Université de Strasbourg
IUT Robert Schuman

Maître de stage : Nicolas Lachiche
Tuteur universitaire : Marie-Paule Muller

Rapport de stage

Jonathan HAEHNEL

Strasbourg, le 10 avril 2012

Table des matières

Remerciements

J'adresse mes remerciements au laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection pour m'avoir permis d'effectuer mon stage au sein de leur service.

Je remercie plus particulièrement :

- Monsieur *Nicolas Lachiche*, mon maître de stage, qui a fait preuve d'une grande disponibilité à mon égard, pour m'avoir fait confiance dans la réalisation.
- Madame *Marie-Paule Muller*, ma tuteur, pour m'avoir consacré un peu de son temps.
- Je remercie également tous *les autres stagiaires* que j'ai pu côtoyer et qui ont rendu mon stage plus convivial.
- Enfin, je remercie l'ensemble des *professeurs du département Informatique de l'IUT Robert Schuman* pour m'avoir épauler durant ces deux dernières années.

Introduction

Afin de mettre en pratique les acquis obtenus durant l'ensemble de ma formation, j'ai effectué un stage de 10 semaines. Ce stage a consisté à développer un environnement de test pour certains algorithmes de fouille de données. Je développerai cette notion tout au long de ce rapport.

Ce rapport présente le travail que j'ai effectué lors de mon stage au sein l'équipe FDBT [1] du LSIIT [2]. Il s'est déroulé du 11 avril au 18 juin 2010 au Pôle API à Illkirch. Pendant le stage, je me suis familiarisé avec un environnement technique spécifique à la fouille de données.

Le projet s'est avéré très intéressant et très enrichissant pour mon expérience professionnelle. En effet, même si ma formation ne s'inscrit pas directement dans le cadre de la fouille de données, cela m'a permis d'en comprendre les bases et les fondements. Grâce à ce stage, j'ai travaillé sur des projets qui m'ont permis d'entrevoir en quoi consiste la profession de chercheur dans le domaine de l'informatique.

Dans un premier temps, ce rapport vous présentera le contexte général du stage, c'est-à-dire, l'entreprise d'accueil, l'existant autour du projet et les différents besoins qui ont motivé ce stage. Ensuite, j'expliquerai les différents aspects qui ont rythmé mon stage. Enfin, nous interpréterons les résultats par rapport aux objectifs initiaux ainsi que les nombreux apports du stages.

1. Fouille de Données et Bioinformatique Théorique
2. Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection

Première partie

Le contexte

Chapitre 1

Présentation de l'entreprise

1.1 Sa philosophie

J'ai effectué mon stage au Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection. C'est un laboratoire de recherche *multi-disciplinaire fédéré par l'imagerie.*
Les grandes disciplines qui y sont représentées sont :

- l'informatique
- le traitement du signal
- l'automatique
- la télédétection

Dans tous ces domaines, l'image joue un rôle majeur. En effet, c'est un type de données complexe privilégié dans les travaux sur l'algorithmique, la programmation, la classification, la fouille de données et l'asservissement [1] visuel.

Le laboratoire est rattaché administrativement à l'Ecole Nationale Supérieure de Physiques de Strasbourg (ENSPS) et il dépend de 3 instituts du CNRS [2] : l'Institut des Sciences Informatiques et de leurs Interactions (INS2I) en rattachement principal, l'Institut des Sciences de l'Ingénierie et des Systèmes (INSIS) et l'Institut national des Sciences de l'Univers (INSU), en rattachements secondaires.

