inférieure obtenue. La situation inverse peut également se produire; un assez grand workflow peut être découpé en plusieurs blocs¹. Cette limite peut également être mise de côté dans notre cas. En effet, si nous considérons que les dates ne sont pas prises en compte, alors nous pouvons aller plus loin et considérer que pour des raisons semblables les règles constituant un workflow donné peuvent être

¹La situation se produit systématiquement lors du *loo*.

utilisées les unes pour prédire les autres. Après tout, ce qui nous intéresse est de pouvoir prédire une action (le concept) dans une situation précise (instance), et non dans un contexte différent (situations précédentes ou suivantes). Le contexte dans notre système est pris en compte de manière implicite, puisque nous nous limitons depuis le départ à n actions consécutives précédant l'action recherchée.

3.2.1.3 Cas considérés

Le processus de test des performances que nous avons mis en oeuvre peut être vu comme étant conceptuellement composé de deux ensembles distincts de manipulations : le pré-traitement des données à analyser et l'analyse proprement dite. Le but du pré-traitement est d'obtenir les échantillons à tester (les groupes contenant les workflows "similaires"), et celui de l'analyse est de procéder à l'évaluation des performances sur ces échantillons. Pour pouvoir tester tous les cas intéressants dont nous avons discuté auparavant, nous sommes obligés de prendre des décisions mutuellement exclusives à chaque étape des manipulations. Sur la figure 3.1 nous détaillons schématiquement la plupart des choix intéressants à effectuer¹. Nous voyons par exemple qu'avant de pouvoir procéder à l'exécution de notre algorithme des k plus proches voisins, nous sommes obligés de faire deux choix : le premier est effectué au niveau de la manière de regrouper les workflows (sur base

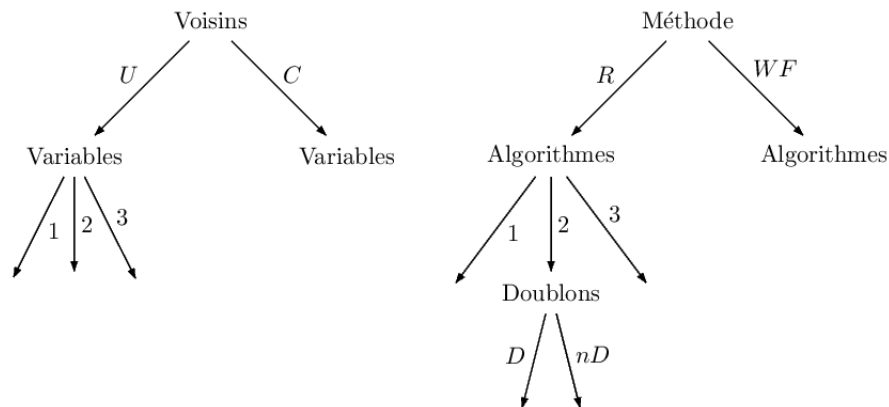


FIG. 3.1 – Choix possibles des configurations des expériences à l'étape de pré-traitement des données (à gauche) et lors de l'analyse des groupes (à droite).

des comptes C ou des workflows U) et le second concerne le choix de la taille des règles. La taille d'une règle est égale à la taille de l'instance (nombre de variables) à laquelle on rajoute un concept. Ici, nous limitons la taille des variables à trois².

¹Il y a d'autres choix plus complexes, comme la manière de construire les workflows. Cependant nous avons choisi de n'effectuer les tests qu'avec une seule manière de découper les tâches en workflows. La justification de cette approche réside dans le fait que nous ne cherchons pas à comparer les différentes découpes (de toute façon rien ne nous indiquera la "bonne" découpe). Nous essayons d'évaluer les performances possibles du meilleur cas théorique (1 jour = 1 workflow).

²Une taille plus importante devrait être testée sur des données plus sûres.

Pour procéder à la mesure des performances des algorithmes sur les groupes il faut d'abord définir la méthode de test. Il s'agit alors soit de l'adaptation de la validation croisée au niveau des workflows *WF*, soit de la validation croisée de base sur les règles *R*. Ensuite, pour chacun des cas nous effectuons systématiquement les tests sur tous les algorithmes retenus. Le dernier critère de configuration consiste à indiquer si les règles qui se répètent dans la définition d'un workflow donné doivent être prises en compte lors des tests ou non. En effet, le fait de permettre de telles répétitions peut avoir un impact considérable sur les performances observées et ce de plusieurs manières ; lorsque les prédictions sont par exemple effectuées sur un workflow avec un grand taux de répétitions (dans le cas de la validation *WF*), il suffit de prédire un seul concept pour multiplier le score de prédiction final de ce workflow par un facteur parfois assez élevé. Il est ainsi plus intéressant de ne prédire chaque concept qu'une seule fois dans ce cas. De l'autre côté éliminer systématiquement les doublons de tous les workflows n'est pas forcément une bonne solution non plus, car *normalement*¹ le fait d'avoir des répétitions dans l'ensemble d'apprentissage reflète la réalité du terrain et leur élimination risque de fortement perturber le modèle construit par l'algorithme de prédiction et par conséquent les prédictions que l'algorithme en question fournit. Le compromis est donc d'éliminer les doublons de l'ensemble de test et de les laisser dans l'ensemble d'apprentissage, ce qui malheureusement lors de la validation *R* est impossible. Dans ce cas nous éliminons les doublons (si tel est le choix) de tous les workflows dès le départ en espérant que l'impact ne sera pas important. Nous retombons alors encore une fois sur une sorte de limite inférieure des performances comme précédemment.

3.2.2 Weka

Le projet Weka a été mis sur pieds dans le but de concevoir une plate-forme unifiée présentant l'état de l'art de l'ensemble des techniques d'apprentissage et permettant de les appliquer aux problèmes de *data mining* du monde réel². Un grand nombre de techniques d'apprentissage standards ont été incorporés dans un *workbench* appelé WEKA (Waikato Environment for Knowledge Analysis). Cet outil permet aux spécialistes de tous horizons d'appliquer des techniques d'apprentissage à leur propres besoins spécifiques de manière complètement transparente.

3.2.2.1 Format .arff

Nous avons déjà parlé de la traduction des suites d'actions extraites de nos *logs* vers des ensembles d'apprentissage acceptables (théoriquement) par des algorithmes de prédictions. Cette traduction aura également lieu bien évidemment dans le futur Workflow Intelligent. Il est aussi évident que telle que nous l'avons présentée, elle n'est pas encore utilisable sur des algorithmes réels. Ainsi, puisque notre choix d'outil technique s'est arrêté sur WEKA nous proposons d'adapter notre description des workflows de manière à ce qu'elle soit compatible avec les algorithmes qui nous sont offerts par celui-ci. Autrement dit, nous allons convertir notre ensemble d'apprentissage dans le format *.arff*. Le format *.arff* (Attribute-Relation File Format) a été conçu spécialement dans le cadre de Weka et est en train de devenir un standard *de facto* de description de listes d'instances parta-

¹Ce n'est très probablement pas le cas de nos workflows fabriqués par extrapolation.

²De ce point de vue Weka poursuit les mêmes objectifs que le projet Bigre.

geant un ensemble d'attributs (ensembles d'apprentissage) dans la communauté scientifique¹.

Les fichiers `.arff` sont composés de deux parties; la première définit l'en-tête contenant la description des propriétés des données du fichier et la deuxième partie décrit les données à proprement parler. Dans le tableau 3.9 nous reprenons l'exemple du workflow introduit dans la section précédente, dont les détails de la structure sont les suivants : l'en-tête contient le nom de la relation et une liste d'attributs. Chacun de ces éléments est défini par une ligne commençant par le signe `@`. Le mot-clé `@relation` est directement suivi par le titre de l'ensemble d'apprentissage, alors que chacun des mots-clés `@attribut` doit obligatoirement être suivi d'abord par le nom de l'attribut et ensuite par une liste de valeur que cet attribut peut prendre. La définition d'un attribut peut éventuellement préciser le type de l'attribut au lieu de donner la liste des valeurs (par exemple `NUMERIC`). La section des données contient l'ensemble des règles comme obtenu précédemment (ensemble de listes d'attributs - instances et concept - séparés par des virgules) et

```
% This is a learning dataset.
@relation EmbossLearningDataset

@attribute one {backtranseq charge compseq helixturnhelix needle octanol}
@attribute two {backtranseq charge compseq helixturnhelix needle octanol}
@attribute thr {backtranseq charge compseq helixturnhelix needle octanol}

@data
% <user> uname <\user> <date> 2003/11/21 <\date>
compseq, backtranseq, backtranseq
backtranseq, backtranseq, helixturnhelix
backtranseq, helixturnhelix, needle
helixturnhelix, needle, backtranseq
needle, backtranseq, charge
backtranseq, charge, octanol
```

TAB. 3.9 – Exemple de description de workflow en format `.arff`

est déclarée par le mot-clé `@data`. Des commentaires sont admis dans toutes les parties du fichier. Ils sont déclarés par un `%` au début de la ligne commentée.

3.2.2.2 Algorithmes Java

Le système est implémenté en Orienté-Objet dans le langage de programmation Java et a été portée sur tous les systèmes d'exploitation majeurs. WEKA est disponible en format `.jar` et contient l'ensemble des codes sources de la plupart des algorithmes d'apprentissage sous forme de hiérarchie de classes `.class` et d'applications `.java`. Et même si le système fournit une interface graphique

¹Ceci est dû en grande partie au caractère *Open Source* du projet.

de manipulation des algorithmes très intuitive¹, sa conception même permet l'invocation des méthodes implémentées à partir d'une ligne de commande `shell`. L'interface d'appel des différentes méthodes (*classifiers*, *clusters*, *filters*, etc.) a même été uniformisée de manière justement à faciliter la tâche des utilisateurs désireux d'utiliser les morceaux de code qui les intéressent dans leurs propres applications extérieures. Ceci est exactement notre cas. Pour des raisons purement techniques - essentiellement liées au fait que la grande partie du travail consiste en un traitement de fichiers - nous avons opté pour une implémentation Perl de nos expérimentations². C'est ainsi que nous faisons appel aux routines WEKA chaque fois qu'un ensemble d'apprentissage et un ensemble de test (s'il ne s'agit pas d'une validation croisée) sont prêts à être utilisés. Voici un exemple d'appel en ligne de commande typique (l'algorithme C4.5 avec élagage) :

```
system("java weka.classifiers.trees.J48 -t $l_set -T $t_set -i");
```

Pour plus de renseignements sur le format des fichiers `.arff`, les détails techniques d'implémentation des différents algorithmes ou toute autre information supplémentaire concernant le projet Weka - consulter [Witten and Frank, 2000].

3.3 Résultats

3.3.1 Cas de base

Il est évident que parmi la multitude de configurations d'expérimentations possibles (la figure 3.2.1.3), le choix du cas de base dépend principalement des objectifs poursuivis par la mise en oeuvre des expériences en question. Rappelons que dans notre cas l'objectif primaire de l'analyse des données Emboss consiste en une évaluation approximative des performances que les algorithmes d'apprentissage choisis sont capables de fournir sur des données de ce type (notamment sur les futurs workflows réels). Ainsi notre cas de base doit être représentatif de la situation la plus générale possible afin de nous renseigner sur les pires résultats possibles (la borne inférieure des performances). Une telle configuration doit avant tout prendre en compte toutes les données disponibles pour les tests. En effet dans ce cas nous nous plaçons dans la position la plus générale possible en ce qui concerne la taille de l'ensemble d'apprentissage. Le deuxième point important est de s'assurer que les données ne contiennent pas de doublons, car ceux-ci ont très naturellement tendance à hausser le taux de prédiction³. Enfin les tests doivent être effectués sur tous les algorithmes de prédiction concernés. Les autres options qui restent n'ont pas vraiment d'impact prévisible sur les performances attendues et relèvent plutôt du domaine des hypothèses à valider expérimentalement. Parmi les hypothèses effectuées nous retrouvons avant tout la manière de construire les k plus proches voisins (le regroupement des comptes utilisateurs) et la méthode de validation (la plus simple à mettre en oeuvre - la validation croisée au niveau des règles).

Le premier résultat expérimental est présenté sur la figure 3.2. L'expérience consiste à construire pour chaque compte utilisateur disponible une liste ordonnée

¹Nous la présentons dans l'Annexe B.

²Rappelons que tous les codes sources de nos expériences sont présentés dans l'Annexe.

³En théorie ceci n'est pas toujours le cas, mais la pratique semble confirmer la tendance.

de k plus proches voisins¹ et d'effectuer ensuite pour cet utilisateur la validation R sur les k ensembles d'apprentissage constitués chacun des k voisins les plus proches de l'utilisateur ($k \in [2; k_{max}]$). Pour construire le résultat final sur l'ensemble des comptes utilisateurs, nous utilisons *la médiane*. En effet, la médiane a le mérite d'être beaucoup moins sensible aux extrêmes que par exemple la moyenne, ce qui dans notre cas est très important puisque justement nous nous intéressons plus aux différentes valeurs possibles des prédictions qu'à leur moyenne sur un jeu de données précis. Les résultats obtenus dans le cas où les workflows des utilisateurs sont constitués par des règles ayant un attribut et un concept (des instances d'une variable) nous fournissent un taux minimal de prédiction avoisinant les 40%. La dynamique des différents algorithmes utilisés pour les tests semble favoriser le C4.5 sans élagage et ce pour les deux valeurs de k utilisées pour la validation croisée. Notons que ceci n'est pas vraiment étonnant, puisque nous savons que lors de l'élagage le modèle d'apprentissage est généralisé ce qui dans le cas de peu de données (peu de voisins) a tendance de diminuer les performances des classificateurs. Nous voyons également que les performances de l'algorithme élagué s'améliorent

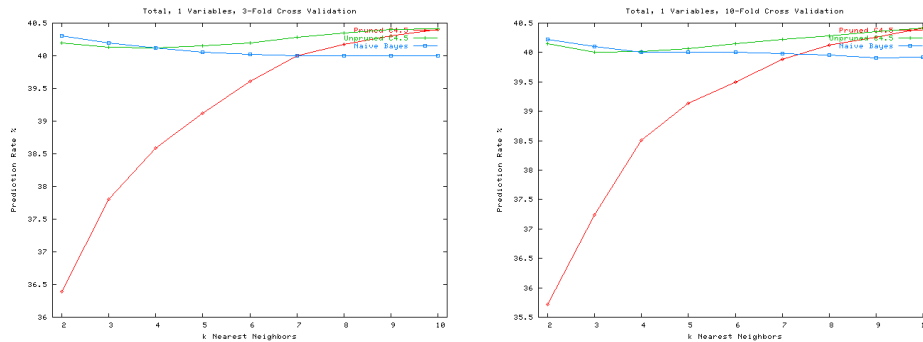


FIG. 3.2 – Evaluation du taux de prédiction en fonction du nombre de voisins sur l'ensemble des workflows disponibles. Les règles sont constituées de deux variables.

rapidement avec l'augmentation de la quantité des données et finissent même par atteindre ceux de l'algorithme non-élagué. La tendance est d'ailleurs beaucoup plus flagrante sur la figure 3.3 qui représente exactement le même cas de base que celui analysé précédemment, mais en augmentant le nombre de variables des règles. De manière générale, élaborer les règles permet de diminuer le nombre de prédictions possibles étant donné une instance et résulte donc *généralement*² en une augmentation des performances de tous les classificateurs. En chiffres l'augmentation des performances dans ce cas-ci correspond à $\pm 5\%$. Un dernier détail à noter concerne l'algorithme naïf de Bayes. En effet, dans le premier cas non seulement il arrive à suivre le C4.5 sans élagage mais il le bat même sur des ensembles d'apprentissage de petites tailles. Mais lorsque les tailles de ces ensembles commencent à augmenter ses performances diminuent. La situation est encore pire lorsque les règles deviennent plus complexes (deuxième cas); un écart considérable se creuse

¹Pour de raisons de faible qualité des données Embossed et du temps de calcul assez élevé, nous limitons le nombre de voisins à dix (il s'agit alors en général de quelques dizaines de workflows). Notons que dans ce cas les tailles des ensembles d'apprentissage peuvent aller jusqu'à 5000 règles.

²Ceci cesse évidemment d'être le cas lorsque le nombre de variables devient trop important.

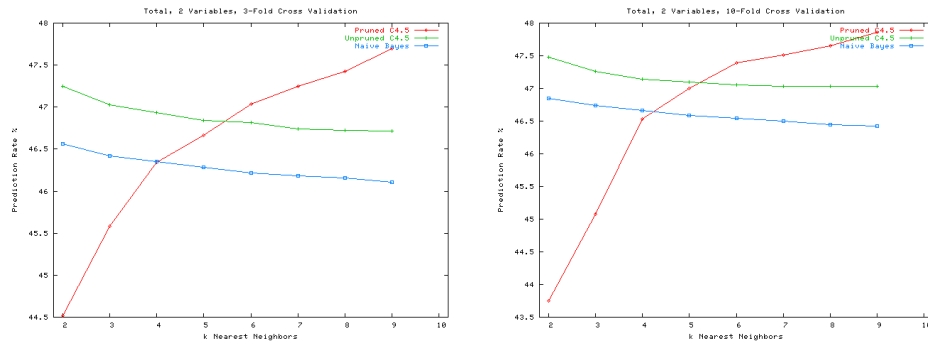


FIG. 3.3 – Evaluation du taux de prédiction en fonction du nombre de voisins sur l’ensemble des workflows disponibles. Les règles sont constituées de trois variables.

entre les deux algorithmes pour toutes les valeurs de k .

Lorsque nous augmentons la taille des règles d’une variable de plus nous obtenons les résultats présentés sur la figure 3.4. Tous les points soulignés lors de la transition des règles d’un seul attribut vers des règles de deux attributs s’accroissent encore plus. La question est alors de trouver la limite supérieure du nombre (optimal) d’attributs des instances des règles. Cela doit se faire sur des workflows réels, en prenant des jeux de données bien précis et de manière expérimentale.

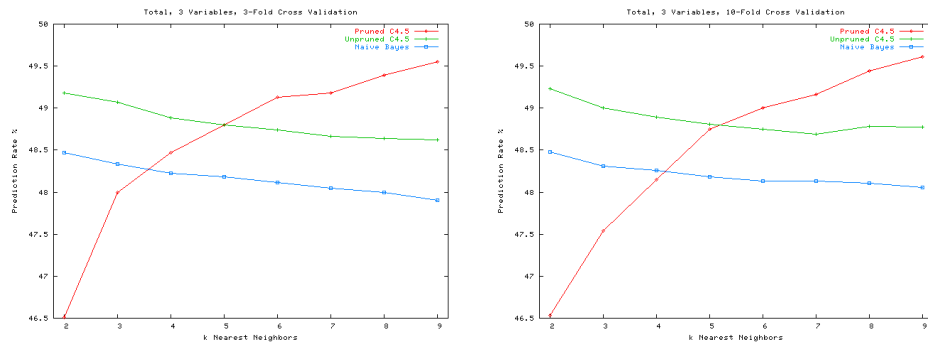


FIG. 3.4 – Cas de base. Les instances sont constituées de trois variables.

L’utilisation de la médiane en tant que mesure du taux de prédiction des différents groupes d’ensembles d’apprentissage permet d’avoir une meilleure idée de la distribution des points au sein de chaque groupe. Pour cela nous utilisons des graphes représentant la dispersion des points de chaque groupe autour de leur médianes respectives à l’aide *des intervalles interquartiles*¹. Sur la figure 3.5 nous

¹Etant donné un ensemble de données, nous le divisons en deux groupes de points (situées des deux côtés de la médiane) de tailles égales lorsque le nombre d’éléments est pair, ou en deux groupes de points y compris la médiane lorsque le nombre d’éléments est impair. Nous calculons ensuite les médianes de chacun des deux groupes (bas et haut) que nous appelons Q_1 et Q_3 respectivement. L’intervale interquartile est alors défini par l’expression $IIQ \equiv Q_3 - Q_1$.

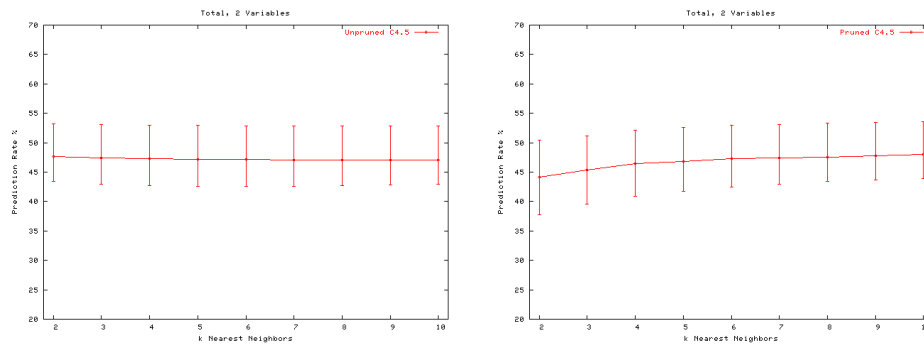


FIG. 3.5 – IIQ de C4.5 avec et sans élagage. Instances de deux variables.

présentons à titre d'exemple les intervalles interquartiles des deux algorithmes C4.5 présentés ci-haut dans le cas de règles de trois variables. Nous constatons que même si les données sont distribuées plus ou moins symétriquement par rapport à la médiane (qui est donc proche de la moyenne), la dispersion des points atteint en moyenne 10% ce qui est assez important compte tenu de l'écart relativement petit entre les médianes de différents algorithmes (<5%). De l'autre côté un résultat pareil est tout à fait logique vu l'hétérogénéité des données de départ et peut dès lors être amélioré par une meilleure sélection des données d'apprentissage.