FIGURE 1.1 – Logo du laboratoire

1. Stratégie de commande permettant de soumettre des systèmes robotiques (bras manipulateur, robot mobile) à une position fixe ou variable en utilisant des informations visuelles.
2. Centre national de la recherche scientifique

1.2 Ses activités

Le laboratoire compte un effectif de 174 personnes (avec 98 permanents et 76 non permanents) répartis dans 7 différentes équipes de recherche :

FIGURE 1.2 – Organigramme des activités

Détail des équipes :

1. L'équipe IGG concentre ses activités de recherche autour de la modélisation géométrique.
2. L'équipe MIV travaille principalement le traitement et l'analyse des images. Ses recherches s'appuient sur de solides bases mathématiques et visent particulièrement les développements algorithmiques novateurs. Les applications sont multiples (imagerie médicale, imagerie astronomique, réalité augmentée, métrologie).
3. L'équipe AVR s'intéresse principalement au domaine de la robotique dans le milieu médical. La robotique médicale a pour objectif principal l'assistance au geste médical pour une amélioration de la qualité des soins.
4. L'équipe RP étudie et conçoit des algorithmes, protocoles et architectures de communication en s'intéressant à tous les éléments d'un réseau de type internet nouvelle génération et IPv6, depuis les réseaux d'extrémités sans fil jusqu'aux problèmes de routage dans le coeur de réseau.
5. L'équipe ICPS étudie la programmation parallèle, et plus particulièrement de la personnalisation et de l'optimisation automatique des programmes.
6. L'équipe TRIO se concentre principalement sur la physique de la mesure associée à des données de type "image", en liaison avec l'observation spatiale de la terre et la modélisation de la biosphère (télédétection).
7. L'équipe FDBT m'accueillant dans le cadre du stage, je la détaillerai plus longuement dans le section **??**

1.3 Zoom sur l'équipe d'accueil

Durant mon stage, j'ai été affecté à l'équipe de Fouille de Données et Bioinformatique Théorique, et plus particulièrement dans la thématique de «Fouille de données et de classification». On compte 8 permanents travaillant sur cette problématique.

Qu'est-ce que la fouille de données ?

C'est un processus qui permet l'extraction de connaissances à partir de données complexes *(par exemple : images, bases de données, etc...)* à travers des méthodes d'apprentissage automatique ou semi-automatique.

A quoi sert la fouille de données ?

Le but de ces recherches est double : il consiste d'une part, à étudier et développer des méthodes d'extraction de connaissances, et d'autre part, à appliquer ces méthodes à l'analyse de bases de données et d'images numériques. Les approches étudiées et développées sont basées sur des méthodes d'apprentissage, de classification non-supervisée et de fouille de données relationnelles.

Quelles applications existe t-il ?

Les principaux domaines d'applications sont les images de télédétection ou médicales, les données biochimiques, ou encore la gestion de données client.

Illustration : «les îlots»

Chaque îlot possède un ou plusieurs bâtiments avec une aire respective. L'objectif de la fouille de données est de trouver une rêgle permettant de dire que l'îlot est soit un îlot pavillonnaire *(par exemple : beaucoups de petits bâtiments)* ou encore un îlot collectif *(par exemple : un bâtiment de grande superficie)*

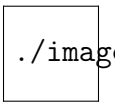


FIGURE 1.3 – Prédiction de la classe d'un îlot

Chapitre 2

Présentation de l'existant

2.1 Des algorithmes de fouille de données

Dans cette partie, je vais présenter brièvement les algorithmes que j'ai rencontré durant mon stage. Pour information, ces algorithmes sont déjà implémentés dans de nombreux logiciels. En effet, durant le stage, il n'est pas demandé de les corriger ou de les réécrire. **Ces principes sont plutôt complexes, c'est pourquoi durant tout le rapport, je vais utiliser la même structure de données et le même jeu de données : la base de données relationnelles** [1] **ILOT.** *(voir annexe* **??***)*

2.1.1 La propositionalisation

La propositionalisation s'applique directement sur les données de la base. Ce mécanisme est utilisé par l'algorithme de Cardinalisation ou Relaggs qui l'implémentent chacun de manière différentes (que nous verrons plus tard). Il existe de nombreux algorithmes de propositionnalisation, mais, dans tous les cas, l'objectif recherché est **le passage de plusieurs tables relationnelles à une seule table**.