3.3.2 Découpe en groupes

Au vu des résultats encourageant obtenus pour l'ensemble de toutes les données mises à notre disposition, l'étape logique suivante est de procéder à l'analyse des performances des mêmes algorithmes et configuration d'expérimentation que précédemment, mais sur des sous-ensembles d'apprentissage bien sélectionnés au préalable. Pour ce faire nous allons appliquer notre méthode des k voisins pour chacun des comptes utilisateurs comme dans le cas général mais cette fois-ci nous essayerons de trouver les utilisateurs ayant des listes de voisins semblables afin de les classer dans de mêmes groupes restreints. Un exemple bien caractéristique de cette approche est présenté dans le tableau 3.10. Nous voyons ici qu'en parcourant

```

machine@linux:~/bin> more logs/users-knn/biofor01-knn
biofor13 biofor11 biofor10 biofor09 biofor06
biofor01 biofor08 biofor07 biofor04 biofor02

machine@linux:~/bin> more logs/users-knn/biofor02-knn
biofor02 biofor04 biofor13 biofor11 biofor10
biofor09 biofor06 biofor01 biofor08 biofor07

```

TAB. 3.10 – Exemples de listes de k plus proches voisins

les différentes listes appartenant aux utilisateurs `bioforXX` on retrouve chaque fois

(à quelques exceptions près) les mêmes comptes qui finalement peuvent être regroupés dans un groupe à part appelé `biofor`. Les autres groupes (plus ou moins isolés) obtenus de la même manière et que nous avons testé sont : `bioinf`, `umhulb`, `ipmb` et `dea`. Sur la figure 3.6 nous présentons les tailles de ces groupes¹. Chaque colonne représente le nombre d'ensembles de k plus proches voisins ($\forall k$) sur lesquelles nous avons effectué une validation croisée par chacun des algorithmes. Remarquons qu'il n'est pas étonnant de voir qu'il s'agit des groupes collectifs utilisés probablement par des novices² suivant un schéma donné dans leurs actions.

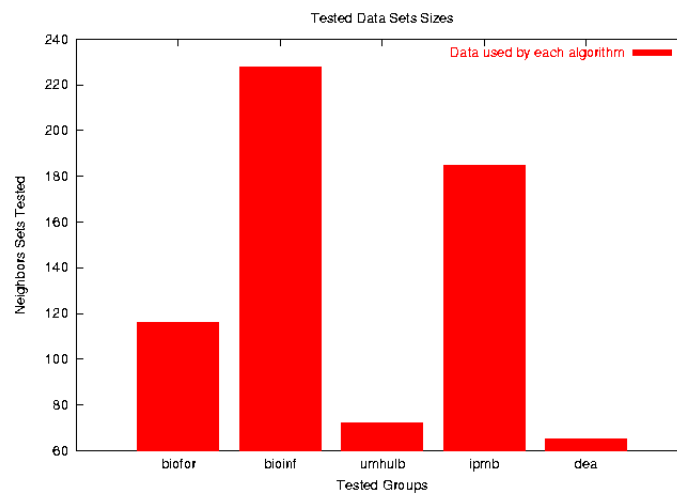


FIG. 3.6 – Tailles des groupes isolées.

Les expérimentations avec les groupes diffèrent de celles des cas de base sur plusieurs points ; ici nous fixons le nombre de variables des règles constituant les workflows à deux (simple choix), nous analysons les groupes avec et sans les doublons dans leurs workflows et, encore plus important, nous présentons comme résultats les médianes de l'ensemble de tous les points d'un certain voisinage (les différents k) pour toutes les valeurs k -Fold de la méthode de validation confondues (nous effectuons la validation R avec un k égal à 3 et 10, mais aussi à $n - loo$)³.

Les résultats sont présentés sur la figure 3.7. La première chose à remarquer c'est le taux de prédiction qu'atteignent les différents groupes ; dans le meilleur des cas nous obtenons un taux d'environ 77%. Il s'agit du taux de C4.5 élagué appliqué sur le groupe `biofor` sans éliminer les doublons. Mais même le pire résultat lorsque les doublons ne sont pas éliminés tourne autour des 60% pour nos groupes. En ce qui concerne la dynamique des algorithmes, elle est de manière générale la même pour les différents groupes ; C4.5 avec élagage bat le même algorithme sans élagage

¹Dans le cas de base nous avons utilisé 2141 ensembles.

²Cette remarque est une pure spéculation. Nous n'avons aucune information sur les comptes.

³Plus haut ces résultats étaient présentés sur des graphes distincts.

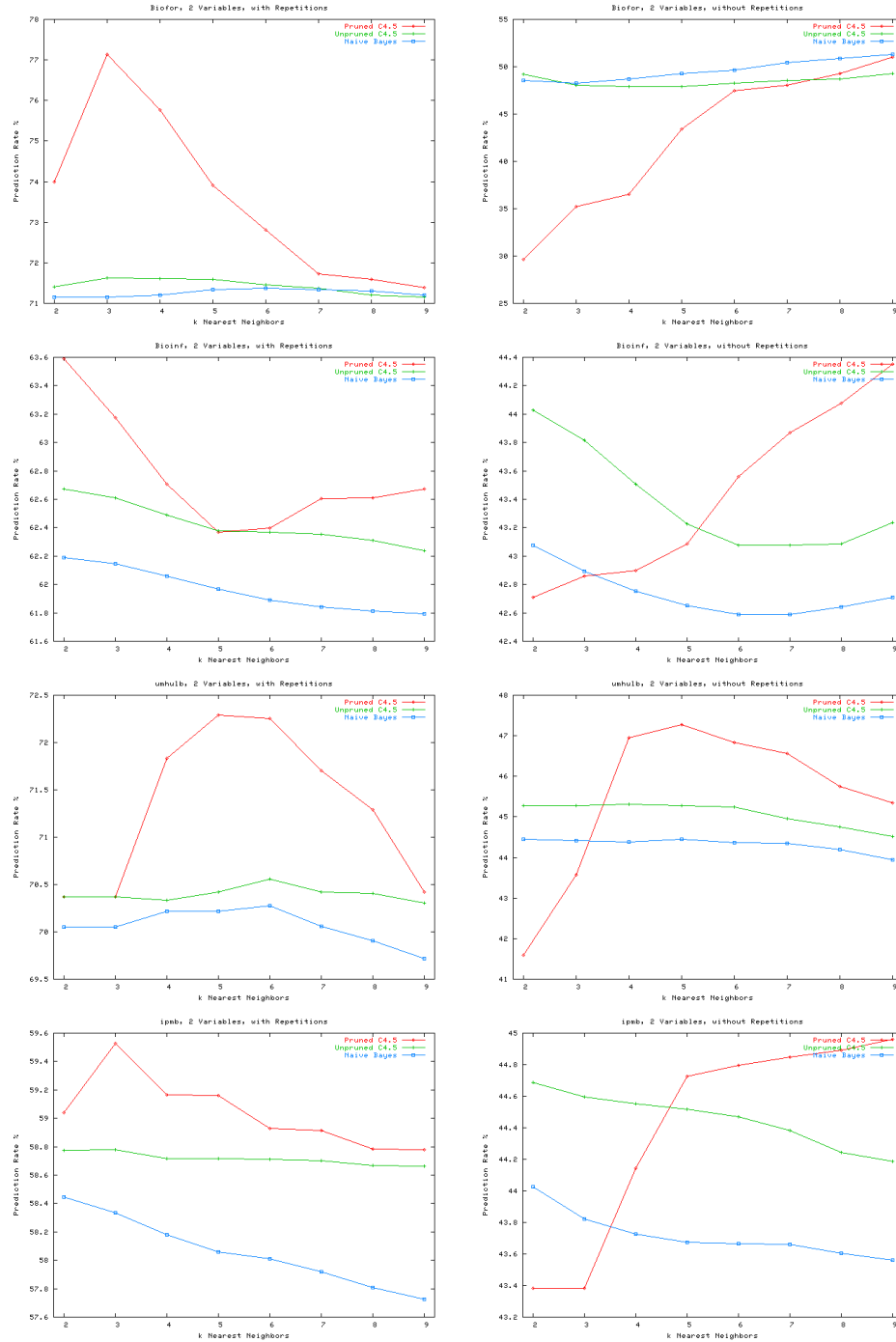


FIG. 3.7 – Taux de prédiction pour différents groupes avec/sans doublons.

et celui de Bayes sur l'ensemble de tous les groupes lorsque les doublons ne sont pas éliminés (les répétitions lui permettent de bien ajuster son arbre de décision surtout pour de petits ensembles d'apprentissage), et arrive à rattraper très vite les deux autres algorithmes lorsque les doublons sont éliminés (le même problème de généralisation excessive que précédemment). Il est également intéressant de remarquer que le taux de prédiction de tous les algorithmes testés descend parfois avec l'augmentation de la taille de l'ensemble d'apprentissage. Ceci est une belle illustration de la nécessité de bien choisir les workflows similaires. En effet, dans notre cas, nous ne nous basons pas sur le contenu des workflows (contenu logique), et donc il se peut que les workflows rajoutés à partir d'un certain moment n'apportent plus rien d'intéressant à l'apprentissage, mais bien au contraire que leur contenu perturbe le modèle du classifieur. Dans ce cas les performances des algorithmes diminuent au fur et à mesure que de nouveaux cas sont rajoutés.

En ce qui concerne le NBC, nous remarquons qu'il a de manière générale des performances bien pires que celles des deux autres algorithmes, avec la seule exception notable du cas du groupe `biofor` sans les doublons. Il s'agit là probablement d'une conséquence directe de l'élimination des doublons. En effet, en supprimant les cas répétitifs dans chaque workflow (il y en a beaucoup à en juger par les performances du même cas sans élimination des doublons), nous nous retrouvons dans une situation où les algorithmes construisant des arbres n'ont pas la possibilité d'affiner leurs modèles d'apprentissage (visiblement ceci influence beaucoup moins le NBC). Pour les autres groupes cette tendance n'est pas aussi accentuée.

En revenant sur la question de la dispersion des points autour des médianes, nous présentons à titre d'exemple sur la figure 3.8 les intervalles interquartiles des résultats obtenus dans le cas de `biofor` sans élimination des doublons. Nous

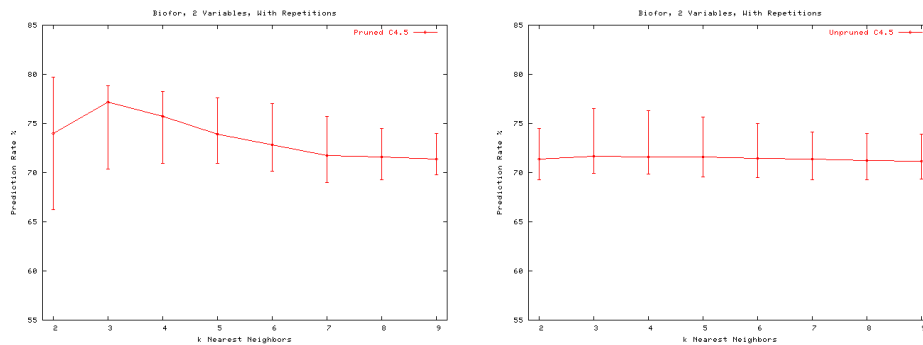


FIG. 3.8 – IIQ dans le cas de `biofor` sans élimination des doublons.

voyons qu'en moyenne la dispersion est plus petite que dans le cas général de base ($\pm 5\%$) ce qui correspond bien à nos attentes. De l'autre côté nous remarquons que la valeur maximale de la dispersion a quasiment doublé pour de petites valeurs de k . Ceci peut s'expliquer par la petite taille du groupe car, en effet, moins il y a de points, et plus l'intervalle interquartile est sensible aux extrêmes; il suffit donc de quelques "mauvais" points (dont la probabilité d'apparaître est plus élevée dans le cas de petits ensembles d'apprentissage) pour influencer toute l'image finale.

3.3.3 Approche *IKDB*

L'étape suivante de nos expérimentations consiste à étudier l'influence de la méthode de validation sur les taux de prédiction que peuvent atteindre nos algorithmes. Jusqu'à présent nous avons utilisé la validation croisée au niveau des règles. Et même si les résultats obtenus par cette méthode conviennent parfaitement à nos besoins d'évaluation des performances potentielles de prédiction des workflows (moyennant l'ensemble des hypothèses évoquées dans la section 3.2.1.2), il peut être assez intéressant de se placer dans un contexte plus proche de la future réalité du terrain et d'adopter une approche semblable à celle décrite pour la prédiction sur base d'une *IKDB*. Nous avons déjà mis en évidence "le problème de la moyenne" qui apparaît lorsque, lors de la validation croisée à chaque étape, les prédictions sont effectuées sur des blocs de tests (et même d'apprentissage) de tailles différentes. En effet, il est alors impossible de calculer correctement (sans fausser le résultat) une valeur exacte de prédiction pour le groupe sur lequel la validation vient d'être effectuée. C'est justement pour vérifier ce qu'il en est réellement de cette hypothèse avancée que nous procédons aux expériences décrites plus loin.

Pour effectuer ces expérimentations nous avons choisi d'utiliser le groupe sur lequel nous avons obtenu les meilleurs performances dans le cas de la validation *R*, à savoir le groupe *biofor* (sans éliminer les doublons). Nous avons comme

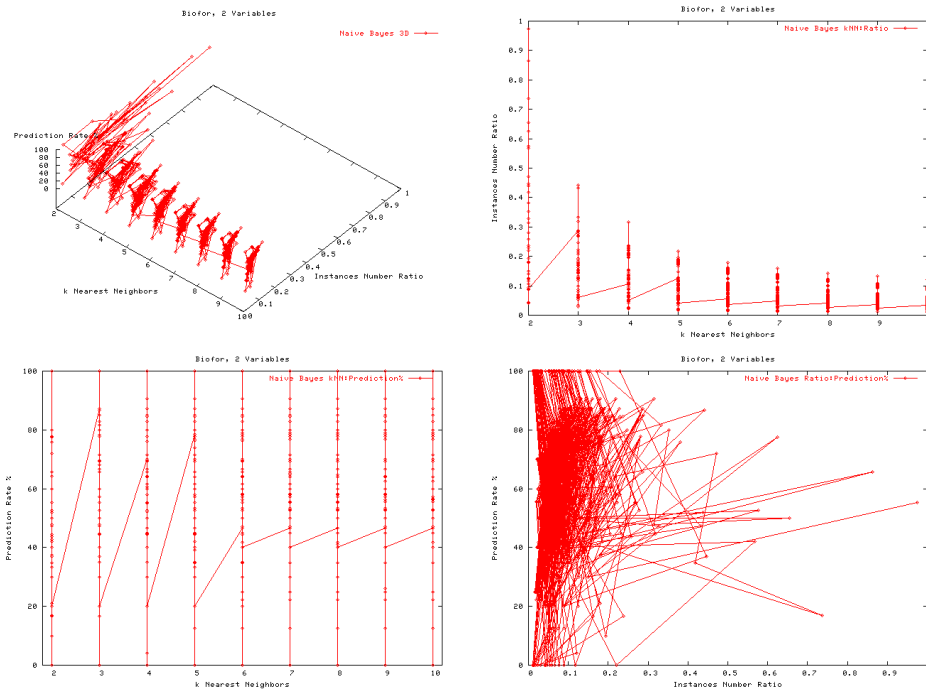


FIG. 3.9 – Validation croisée au niveau des workflows appliquée au groupe *biofor*.

précédemment effectué l'évaluation du taux de prédiction en fonction du nombre k de voisins. Pour cela nous prenons dans chaque ensemble de k voisins à tour de

rôle chacun des workflows de l'ensemble et nous l'évaluons par rapport au reste de cet ensemble (sorte de validation croisée baptisée "validation *WF*"). Le résultat est un ensemble de points indiquant chacun le taux de prédiction calculé et la valeur de k correspondante, mais également les deux valeurs indiquant les tailles des ensembles de test et d'apprentissage. En prenant le rapport entre les deux tailles nous obtenons pour le groupe testé les résultats indiqués sur la figure 3.9.

En analysant les différentes vues de l'image obtenue nous constatons comme prédit qu'il est presque impossible de retenir des valeurs exactes quelconques en ce qui concerne le taux de prédiction pour les différents ensembles de voisins testés. En effet la relation présentée sur la vue de gauche montre que pour chaque k nous obtenons une dispersion des points allant de 0% jusqu'à 100%. La vue de droite nous permet de voir que ces points extrêmes correspondent le plus souvent aux cas où les ensembles de tests sont beaucoup plus petits que les ensembles d'apprentissage (rapport $< \frac{1}{10}$), et en regardant la vue d'en haut nous remarquons que même dans le peu de cas (très rarement) où le rapport est supérieur à $\frac{1}{10}$ il s'agit des ensembles contenant peu de règles (petits k). Une réflexion analogue nous permet d'exclure les points les moins représentatifs (les plus distants des concentrations principales) de chaque ensemble de points correspondants aux différents k . Nous obtenons ainsi pour chaque k une fourchette des prédictions contenant la majorité des points, dont les bornes inférieure et supérieure (et donc la qualité des prédictions de l'ensemble des points) augmentent avec k . Et même si l'écart entre les deux bornes reste beaucoup trop flou pour être quantifié¹, il nous donne tout de même une indication quant à l'efficacité de la méthode de prédiction proposée.

3.4 Conclusion

Avant de procéder à l'implémentation technique du Workflow Intelligent, il est nécessaire de s'assurer que l'exploitation du caractère répétitif de l'environnement du simple Workflow (la prédiction des actions) permet d'améliorer les performances de son utilisation (fournit un taux de prédiction convenable). Pour cela nous avons étudié les performances des algorithmes d'apprentissage à la base du Workflow Intelligent sur l'exemple des données réelles que sont les *logs* des utilisateurs Emboss.

Nous avons commencé l'étude par le pré-traitement des *logs*. Au cours de cette procédure plusieurs hypothèses simplificatrices ont été mises en évidence ainsi que leur influence sur les performances qu'elles entraînent. C'est également à cette étape-là de l'étude que nous avons introduit le format de représentation des données sous forme de workflows et les algorithmes d'apprentissage Weka capables de gérer ce format. La suite de l'étude consistait en l'estimation des bornes inférieures des performances des algorithmes C4.5 (avec/sans élagage) et NBC étant donnée une certaine configuration de test. La première configuration testée a été celle qui nous place théoriquement dans les pires conditions de prédiction. Il s'agit de la configuration la plus générale, à savoir celle qui reprend toutes les données (très hétérogènes) disponibles et qui prévoit une validation croisée ne pre-

¹Il est en tous cas beaucoup plus grand que les intervalles interquartiles observés plus haut

nant en compte ni le caractère temporelle propre aux workflows, ni les autres liens logiques existant entre les données testées. Le pire résultat de cette approche était de $\pm 40\%$ (la médiane de l'ensemble des résultats) et ne nous a pas permis de donner la préférence à un algorithme quelconque parmi ceux testés. Le deuxième pas a été de “nettoyer” l'ensemble d'apprentissage grâce à la construction de groupes de comptes utilisateurs similaires par une approximation de l'algorithme *kNN*. Les résultats dans ce cas étaient nettement meilleurs (ils atteignaient dans le meilleur des cas $\pm 75\%$) et l'algorithme C4.5 avec élagage arrivait à battre les deux autres dans la grande majorité des cas. La dernière configuration avait pour but d'analyser les performances de prédiction en respectant l'intégrité de chaque workflow.

En conclusion de nos analyses nous pouvons affirmer que le Workflow Intelligent pourrait effectivement être une amélioration utile du Workflow de base, car un taux de prédiction de $\pm 65\%$ ¹ (au moins deux actions correctes sur trois prédites) est un objectif tout à fait réaliste compte tenu des résultats obtenus sur les données de test (de faible qualité) Emboss. Néanmoins soulignons que pour atteindre cet objectif, plusieurs conditions telles que par exemple la construction efficace des groupes de workflows similaires (*IKDB*), doivent impérativement être remplies.

¹Ce taux peut être qualifié de convenable.

Conclusion

L'objectif de ce mémoire était la conception du Workflow Intelligent. Il s'agissait en effet d'améliorer le Workflow de base permettant aux utilisateurs de mailler automatiquement des services bio-informatiques regroupés au sein de la fédération BIGRE. L'approche que nous avons proposée pour cela consiste à analyser les expérimentations effectuées précédemment par d'autres utilisateurs (workflows experts) pour en déduire les actions possibles de l'utilisateur donné lors de la réalisation de son workflow. Les principes théoriques derrière cette déduction des actions de l'utilisateur s'inscrivent dans le cadre de la théorie de l'apprentissage.

Puisque l'apprentissage est au coeur de notre travail, nous avons commencé par proposer un état de l'art complet de la théorie de l'apprentissage inductif de concept en présentant le fonctionnement et l'analyse des performances des principaux algorithmes de ce domaine. Finalement pour nos besoins spécifiques de prédiction des actions de l'utilisateur nous avons retenu les deux variantes de l'algorithme C4.5 (sans et avec élagage) et l'algorithme probabiliste NBC. Les deux premiers, qui servent généralement d'algorithmes de référence, ont l'avantage d'avoir un faible coût en temps de calcul et d'être assez résistants au bruit dans les données d'apprentissage. Le choix du deuxième algorithme a été dicté par son extrême simplicité et par sa faible probabilité d'erreur. Tous ces algorithmes sont implémentés et disponibles en Java dans le *workbench* WEKA¹. Le CBR est lui aussi retenu, mais pour une application différente de celle des trois autres algorithmes déjà énoncés. En effet, cet algorithme convient parfaitement à la tâche de construction des ensembles de workflows experts - les *IKDB* - sur base des définitions des workflows en cours de réalisation. Il a une complexité d'apprentissage quasi nulle et une complexité de prédiction linéaire en nombre d'instances.