C'est quelque chose d'important en fouille de données, car on disposent souvent (à l'image des tables îlot-bâtiment) de plusieurs table d'entrée alors que les **logiciels d'interprétation des données n'en demandent que une seule**, il est donc essentiel d'effectuer une fusion de table grâce à la propositionnalisation.

Exemple : Pour chaque îlot, on calcule le nombre de bâtiments associées et on ajoute ce champs dans une table annexe. Cet algorithme fonctionnne à la fois sur des bâtiments ayant une aire numérique et également sur des bâtiments ayant une aire catégorielle [2]*.*

Table : resultat	
idîlot	nbbâtiment
1	5
2	3

TABLE 2.1 – Résultat après propositionalisation

1. Une base de données relationnelle est une base de données structurée avec une relation par type d'objet.
2. Aire sous forme de chaîne de caractères (exemple : petit, moyen et grand)

2.1.2 Algorithme de cardinalisation

Cet algorithme cherche à déterminer pour chaque îlot, l'aire minimum de l'îlot, bâtiment après bâtiment. On cherche l'aire telle que le nombre de bâtiments soit supérieur ou égal à 1 jusqu'au nombre maximum de bâtiments des îlots.

Table : resultat							
idîlot	aireîlot	aireminbat1	aireminbat2	aireminbat3	aireminbat4	aireminbat5	classe
1	400	25	30	60	70	80	yes
2	375	15	55	90	-	-	yes

TABLE 2.2 – Résultat après cardinalisation

2.1.3 Algorithme «RELAGGS»

Cet algorithme permet de faire resortir de nouvelles informations du modèle : par exemple, pour chaque îlot, il calcule le bâtiment avec le minimum d'aire, le bâtiment avec le maximum d'aire, la moyenne des aires, etc...

Table : resultat					
idîlot	aireîlot	airemin	airemax	aireavg	classe
1	400	25	80	53	yes
2	375	15	90	32	yes

TABLE 2.3 – Résultat de RELAGGS

Il est possible de l'utiliser avec un autre algorithme : **la discretisation**. Celui-ci découpe le modèle en intervalle d'aire et qui pour chaque intervalle d'aire de l'îlot calcule le nombre de bâtiments associés à l'intervalle.

Table : resultat						
idîlot	aireîlot	nbaire1	nbaire2	nbaire3	nbaire4	classe
1	400	1	2	4	5	yes
2	375	1	1	2	3	yes

TABLE 2.4 – Résultat après le passage de l'algorithme pour des intervalle de 0 à 100 avec un pas de 25

2.2 Une multitude de logiciels

Durant mon stage, j'ai aussi du m'habituer à l'utilisation et comprendre le fonctionnement de nombreux logiciels de fouille de données. Ces logiciels devaient travailler en colaboration, car l'on prenait le résultat de l'un comme parametre d'entrée de l'autre.

2.2.1 DataGenerator

Le **DataGenerator** est un petit logiciel codé en Java avec une interface graphique permettant de générer aléatoirement des données dans une base de données en définisant les valeurs maximales des champs des différentes tables.
Par exemple : générer 100 instances d'îlots qui ont au maximum 10 instances de bâtiments, chacun avec une aire d'un bâtiment variant de 0 à 100.

Après avoir généré les données, le logiciel intègre une seconde fonctionnalité : **l'étiquetage des données**. Il permet de donner une valeur (par défaut, vrai ou faux) à une ligne de la table, si celle-ci respecte une condition de filtrage.
Par exemple pour la table îlot, la colonne «classe» est vraie, s'il y a au moins un bâtiment ayant une aire supérieure à 50, sinon c'est faux.

2.2.2 Proper

Proper est un vaste logiciel codé en Java. Je n'ai utilisé que les deux fonctionnalités suivantes :

1. Le **Builder** qui permet de choisir et de configurer les différents algorithmes présentés dans la section **??**. A ce stade le logiciel génère un fichier XML avec toutes les informations et paramètres choisis.
2. Le **Runner** permet à partir du fichier XML généré à l'étape précédente, d'exécuter les algorithmes choisis dans le Builder. Les tables résultats s'ajoutent au serveur et un fichier de type «ARFF» est généré par le système.

2.2.3 WEKA explorer

Ce logiciel permet d'interpréter les résultats des différents algorithmes. En effet, il prend en entrée un fichier ARFF et permet de configurer et d'exécuter l'un des classeurs WEKA [3] sur l'ensemble de données actuel. On peut choisir d'effectuer une validation croisée ou de tester sur un ensemble de données distinct.
Le résultat de la validation est un pourcentage de réussite.

3. Un classeur est un outil permettant de construire un arbre de décision et ainsi prédire la classe d'une donnée.

Chapitre 3

Identification d'un besoin

3.1 Etat actuel

L'objectif principal du stage est le test des différents algorithmes de fouille de données en faisant varier un paramètre initial. Nous devons donc réussir à répondre à de nombreuses interrogations :

- Il faut trouver les jeux de données les plus efficaces pour chaque algorithme.
- Il faut également trouver les avantages et inconvénients d'un algorithme par rapport à un autre (stabilité, rapidité et robustesse avec de grosses quantités de données)
- Il faut trouver les similitudes en terme de réussite des différents algorithmes.
- Il faut dessiner et interpréter le comportement d'un algorithme en faisant varier les données initiales.

Avant le stage, il était déjà possible de réaliser des tests en utilisant la méthode suivante :

1. Vider manuellement *(avec une requette SQL)* les données de la base de données de travail. *(par exemple : les tables îlot-bâtiment)*
2. Ouvrir le DataGenerator, choisir un paramètre à faire varier *(par exemple : l'aire d'un bâtiment)*, choisir une valeur initiale *(par exemple : une aire de 0 à 20 maximum pour commencer)*, lancer la génération.
3. Etiqueter les données par rapport à une condition *(par exemple : la colonne classe)*
4. Lancer le Builder de proper, et choisir la configuration de l'algorithme à tester.
5. Enregistrer le fichier XML.
6. Ouvrir le Runner de proper, charger le fichier XML et exécuter. *(un fichier ARFF est créé au passage de l'algorithme)*
7. Ouvrir l'explorer de WEKA, choisir le mode de validation, lancer le processus et récupérer la valeur de réussite.
8. Recommencer les opérations suivantes : 1 - 2 - 3 - 6 - 7 en augmentant le paramètre choisis au début du test. *(par exemple : l'aire de 0 à 40)*

./images/schema_testmanuel.png

FIGURE 3.1 – Schématisation du processus manuel de test

3.2 Problématique abordée

Comme vous pouvez le constater le processus est fonctionnel, mais il est très long et fastidieux à réaliser à grande echelle (avec de nombreux points de mesure et beaucoup de données à traiter). Il faut switcher sans cesse de logiciel en logiciel, ce qui accroit le risque d'erreur.

Par exemple : en testant les quatres algorithmes et en faissant 10 mesures (param. variable de 10 à 100), il faut faire pas moins de ***1600 manipulations*** *: voir* ***Annexe ??***

Avec le nombre important de tests à réaliser durant le stage, il devait être essentiel de réduire la durée de chacun des tests. **C'est pourquoi l'idée d'une application en Java permettant d'effectuer de facon automatique tes tests sur certains algorithmes est née.** A la fin du processus, l'application nous communiquera directement les résultats sous forme de courbe.

Certes, en développant cette application, une partie du stage serait «perdue». Mais, d'un autre côté, cela accélèrera les tests durant le stage et même ceux réalisés par l'équipe après le stage. C'est donc doublement bénéfique !