L'étape suivante dans notre réflexion a été celle de mettre au point l'IHM du Workflow Intelligent. Le choix finalement retenu consiste à rajouter au Workflow de base une barre de prévision capable d'afficher les quelques prochaines actions les plus probables de l'utilisateur. Une telle approche a l'avantage d'être en même temps hautement intelligible et complètement non obstructive au travail de l'utilisateur. Le mécanisme de fonctionnement de la barre des prévisions est constitué par un système d'interaction entre les trois agents de la couche IA du Workflow situés sur les Clients et l'interface cliente du Médiateur. Le processus de prédiction se déroule en cascade : l'interface utilisateur de la KDB renvoie l'ensemble d'ap-

¹Notons au passage que d'autres algorithmes étaient également envisageables, tel par exemple RISE, et qu'il est toujours intéressant de tester leurs performances sur des jeux de données réels.

prentissage au Client suite à une requête concernant un profil précis, la base des connaissances locale est alors construite par un des agents du Client qui passe ensuite la main à l'agent chargé de produire les futures prédictions sur base de l'historique et de la *IKDB* fraîchement construite, qui lui à son tour passe la main au dernier agent chargé d'afficher le résultat prédit dans la barre des prévisions.

Dans notre travail nous partons de l'hypothèse qui consiste à supposer qu'il existe une (forte) corrélation entre les différents workflows réalisés sur des sujets semblables. Même si intuitivement l'approche semble valable, il est important de vérifier ce qu'il en est en réalité. La dernière partie de notre travail consistait donc à mesurer expérimentalement les performances des algorithmes d'apprentissage retenus sur des jeux de données réels - les *logs* d'utilisation d'Emboss. Nous avons ainsi réalisé plusieurs configurations de tests mettant chacune en évidence l'influence des différents paramètres de réalisation sur les performances de prédiction. Parmi les résultats importants obtenus nous pouvons citer les bonnes performances de l'algorithme C4.5 sans élagage (de manière générale les pires performances étaient de l'ordre de 30% alors que les meilleurs atteignaient presque 80%), l'influence notable du choix du nombre d'attributs de l'ensemble d'apprentissage (*IKDB*) sur les performances et l'importance *capitale* d'une "bonne" construction de la *IKDB*.

Le travail présenté dans ce mémoire n'est pour l'instant qu'à ses débuts théoriques et sa réalisation devra être poursuivie ultérieurement. En guise de rappel, soulignons les principales voies de recherche qui se présentent pour la suite. Comme indiqué dans le Cahier des charges, les futures tâches se divisent en deux catégories : la gestion de la KDB du côté Médiateur et l'implémentation du mécanisme de prédiction du côté Client. Si les tâches de la première catégorie ne sont pas encore clairement définies, celles de la deuxième sont au contraire bien établies à présent. La réalisation Java des agents de la couche intelligente du Workflow devrait être considérablement facilitée par la disponibilité des codes sources des algorithmes d'apprentissage WEKA, l'étape suivante étant de les faire communiquer entre eux via notamment les différentes structures de données nécessaires. Une fois les deux côtés Médiateur et Client réalisés, il sera encore nécessaire d'attendre que le Workflow de base soit opérationnel pour procéder aux tests de performances finales avant de mettre un terme définitif à la réalisation du Workflow Intelligent.

Annexe A

```
# L'objectif de ce script est de reconstituer les workflows des
# utilisateurs ayant un compte Emboss à partir du fichier des
# logs. Il commence par enregistrer la suite des actions de
# l'utilisateur en question dans le fichier logs/users/USER
# correspondant en indiquant l'action effectuée et la date.
# Ensuite des fichiers pseudo-BPML contenant les descriptions des
# différents workflows sont créés dans logs/users-seq/USER-seq
# sur base de la liste ainsi constituée.

#!/usr/bin/perl

use Shell;

# Declaration des variables globales
$nombreVar=2; print "Creating /logs...\n";
$files_dir_logs="logs/"; $files_dir_users="logs/users/";
$files_dir_useq="logs/users-seq/"; $files_dir_uwf="logs/users-wf/";
mkdir($files_dir_logs); mkdir($files_dir_users);
mkdir($files_dir_useq); mkdir($files_dir_uwf);
%month_list=('Jan','01','Feb','02','Mar','03','Apr','04','May','05','Jun','06',
            'Jul','07','Aug','08','Sep','09','Oct','10','Nov','11','Dec','12');

# Parcours du fichier des logs. Lorsque les lignes rencontrées ne présentent
# pas d'interet particulier (p.e. les actions administratives) nous ne les
# prenons pas en considération. Les actions sont mises dans une hash.

print "Creating /logs/users...\n";
open (IN,$ARGV[0]);
while (<IN>){
    ($action,$user,$day,$month,$date,$time,$year)=
        ($_~/(.+)\t(.+)\t(.+)\s(.+)\s(.+)\s(.+)\s(.+)/);
    if ($user ne 'gbottu' && $user ne 'embadmin' && $action ne 'wosname'
        && $action ne 'showdb' && $action ne 'infoseq' && $action ne 'showseq'){
        $DD=$date; $MM=$month_list{$month};
        $action_time="$action \t $year".'/'. "$MM".'/'. "$DD".'.':'. "$time\n";
        my @actions_time;
        if (!$user_act{$user}){
```

```

        if ($user ne '') {
            my @actions_time; push @actions_time,$action_time;
            $user_act{$user}=\@actions_time;}
        }else{
            my @actions_time; @actions_time=@{$user_act{$user}};
            push @actions_time,$action_time;
            $user_act{$user}=\@actions_time;
        }
    }
} close IN;

# Boucle d'impression des actions dans les fichiers extérieurs.

while (my ($user,$reftab) = each %user_act){
    open(fileOUT, ">$files_dir_users"."$user");
    my @suite=@$reftab;
    print fileOUT join /, /,@suite; print fileOUT "\n";
    close(fileOUT);
}

$files_list=ls($files_dir_users);
@user_list=split(/\s/,$files_list);
print "Creating /logs/users-seq...\n";
for ($fileIndex=0; $fileIndex <= $#user_list; $fileIndex++) {
    open (IN,"$files_dir_users"."@user_list[$fileIndex]");
    open(fileOUT, ">>$files_dir_useq"."@user_list[$fileIndex]".-seq");
    $dateComp=""; $flag=0; print fileOUT "\n";

# Parcours des actions de chaque utilisateur et écriture des workflows
# en output. C'est ici qu'on doit spécifier la manière de découper en WF

while (<IN>){
    ($action,$date,$heure)=($_=~/(.+)\t(.{11})(.{9})/);
    $date =~ s/^\s+//; $date =~ s/\s+$//;
    $action_time="$action\t $heure\n";
    #print "$action_time";
    my @actions_time;
    if ($dateComp ne $date) {
        $dateComp=$date;
        # Ajout des balises lorsque la date change: 1 WF = 1 jour
        if ($flag!=0) {print fileOUT " <\\contents>\n<\\workflow>\n\n";}
        $flag=1;
        if ($date ne "")
            {print fileOUT "<workflow>\n <date> $date
                <\\date>\n <contents=linear> \n";}
    }

    print fileOUT $action_time;
} close (fileOUT);
close IN;

```

```

}

# Exemple illustratif d'appel de l'algorithme C4.5 élagué.
# On effectue une validation croisée sur un groupe de workflows
# contenus dans les comptes des utilisateurs biofor01 - biofor12.
# Les workflows sont d'abord transformés en ensembles d'apprenti-
# ssages. Ensuite les données sont passées aux algorithmes java,
# dont l'output est parsé - on extrait les données interessantes.

#!/usr/bin/perl

use Shell;

$files_dir_users="logs/users/";
#$files_list=ls($files_dir_users);
# Declaration du groupe
$users_list="biofor01 biofor02 biofor03 biofor04 biofor05 biofor06
             biofor07 biofor08 biofor09 biofor10 biofor11 biofor12";
&group_learn($users_list);
&make_lKDB("biofor10",$files_list);
#print &split_WF("biofor10-lkdb");
&user_learn("biofor10",$users_list);
&c45_pruned_learn("logs/users-lkdb/biofor10-lkdb.arff",
                  "logs/tmp/biofor10-0.arff");

# Procédure d'apprentissage sur un groupe donné. Son role est
# de passer en boucle chaque paire <compte - autres des comptes>

sub group_learn {
my(@args) = @_; $the_group=@args[0]; $user=0;
@users=split(/\s/, $the_group); $users_list="";
while ($user<=#users) {
    $the_user=@users[$user];
    for ($i=0;$i<=#users;$i++){
        if ($i!=$user) {
            $users_list=$users_list.@users[$i]." ";
        }
    }
    print @users[$user].":".$users_list."\n";
    &user_learn(@users[$user], $users_list);
    $user++; $users_list="";
}
}

# Accepte un ensemble de workflows à tester de the_user et les verifie
# par rapport à la base de données constituée des WF des users de la
# liste users_list (il s'agit de l'approche lKDB).

sub user_learn {

```

```

my(@args) = @_; $the_user=@args[0]; $users_list=@args[1];
&pre_treat("$the_user $users_list"); # $users_list");
# Creation d'un ensemble d'apprentissage .arff pour the_user
&make_lkDB($the_user,$the_user);
# Éventuellement les doublons sont éliminés
#system("uniq logs/users-lkdb/$the_user-lkdb.arff |
#                               cat > logs/tmp/$the_user-tmp.arff");
# Les WF de l'ensemble d'apprentissage sont séparés
$wfNum=&split_WF($the_user);
print "  Users list (1): ".$users_list."\n";
# Creation d'un ensemble d'apprentissage .arff pour les users_list
&make_lkDB($the_user,$users_list);
rm("header"); print "\n"; # $i="tmp";
# Pour chaque WF l'algorithme d'apprentissage est invoqué.
# Les autres WF sont rajoutés à l'ensemble d'apprentissage.
for ($i=0; $i<=$wfNum; $i++) {
    &c45_pruned_learn("logs/users-lkdb/$the_user-lkdb.arff",
                    "logs/tmp/" . @args[0] . "-" . $i . ".arff");
}
}

# Procédure d'invocation de C4.5 élagué

sub c45_pruned_learn {
my(@args) = @_; $learn_set=@args[0]; $test_set=@args[1];
system("cd Weka; java weka.classifiers.trees.J48 -t ../$learn_set -T
    ../$test_set -i | grep -E 'Total Number of Instances|Time
                                taken|Correctly' > ../stats");

open(INFO, "stats"); @array=<INFO>; close (INFO);
#print "": $learn_set | $test_set\n";
$line=@array[0]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s/, $line); $t_c=@fields[5];
$line=@array[1]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s/, $line); $t_p=@fields[8];
$line=@array[3]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s+/, $line); $N=@fields[4];
$line=@array[4]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s+/, $line); $err=@fields[4];
$line=@array[5]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s+/, $line); $n=@fields[4];
#print @array;
print "$test_set | $t_c | $N | $t_p | $n | $err\n";
#print "t_c: $t_c t_p: $t_p N: $N %_err: $err n: $n\n";
}

# Prend un fichier .arff en entree et produit des petits fichiers
# .arff contenant les workflows du grand fichier dans le repertoire
# logs/tmp/ - les noms sont username-lkdb-X.arff

sub split_WF {