Deuxième partie

Un projet en trois temps

Chapitre 4

Modification ergonomique de «Proper»

Le «Builder» est un logiciel complet et efficace, mais souvent cité pour sa mauvaise ergonomie. En effet, certains points du logiciel sont mal conçus et ont une logique déroutante. Cette première phase du stage du fait de sa courte durée (cf. planning prévisionnel : *Annexe* **??**) peut être considéré comme une mise en bouche.

4.1 Problèmes

Dans cette partie, je vais vous lister les différents problèmes que j'ai corrigé dans l'interface graphique. Pour faciliter votre compréhension, les explications seront toujours illustrées d'une capture d'écran. Ces captures d'écran concernent uniquement l'onglet «Cardinalization» du Builder, mais, il faut savoir que les problèmes se répètent dans un grand nombre d'onglets.

La connexion à une base de donnée

Dans les sous-onglets «Proper» et «Export», on nous demande de renseigner une base de donnée. En cliquant sur le bouton «...», une boite de dialogue [1] s'ouvre, celle-ci nous permet de nous connecter.

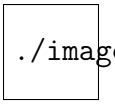


FIGURE 4.1 – Ancien système de connexion à une base de données

ON PEUT RAPIDEMENT IDENTIFIER DEUX PROBLÈMES :

- Il n'est **pas utile d'afficher le driver** [2], car ce dernier est facilement détectable en interne. L'utilisateur n'est pas censé connaître ce champ pour se connecter.
- De plus, l'**URL de connexion** [3] **n'est pas nécessaire**, on peut la générer en interne avec le host [4], le port et le type de base de données.

1. En informatique, une boîte de dialogue est un composant d'interface graphique affichée par un programme pour informer l'utilisateur d'un événement ou obtenir une information de l'utilisateur.
2. Le driver est un «logiciel» qui permet d'établir une connexion entre un programme java et un système de gestion de bases de données.
3. L'URL est du type : `jdbc:TYPE_BD://HOST:PORT/[DATABASE]`
4. nom ou IP de la machine contenant les bases de données.

Après avoir remplit tous les champs, on peut enfin choisir notre base de données dans une liste déroulante. L'**information du driver persiste**, celle-ci est encore plus inutile sur cette nouvelle page.

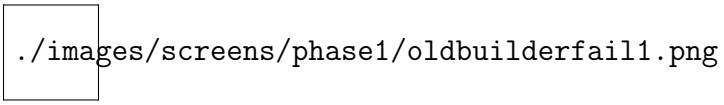


FIGURE 4.2 – Interface permettant le choix d'une base de données

Choix d'une table et d'un champ

De la même manière, dans une autre boite de dialogue, il faut choisir une table et un champ de travail.

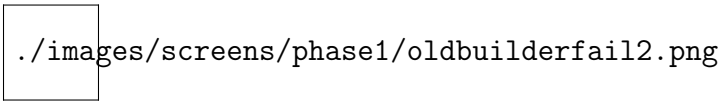


FIGURE 4.3 – Interface permettant le choix d'une table

Le driver est toujours visible. De plus, pour choisir une table, il faut **choisir à nouveau la base de données** (alors que celle-ci a été choisie à l'étape précédente), c'est vraiment pas logique.

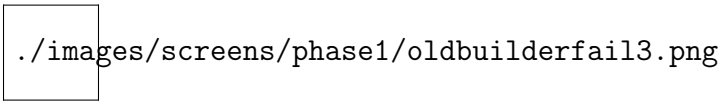


FIGURE 4.4 – Interface permettant le choix d'un champ

De même s'il l'on veut choisir un champ, il faut re-sélectionner la base de donnée, puis la table.

Par ailleurs, le logiciel **ne gère pas les erreurs de chronologies**, c'est à dire que l'on peut facilement choisir une table avant d'avoir choisit une base de donnée, aucune erreur n'est émise.

Double cases à cocher

Dans toute l'application, on trouve des propriétés à cocher, mais certaines possèdent une seconde case totalement inutile à droite.