```

```

my(@args) = @_; $filename="logs/users-lkdb/" . @args[0] . "-lkdb.arff";
system("uniq $filename > uniq.arff; cat uniq.arff > $filename");
rm("uniq.arff"); open(INFO, $filename); @array=<INFO>; close (INFO);
print " Splitting ".$filename; print "...\\n";

$i=0; @header=(); $line=@array[$i]; $line =~ s/^\\s+//; $line =~ s/\\s+$//;
push(@header,$line); push(@header,"\\n");

while ($line ne "\\@data"){
    $i++; $line=@array[$i]; $line =~ s/^\\s+//; $line =~ s/\\s+$//;
    if ($line ne "") { push(@header,$line); push(@header,"\\n"); }
} $i++;

$wfNum=0; mkdir("logs/tmp/");
#open(fileOUT, ">logs/tmp/" . @args[0] . "-" . $wfNum . ".arff");
open(fileOUT, ">logs/tmp/tmp");
print " > (via tmp) logs/tmp/" . @args[0] . "-" . $wfNum . ".arff\\n";
print fileOUT "@header"; # $i++;
$temp = @array[$i]; print fileOUT $temp; $i++;
$temp = @array[$i]; $fold=0;

while ($i< $#array) {

    #if ($temp =~ /%/ ) {
    if ($fold==31) {
        close(fileOUT);
        system("uniq logs/tmp/tmp > logs/tmp/@args[0]-$wfNum.arff");
        $wfNum++; $fold=0;
        #open(fileOUT, ">logs/tmp/" . @args[0] . "-" . $wfNum . ".arff");
        open(fileOUT, ">logs/tmp/tmp");
        print " > (via tmp) logs/tmp/" . @args[0] . "-" . $wfNum . ".arff\\n";
        print fileOUT "@header"; # $i++;
        print fileOUT $temp;
    } else {print fileOUT $temp; }
    $i++; $temp = @array[$i];
    if ($temp =~ /%/ ) {} else {$fold++;}

}

close(fileOUT);
system("uniq logs/tmp/tmp > logs/tmp/@args[0]-$wfNum.arff");

return $wfNum;
}

# Crée le fichier .arff contenant les EA des users de la file_list
# dans logs/users-lkdb/ pour un utilisateur donné. Le fichier
# logs/users-seq doit exister (on lit les sequences de leurs actions).

```

```

sub make_lKDB {
my(@args) = @_; $the_user=@args[0]; $files_list=@args[1];

$nombreVar=2; $files_dir_useq="logs/users-seq/";
$files_dir_lkdb="logs/users-lkdb/"; mkdir($files_dir_lkdb);

@user_list=split(/\s/,$files_list);
print "  Creating the logs/users-lkdb/$the_user-lkdb.arff file.\n";
print "  Users list: @user_list\n";
$nombreVar=$nombreVar+1;

print "  ... adding the users actions; \n";
open(fileOUT, ">tmp"); @instances=();
for ($fileIndex=0; $fileIndex <= $#user_list; $fileIndex++) {
  open(INFO, "$files_dir_useq".@user_list[$fileIndex]."-seq");
  @array=<INFO>; close (INFO);
#print "... adding the ".@user_list[$fileIndex]." user actions; \n";
$i=0;
while ($i<#$array) {
  $line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
  @block=();
  if ($line eq "<workflow>") {
    $line = @array[$i+1]; $line =~ s/^\s+//; $line =~ s/\s+$//;
    push(@block,"% <user> ".@user_list[$fileIndex]." <\user> $line\n");
    $i=$i+3; @wf=();
    $line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
    while ($line ne "<\contents>"){
      push(@wf, "@array[$i]");
      $i++; $line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
    }
    for ($j=0; $j<=(#$wf + 1 - $nombreVar);$j++) {
      push(@block,"");
      for ($l=0; $l<$nombreVar; $l++) {
        $temp = @wf[$j+$l]; @dummy=split(/\t/, $temp); $temp=@dummy[0];
        push(@instances, $temp."\n");
        if ($l!=( $nombreVar-1)) {$temp=$temp.", "}
        push(@block,$temp);
      } push(@block,"\n");
    } #push(@block,"<\contents>\n<\workflow>\n\n");

    if ($#block > 0) {print fileOUT "@block"."\n";}
  }
  $i++;
}
}

close(fileOUT); #empty=""; @uniq=();
#@instances=sort(@instances);
#%seen = (); foreach $item (@instances)
#   {push(@uniq, $item) unless $seen{$item}++;}

```

```

#$inst = $empty; #print @uniq;
#for ($i=0; $i <= $#uniq; $i++)
# { $inst = $inst."@uniq[$i]"; $inst =~ s/^\s+//;
#   $inst =~ s/\s+$//; $inst = $inst." "}
open(INFO, "header"); @header=<INFO>; close (INFO);
open(INFO, "tmp"); @array=<INFO>; close (INFO);
open(fileOUT, ">$files_dir_lkdb".$the_user."-lkdb.arff");
print fileOUT "@header";
print fileOUT "@array";
close(fileOUT);

return 1;

}

# La procédure sert à créer une liste d'actions (chaque action est
# reprise une seule fois). Celle-ci est indispensable à la constru-
# ction du fichier .arff acceptable par les algorithmes weka.

sub pre_treat {
my(@args) = @_; $files_list=@args[0];
@instances=(); @user_list=split(/\s/, $files_list);
print " Analysing logs/users-seq/*-seq files
      for ".@user_list[$fileIndex]."-seq"." file... \n";
for ($fileIndex=0; $fileIndex <= $#user_list; $fileIndex++) {
open(INFO, "logs/users-seq/" .@user_list[$fileIndex]."-seq");
@array=<INFO>; close (INFO);
#print "Analysing logs/users-seq/"
      @user_list[$fileIndex]."-seq"." file... \n";

$i=0;
while ($i< $#array) {
$line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
if ($line eq "<workflow>") {
$i=$i+3;
$line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
while ($line ne "<\content>"){
if ($line ne "<\content>")
{@dummy=split(/\s/, $line);
$temp=@dummy[0]; push(@instances,$temp."\n"); }
$i++; $line = @array[$i]; $line =~ s/^\s+//; $line =~ s/\s+$//;
} #print "out\n";
}
}
}

}

#print @instances; print "\n\n";
@uniq=(); @instances=sort(@instances);
%seen = (); foreach $item (@instances)

```

```

        {push(@uniq, $item) unless $seen{$item}++;}
$inst = $empty; #print @uniq;
for ($i=0; $i <= $#uniq; $i++)
    { $inst = $inst."@uniq[$i]"; $inst =~ s/^\s+//;
      $inst =~ s/\s+$//; $inst = $inst." "}
#print "$inst\n";

open(fileOUT, ">header");

print fileOUT "% This is the learning dataset.\n \n";
print fileOUT "@relation EmbossLearningDataset \n \n";
print fileOUT "@attribute one {$inst} \n";
print fileOUT "@attribute two {$inst} \n";
print fileOUT "@attribute three {$inst} \n \n";
print fileOUT "@data \n";
close(fileOUT);

}

# Script de regroupement des comptes utilisateur. Parcourt la
# liste des utilisateurs en trouvant pour chacun d'entre eux
# des k plus proches voisins sur base de sa liste d'actions.
# Lorsque deux comptes ont plusieurs actions en commun, on aug-
# mente le compteur de 1 (initialement 0), sinon on decremente.

#!/usr/bin/perl

use Shell;

$files_list=ls("logs/users/");
@user_list=split(/\s/,$files_list);
mkdir("logs/users-knn/");
for ($fI=0; $fI<=#user_list; $fI++) {
    open(fileOUT, ">tmp");
    for ($fileIndex=0; $fileIndex <= $#user_list; $fileIndex++) {
        # Comparaison de chaque utilisateur contre le reste
        print fileOUT &sss(@user_list[$fI], @user_list[$fileIndex]);
        print fileOUT " @user_list[$fileIndex] \n";
    }
    # Le résultat obtenu est ordonné en fct du score
    system("sort -rn tmp > tmp1"); @block=();
    open(INFO, "tmp1"); @tmp1=<INFO>; close (INFO);
    $line=@tmp1[0]; $line =~ s/^\s+//; $line =~ s/\s+$//;
    @fields=split(/\s+/, $line); $n=@fields[0];
    # Les mauvais résultats peuvent eventuellement etre eliminé
    # pour n'en laisser par exemple un nombre précis de voisins
    if ($n>$numVar) {
        for ($ii=0; $ii<=9; $ii++) {

```

```
$line=@tmp1[$ii]; $line =~ s/^\s+//; $line =~ s/\s+$//;
@fields=split(/\s+/, $line); $n=@fields[0];
push(@block, $line."\\n");
}
if ($#block>=1) {
  open(fileOUT, ">logs/users-knn/".@user_list[$fI]."-knn");
  print fileOUT @block;
}
}
close(fileOUT);
}

# Procédure de comparaison des actions. Fonction "f()";
sub sss {
  my(@args) = @_;
  open(INFO, "logs/users-instances/".@args[0]."-instances");
  open(INFO, "logs/users-instances/".@args[1]."-instances");
  @b10a=<INFO>; close (INFO); @b01a=<INFO>; close (INFO);
  @b10=split(/\s/, @b10a[0]); @b01=split(/\s/, @b01a[0]);
  $k=0;
  for ($i=0; $i<=$#b10; $i++) {
    for ($j=0; $j<=$#b01; $j++) {
      if (@b10[$i] eq @b01[$j]) {$k++; }
    }
  }
  $f=$k-(@b10-$k)-(@b01-$k);
  return $f;
}
```

Annexe B

Nous présentons ci-dessous un exemple complet d'un ensemble d'apprentissage présenté dans le format `.arff`. Il s'agit de l'ensemble d'apprentissage constitué des treize comptes utilisateur `bioforXX`. Nous utilisons ensuite cet ensemble pour illustrer l'interface graphique du *workbench* WEKA. Nous donnons quatre *screenshots* de l'application reprenant les étapes principales de l'analyse des données à l'aide de WEKA. Les données sont d'abord introduites dans l'application, ensuite ayant choisi un algorithme d'apprentissage et une méthode de validation (nous utilisons la validation croisée $k = 10$) nous obtenons un *listing* complet des résultats, et obtenons même l'occasion de visualiser l'arbre de décision construit par le classifieur. Nous terminons la présentation par l'affichage du *listing* fourni par l'algorithme.

```
% This is the learning dataset.

@relation EmbossLearningDataset

@attribute one {blast blast2seq bscan dialign diffseq dotmatcher dottup
ehmmpfam eprimer3 fasta fuzzpro getorf makeblastdb matcher megamerger
needle plotorf ps_scan pscan seqmatchall sim_lav stretcher }
@attribute two {blast blast2seq bscan dialign diffseq dotmatcher dottup
ehmmpfam eprimer3 fasta fuzzpro getorf makeblastdb matcher megamerger
needle plotorf ps_scan pscan seqmatchall sim_lav stretcher }
@attribute thr {blast blast2seq bscan dialign diffseq dotmatcher dottup
ehmmpfam eprimer3 fasta fuzzpro getorf makeblastdb matcher megamerger
needle plotorf ps_scan pscan seqmatchall sim_lav stretcher }

@data
% <user> biofor01 <\user> <date> 2003/05/13 <\date>
  dotmatcher , dotmatcher , dotmatcher
  dotmatcher , dotmatcher , stretcher
  dotmatcher , stretcher , stretcher
  stretcher , stretcher , matcher
  stretcher , matcher , matcher
  matcher , matcher , sim_lav
  matcher , sim_lav , fasta
  sim_lav , fasta , fasta
  fasta , fasta , fasta
  fasta , fasta , blast
```

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Undo | Save...

Filter | Choose None | Apply

Current relation
 Relation: EmbossLearningDataset | Instances: 397 | Attributes: 3

Selected attribute
 Name: three | Type: Nominal | Missing: 0 (0%) | Distinct: 14 | Unique: 0 (0%)

Label	Count
blast	59
blast2seq	0
bscan	20
dialign	0
diffseq	0
dotmatcher	65
dotup	2
ehmpfam	24
eprimer3	35
fasta	32
fuzzpro	19
getorf	17

Attributes

No.	Name
1	one
2	two
3	three

Class: three (Nom) | Visualize All

Attribute	Count
blast	59
blast2seq	0
bscan	20
dialign	0
diffseq	0
dotmatcher	65
dotup	2
ehmpfam	24
eprimer3	35
fasta	32
fuzzpro	19
getorf	17

Status OK | Log | x 0

Weka Explorer | Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier: Choose **j48 -C 0.25 -M 2**

Test options

- Use training set
- Supplied test set
- Cross-validation Folds: **10**
- Percentage split: **66** %

More options...

(Nom) three

Start Stop

Result list (right-click for options)

19:35:24 - trees.j48

Classifier output

```

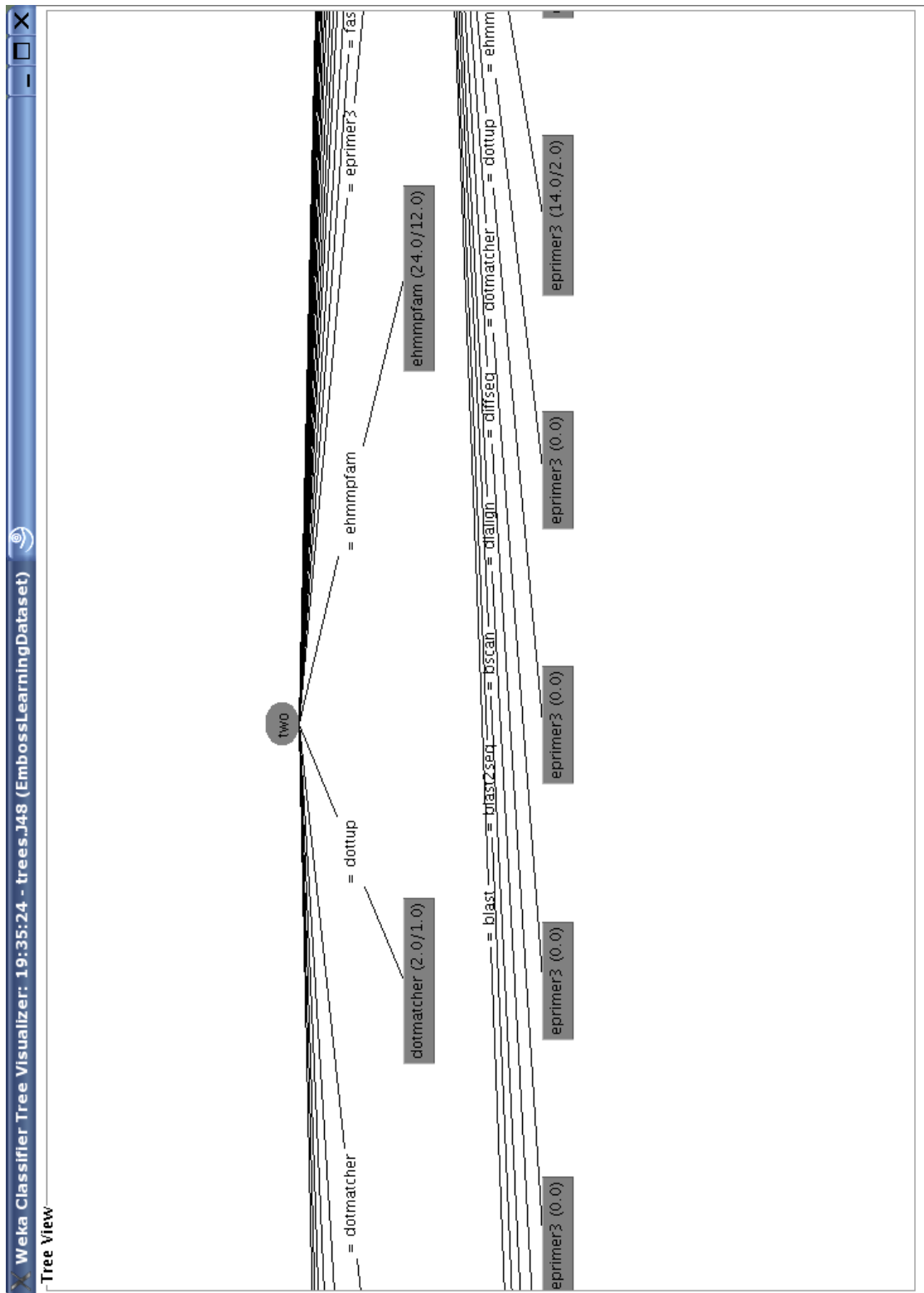
two = stretcher: stretcher (38.0/12.0)
Number of Leaves : 64
Size of the tree : 67

Time taken to build model: 0.09 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 263 66.2469 %
Incorrectly Classified Instances 134 33.7531 %
Kappa statistic 0.6253
Mean absolute error 0.0374
Root mean squared error 0.1421
Relative absolute error 45.3412 %
Root relative squared error 70.0959 %
Total Number of Instances 397

=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure Class
0.763 0.018 0.882 0.763 0.818 blast
0 0 0 0 0 blast2seq
0.05 0.024 0.1 0.05 0.067 bscan
0 0 0 0 0 dialign
0 0 0 0 0 diffseq
0.985 0.036 0.842 0.985 0.908 dotmatcher
0 0.003 0 0 0 dottup
0.667 0.059 0.421 0.667 0.516 ehmpfam
0.886 0.03 0.738 0.886 0.805 eprimer3
0.688 0.038 0.611 0.688 0.647 fasta
0.316 0.019 0.462 0.316 0.375 fuzzpro
0 0.018 0 0 0 getorf
    
```