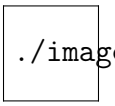


FIGURE 4.5 – Illustration des doubles cases à cocher

4.2 Les changements réalisés

Pour corriger ses défauts, j'ai du créer *quatre nouvelles classes et modifier trois classes déjà existantes.*

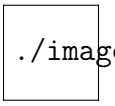


FIGURE 4.6 – Nouvelle interface de connexion à une base de données

La **boite de dialogue de connexion a été complément refondue**, l'URL de connexion et le driver sont déterminés en interne.
Pour information, le champ «Default Database» n'est que éditable quand le type de base est PostgreSQL, car ce dernier a besoin de connaître une base pour lister toutes les bases contenues sur le serveur.

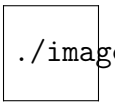


FIGURE 4.7 – Choix d'une base, table, champ et lancement du message d'erreur.

Une fois connecté, l'on peut apercevoir la liste des bases de données. *Pour le choix d'un table et d'un champ les informations précédentes ne sont plus demandées !*
De plus, si l'on saute des étapes, un message d'erreur est lancé.

Concernant les double cases à cocher, j'ai supprimé celle à droite, de toute façon, elle n'avait aucune action sur le code.

Chapitre 5

Automatisation d'un environnement de tests

5.1 Détails sur le processus de test

Avant de commencer le développement, une longue analyse s'impose. En effet, il faut d'abord étudier l'existant, car notre application doit utiliser simultanément trois logiciels et plusieurs algorithmes déjà implémentés.
Dans ce chapitre, je répondrais à certaines questions fondamentales pour la compréhension des parties futures.

Pourquoi effectuer des tests ?

Le laboratoire a conçu et implémenté deux nouveaux algorithmes dans le domaine de la fouille de données (la cardinalisation et la cardinalisation avec des quantiles). Il est donc intéressant déterminer les points forts de ses algorithmes par rapport aux algorithmes déjà existants (Discrétisation et RELAGGS).

Qu'est ce qu'un test dans notre contexte ?

Un test est mécanisme automatique permettant de dégager des caractéristiques *(temps d'exécution, réussite, etc..)* pour chaque algorithme sur un jeu d'essai. [1]

Ces différents résultats sont ensuite enregistrés pour pouvoir dégager des propriétés communes ou non, aux algorithmes que l'on a testé.

Dans notre application, un test n'est en réalité pas un seul traitement sur certains algorithmes mais, une série de traitement. En effet, nous choisissons un paramètre variable avec un certain taux d'accroissement et pour chacun de ses états, nous relançons le traitement.

FIGURE 5.1 – Schématisation d'un test

1. Données initiales contenues sur un support (base de données ou fichier) utilisées par les algorithmes de fouille de données

Comment peut t-on représenter un test ?

Notre test est représenté par un suite d'instructions que l'on va appelées : **«PROCESSUS DE TEST»**.

Notre application sera capable de tester les quatre algorithmes simultanément :
- La cardinalisation *(voir partie* **??***)*
- RELAGGS *(voir partie* **??***)*
- La cardinalisation avec les quantiles *(algorithme combinant discrétisation, puis cardinalisation)*
- La discretisation *(voir partie* **??***)*

Les caractéristiques récupérées à chaque étape sont :
- Le pourcentage de validation (réussite) de l'algorithme
- Le temps d'exécution de l'algorithme
- Le temps d'exécution de la procédure de validation (WEKA)
- Le nombre de nœuds de l'arbre de décision créer lors de la validation
- Le nombre de colonnes dans la table créée par l'algorithme

Quelles sont les étapes du processus de test ?

Dans un premier temps, il faut **configurer l'environnement de test**. C'est à dire, il faut choisir :

- les algorithmes à tester
- la base de données de travail
- le paramètre de test variable
- les paramètres pour la génération aléatoire des données
- les paramètres pour l'étiquetage des données
- les paramètres pour l'algorithme de discrétisation
- le classeur WEKA à utiliser

Information : *Le résultat est la création d'un fichier XML contenant les informations saisis ci-dessus.*

Une fois la configuration terminée, on peut réellement **lancer le processus** qui est le même pour chaque algorithme.