Status: OK



```
fasta , blast , makeblastdb
blast , makeblastdb , makeblastdb
makeblastdb , makeblastdb , makeblastdb
makeblastdb , makeblastdb , blast

% <user> biofor01 <\user> <date> 2003/05/14 <\date>
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ps_scan
ehmpfam , ps_scan , bscan
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ehmpfam
ehmpfam , ehmpfam , ehmpfam
ehmpfam , ehmpfam , bscan
ehmpfam , bscan , bscan
bscan , bscan , ehmpfam
bscan , ehmpfam , fuzzpro
ehmpfam , fuzzpro , getorf
fuzzpro , getorf , getorf
getorf , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3

% <user> biofor01 <\user> <date> 2003/05/26 <\date>
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , matcher
stretcher , matcher , stretcher
matcher , stretcher , matcher
stretcher , matcher , fasta
matcher , fasta , blast
fasta , blast , makeblastdb
blast , makeblastdb , blast

% <user> biofor02 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , dottup
dotmatcher , dottup , dottup
dottup , dottup , dotmatcher
dottup , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , fasta
sim_lav , fasta , blast

% <user> biofor02 <\user> <date> 2003/05/14 <\date>
ps_scan , ps_scan , ps_scan
ps_scan , ps_scan , bscan
```

```
ps_scan , bscan , bscan
bscan , bscan , bscan
bscan , bscan , ehmpfam
bscan , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , eprimer3
fuzzpro , eprimer3 , getorf
eprimer3 , getorf , eprimer3
getorf , eprimer3 , getorf
eprimer3 , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3

% <user> biofor04 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , sim_lav
sim_lav , sim_lav , sim_lav
sim_lav , sim_lav , fasta
sim_lav , fasta , blast

% <user> biofor04 <\user> <date> 2003/05/14 <\date>
ps_scan , bscan , bscan
bscan , bscan , bscan
bscan , bscan , ehmpfam
bscan , ehmpfam , ps_scan
ehmpfam , ps_scan , ps_scan
ps_scan , ps_scan , ps_scan
ps_scan , ps_scan , bscan
ps_scan , bscan , bscan
bscan , bscan , fuzzpro
bscan , fuzzpro , eprimer3
fuzzpro , eprimer3 , getorf
eprimer3 , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3
eprimer3 , eprimer3 , getorf
eprimer3 , getorf , eprimer3

% <user> biofor06 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , stretcher
stretcher , stretcher , matcher
```

```
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , fasta
sim_lav , fasta , fasta
fasta , fasta , blast
fasta , blast , makeblastdb
blast , makeblastdb , blast
makeblastdb , blast , blast
blast , blast , blast

% <user> biofor06 <\user> <date> 2003/05/14 <\date>
ps_scan , ps_scan , bscan
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ehmpfam
ehmpfam , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , fuzzpro
fuzzpro , fuzzpro , eprimer3
fuzzpro , eprimer3 , getorf
eprimer3 , getorf , getorf
getorf , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3

% <user> biofor08 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , sim_lav
sim_lav , sim_lav , sim_lav
sim_lav , sim_lav , fasta
sim_lav , fasta , blast
fasta , blast , makeblastdb
blast , makeblastdb , makeblastdb
makeblastdb , makeblastdb , makeblastdb
makeblastdb , makeblastdb , blast

% <user> biofor08 <\user> <date> 2003/05/14 <\date>
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ehmpfam
ehmpfam , ehmpfam , ehmpfam
ehmpfam , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , fuzzpro
```

```
fuzzpro , fuzzpro , getorf
fuzzpro , getorf , getorf
getorf , getorf , eprimer3

% <user> biofor09 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , sim_lav
sim_lav , sim_lav , sim_lav
sim_lav , sim_lav , fasta
sim_lav , fasta , fasta
fasta , fasta , fasta
fasta , fasta , blast
fasta , blast , blast
blast , blast , makeblastdb
blast , makeblastdb , blast
makeblastdb , blast , blast

% <user> biofor09 <\user> <date> 2003/05/14 <\date>
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ps_scan
ehmpfam , ps_scan , bscan
ps_scan , bscan , ehmpfam
bscan , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , getorf
fuzzpro , getorf , getorf
getorf , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3
eprimer3 , eprimer3 , getorf
eprimer3 , getorf , eprimer3
getorf , eprimer3 , eprimer3

% <user> biofor10 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , sim_lav
matcher , sim_lav , sim_lav
sim_lav , sim_lav , fasta
```

```
sim_lav , fasta , fasta
fasta , fasta , fasta
fasta , fasta , blast
fasta , blast , blast
blast , blast , blast
blast , blast , makeblastdb
blast , makeblastdb , blast

% <user> biofor10 <\user> <date> 2003/05/14 <\date>
ps_scan , ps_scan , ps_scan
ps_scan , ps_scan , bscan
ps_scan , bscan , bscan
bscan , bscan , ehmpfam
bscan , ehmpfam , ehmpfam
ehmpfam , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , getorf
fuzzpro , getorf , eprimer3
getorf , eprimer3 , eprimer3
eprimer3 , eprimer3 , eprimer3

% <user> biofor11 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , sim_lav
matcher , sim_lav , fasta
sim_lav , fasta , fasta
fasta , fasta , fasta
fasta , fasta , blast
fasta , blast , makeblastdb
blast , makeblastdb , blast
makeblastdb , blast , blast
blast , blast , blast

% <user> biofor11 <\user> <date> 2003/05/14 <\date>
ps_scan , ps_scan , ps_scan
ps_scan , ps_scan , bscan
ps_scan , bscan , bscan
bscan , bscan , bscan
bscan , bscan , ehmpfam
bscan , ehmpfam , ehmpfam
ehmpfam , ehmpfam , ehmpfam
ehmpfam , ehmpfam , fuzzpro
ehmpfam , fuzzpro , fuzzpro
fuzzpro , fuzzpro , fuzzpro
fuzzpro , fuzzpro , getorf
fuzzpro , getorf , eprimer3
getorf , eprimer3 , eprimer3
```

```
eprimer3 , eprimer3 , eprimer3

% <user> biofor11 <\user> <date> 2003/05/15 <\date>
blast , blast , blast

% <user> biofor11 <\user> <date> 2003/05/16 <\date>
blast , blast , blast

% <user> biofor13 <\user> <date> 2003/05/13 <\date>
dotmatcher , dotmatcher , dotmatcher
dotmatcher , dotmatcher , stretcher
dotmatcher , stretcher , stretcher
stretcher , stretcher , stretcher
stretcher , stretcher , matcher
stretcher , matcher , matcher
matcher , matcher , matcher
matcher , matcher , stretcher
matcher , stretcher , sim_lav
stretcher , sim_lav , sim_lav
sim_lav , sim_lav , sim_lav
sim_lav , sim_lav , fasta
sim_lav , fasta , fasta
fasta , fasta , fasta
fasta , fasta , blast
fasta , blast , makeblastdb

% <user> biofor13 <\user> <date> 2003/05/14 <\date>
ps_scan , ps_scan , ps_scan
ps_scan , ps_scan , bscan
ps_scan , bscan , ehmpfam
bscan , ehmpfam , ps_scan
ehmpfam , ps_scan , ps_scan
ps_scan , ps_scan , fuzzpro
ps_scan , fuzzpro , getorf
fuzzpro , getorf , getorf
getorf , getorf , eprimer3
```

=== Run information ===

```
Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:    EmbossLearningDataset
Instances:   397
Attributes:  3
              one
              two
              three
Test mode:   10-fold cross-validation
```

=== Classifier model (full training set) ===

J48 pruned tree

```
-----  
  
two = blast  
| one = blast: blast (39.0/2.0)  
| one = blast2seq: blast (0.0)  
| one = bscan: blast (0.0)  
| one = dialign: blast (0.0)  
| one = diffseq: blast (0.0)  
| one = dotmatcher: blast (0.0)  
| one = dottup: blast (0.0)  
| one = ehmpfam: blast (0.0)  
| one = eprimer3: blast (0.0)  
| one = fasta: makeblastdb (8.0/2.0)  
| one = fuzzpro: blast (0.0)  
| one = getorf: blast (0.0)  
| one = makeblastdb: blast (3.0)  
| one = matcher: blast (0.0)  
| one = megamerger: blast (0.0)  
| one = needle: blast (0.0)  
| one = plotorf: blast (0.0)  
| one = ps_scan: blast (0.0)  
| one = pscan: blast (0.0)  
| one = seqmatchall: blast (0.0)  
| one = sim_lav: blast (0.0)  
| one = stretcher: blast (0.0)  
two = blast2seq: dotmatcher (0.0)  
two = bscan: ehmpfam (24.0/12.0)  
two = dialign: dotmatcher (0.0)  
two = diffseq: dotmatcher (0.0)  
two = dotmatcher: dotmatcher (75.0/11.0)  
two = dottup: dotmatcher (2.0/1.0)  
two = ehmpfam: ehmpfam (24.0/12.0)  
two = eprimer3  
| one = blast: eprimer3 (0.0)  
| one = blast2seq: eprimer3 (0.0)  
| one = bscan: eprimer3 (0.0)  
| one = dialign: eprimer3 (0.0)  
| one = diffseq: eprimer3 (0.0)  
| one = dotmatcher: eprimer3 (0.0)  
| one = dottup: eprimer3 (0.0)  
| one = ehmpfam: eprimer3 (0.0)  
| one = eprimer3: eprimer3 (14.0/2.0)  
| one = fasta: eprimer3 (0.0)  
| one = fuzzpro: getorf (3.0)  
| one = getorf: eprimer3 (9.0/1.0)  
| one = makeblastdb: eprimer3 (0.0)
```

```

| one = matcher: eprimer3 (0.0)
| one = megamerger: eprimer3 (0.0)
| one = needle: eprimer3 (0.0)
| one = plotorf: eprimer3 (0.0)
| one = ps_scan: eprimer3 (0.0)
| one = pscan: eprimer3 (0.0)
| one = seqmatchall: eprimer3 (0.0)
| one = sim_lav: eprimer3 (0.0)
| one = stretcher: eprimer3 (0.0)
two = fasta: fasta (32.0/10.0)
two = fuzzpro: fuzzpro (19.0/9.0)
two = getorf: eprimer3 (17.0/5.0)
two = makeblastdb: blast (11.0/4.0)
two = matcher: matcher (35.0/11.0)
two = megamerger: dotmatcher (0.0)
two = needle: dotmatcher (0.0)
two = plotorf: dotmatcher (0.0)
two = ps_scan: ps_scan (24.0/9.0)
two = pscan: dotmatcher (0.0)
two = seqmatchall: dotmatcher (0.0)
two = sim_lav: sim_lav (20.0/9.0)
two = stretcher: stretcher (38.0/12.0)

```

Number of Leaves : 64

Size of the tree : 67

Time taken to build model: 0.09 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	263	66.2469 %
Incorrectly Classified Instances	134	33.7531 %
Kappa statistic	0.6253	
Mean absolute error	0.0374	
Root mean squared error	0.1421	
Relative absolute error	45.3412 %	
Root relative squared error	70.0959 %	
Total Number of Instances	397	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.763	0.018	0.882	0.763	0.818	blast
0	0	0	0	0	blast2seq
0.05	0.024	0.1	0.05	0.067	bscan
0	0	0	0	0	dialign
0	0	0	0	0	diffseq

0.985	0.036	0.842	0.985	0.908	dotmatcher
0	0.003	0	0	0	dottup
0.667	0.059	0.421	0.667	0.516	ehmpfam
0.886	0.03	0.738	0.886	0.805	eprimer3
0.688	0.038	0.611	0.688	0.647	fasta
0.316	0.019	0.462	0.316	0.375	fuzzpro
0	0.018	0	0	0	getorf
0.5	0.01	0.6	0.5	0.545	makeblastdb
0.686	0.03	0.686	0.686	0.686	matcher
0	0	0	0	0	megamerger
0	0	0	0	0	needle
0	0	0	0	0	plotorf
0.789	0.024	0.625	0.789	0.698	ps_scan
0	0	0	0	0	pscan
0	0	0	0	0	seqmatchall
0.35	0.024	0.438	0.35	0.389	sim_lav
0.684	0.033	0.684	0.684	0.684	stretcher

Bibliographie

- [Aha D.W. and H., 2000] Breslow L.A. Aha D.W. and Muñiz-Avila H. Conversational case-based reasoning. *Applied Intelligence*, 14 :9–32, 2000.
- [BPML, 2002] BPML. Business process modeling language, 2002.
- [Breiman L. and Stone, 1984] A. Olshen Breiman L., R. Friedman and J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- [Clark and Niblett, 1989] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3 :261–283, 1989.
- [Cypher, 1993] A. Cypher. *Watch What I Do. Programming by Demonstration*. MIT Press, 1993.
- [Dallons and Buyle, 2003] Quentin Dallons and Pierre Buyle. Partage de ressources bioinformatiques hétérogènes - conception et implémentation d'une fédération de médiateurs. Master's thesis, Institut d'Informatique, 2003.
- [de Bruijn and Martin, 2002] Berry de Bruijn and Joel Martin. Literature mining in molecular biology, 2002.
- [Domingos, 1994] P. Domingos. The RISE system : Conquering without separating, 1994.
- [Domingos, 1996] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2) :141–168, 1996.
- [Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- [Goble *et al.*, 2003] Carole Goble, Chris Wroe, Robert Stevens, and the myGrid consortium. The myGrid Project : Services, Architecture and Demonstrator, 2003.
- [Hirsh, 1992] Haym Hirsh. *Polynomial-Time Learning with Version Spaces*. MIT Press, proceedings of the tenth national conference on artificial intelligence (aaai92) edition, 1992.
- [Lieberman, 2000] H. Lieberman. *Your Wish is My Command : Programming By Example*. The Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann, 2000.
- [Mitchell, 1997] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [Quinlan, 1993] J.R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
- [Rich, 2001] I. Rich. An empirical study of the naive bayes classifier. *Workshop on Empirical Methods*, 2001.

- [Ruvini, 2000] J.-D. Ruvini. *Assistance à l'utilisation d'un environnement interactif : Apprentissage des habitudes de l'utilisateur*. Thèse de Doctorat. Université de Montpellier II, 2000.
- [van der Aalst and van Hee, 2002] W. van der Aalst and K. van Hee. *Workflow Management : Models, Methods, and Systems*. MIT Press, 2002.
- [Witten and Frank, 2000] Ian H. Witten and Eibe Frank. *Data Mining : Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.