Etape 1 : Nettoyage de la base de données

Etape 2 : Génération des données aléatoirement

Etape 3 : Étiquetage des données

Etape 4 : Lancement de l'algorithme (mécanisme de propositionalisation ou de discrétisation)

Etape 5 : Exporter les résultats de l'algorithme dans un fichier ARFF

Etape 6 : Validation du fichier ARFF avec Weka explorer

Etape 7 : Enregistrement de toutes les caractéristiques trouvées dans un fichier CSV ou dans une base de données.

Ces étapes se répètent pour chaque état du paramètre variable ainsi que pour chaque algorithme choisi.

Ci-dessous, un schéma permettant d'expliquer simplement les différentes étapes du processus :

FIGURE 5.2 – Étapes d'un processus de test

5.2 DataTester : le script

5.2.1 Explications

Dans un premier temps, j'ai du créer un **script Java permettant de réaliser les grandes actions de l'application finale automatiquement et sans interface utilisateur.**

Le script et ses classes peuvent donc être considérés comme le moteur de l'application. L'interface graphique n'est qu'un supplément que l'on viendra fixer sur ce script. Commencer par la création du script permet de ne pas partir dans toutes les directions et de bien structurer sa réflexion *(les choses de second plan comme la GUI sont écartées pour le moment).*

Au total, le script compte **11 classes Java** répartis sur deux packages. *(voir Annexe ??)* Il utilise plusieurs logiciels existants de manière plus ou moins explicite. Le lancement de chaque algorithme et l'export des résultats est directement effectué en ligne de commande (via Proper). De même pour la validation des résultats (via WEKA). Par contre, les sources des petits logiciels permettant la génération aléatoire des données, l'étiquetage des données et la discrétisation des données ont été directement intégrées dans notre script.

5.2.2 Modifications apportées au processus de test

Durant la réalisation du script et de l'application, j'ai du modifier mes plans initiaux. En effet, il y a eu de nombreuses modifications du cahier des charges et du planning. Certains changements étaient si profond qu'ils ont même affectés le processus de test.

Absence du fichier XML :

Finalement, j'ai réussis à me passer du fichier XML de configuration généré par Proper pour lancer un algorithme et pour exporter ses résultats. J'appelle directement l'algorithme en ligne de commande de la manière suivante :

```
1
2 java proper.app.CARDINALIZER -any_index -database johnny -driver
3 org.postgresql.Driver -exclude_tables _flat,_relaggs,_remilk,_*,
4 _cardinalized,_recursed,_file*,_identifier_ -field classe -join
5 leftouter -max_depth =1 -password mot2pass result_table _cardinalized
6 -table ilot -url jdbc:postgresql://localhost:5432/ -user postgres
7
8 Dans le cas, general: java CHEMIN_ALGORITHM LIST_PARAMETRE
```

Enregistrement des résultats sur plusieurs supports :

A la fin de chaque test, les résultats sont enregistrés dans un fichier CSV permettant de retracer directement les courbes dans un logiciel de calcul (excel ou openoffice calc). Cependant, si l'on veut affiner certaines valeurs du test (rajout de valeurs) , nous étions obligé de recommencer entièrement le test et donc de recalculer les valeurs déjà trouvées auparavant.

Ce problème a été résolus en enregistrant également les données dans une base de données spécifique, il suffira de rajouter simplement les nouvelles valeurs (points de la courbe) dans la table.

Répétitions du test :

En observant de plus près, les courbes résultats après plusieurs tests, aucune courbe n'était vraiment exploitable, en effet, l'amplitude des variations était trop forte et trop rapide. Pour modifier cela, j'ai mis en place un système de répétition. Chaque test est répété n fois et, l'on enregistre la moyenne et l'écart-type de tous les résultats dans le CSV.

Différents mode de lancement d'un test :

1. Directement par ligne de commande sans interface graphique en donnant en paramètre le fichier de configuration. *(mode d'utilisation avancée)*
2. En mode graphique et en complètant les différentes pages de configuration. *(utilisation normale)*
3. En mode graphique et en ouvrant directement le fichier de configuration *(utilisation graphique avancée)*

Le processus de configuration manuel d'un test avec l'interface graphique est assez long, c'est pourquoi, il est plus rapide d'ouvrir directement un fichier de configuration.

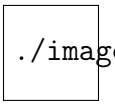


FIGURE 5.3 – Représentation schématique de l'application

Pour bien comprendre... rien ne vaut un bon schéma !

5.3 DataTester : l'interface graphique

Pour faciliter la configuration du test et également la lecture des résultats, il est important de réaliser une interface graphique.

L'interface doit être simple à utiliser et doit laisser la possibilité à un utilisateur avancé d'éviter la configuration manuelle en utilisant un fichier de configuration. *(voir Annexe ??)* L'application doit être un minimum ergonomique même si elle ne sera que utilisé durant le stage.

D'un point de vue technique, elle est codée en *Swing* et utilise la librairie *JFreeChart* pour dessiner les graphiques. Elle se compose de **18 classes** réparties dans deux packages *(voir Annexe ??)*.

Je vais maintenant vous faire découvrir l'interface graphique, elle se compose de trois parties représentées **par trois onglets**.

5.3.1 La configuration

L'onglet *«Setup»* est le seul accessible au lancement de l'application. Il contient **6 sous-onglets** permettant la configuration successive des différentes instructions du processus de test.

C'est uniquement après avoir correctement complété cette partie que vous aurez accès au second onglet. Cela s'apparente à une installation d'un programme.

Les 6 étapes de la configuration :

General : Paramètres de base et choix des algorithmes à tester

Database : Connexion à la base de données de travail

Generation : Choix du paramètres de test variable et des paramètres de génération aléatoire des données

Labeling : Choix des étiquettes et de la condition d'étiquetage

Discretisation : Choix des paramètres utiles pour la discrétisation

Weka : Choix du classeur

./images/screens/phase2/confPanel.png

FIGURE 5.4 – GUI : Aperçu de la phase de configuration

5.3.2 L'exécution

Dans cet onglet, il est possible de lancer l'exécution du processus de test. Durant l'exécution, un champ de texte, nous informe des *différentes notifications*. On peut également suivre l'avancement du test sur la barre de progression. A tout moment, une pression sur le bouton «Stop» interrompt le test.

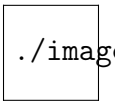


FIGURE 5.5 – GUI : Aperçu de la page d'exécution d'un test

5.3.3 L'affichage des résultats

Une fois l'exécution terminée, l'onglet *«Résultat»* se déverrouille. Pour chaque algorithme, **un graphique est crée** avec chaque caractéristiques enregistrées durant la procédure.

De plus, dans un second temps une table, nous affiche toutes les informations dans une base de données résultat.

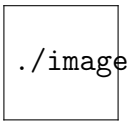


FIGURE 5.6 – GUI : Aperçu de l'affichage des résultats

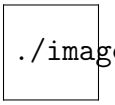
Chapitre 6

Réalisation des tests sur différents algorithmes et interprétation

Maintenant que le logiciel est plus ou moins fonctionnel, il est temps de l'utiliser et de commencer les différents tests. Durant cette dernière semaine de stage, j'ai en effet commencé quelques tests mais, nous n'avons pas eu le temps d'interpréter et de synthétiser tout les resultats. C'est dans cette situation que l'on s'apercoit que le stage était un peu trop court. Une semaine de plus n'aurait pas fait de mal.

Maintenant, je vais vous présenter notre premier test qui était exploitable.

Durant ce test, l'on teste les quatres algorithmes, l'on va particulièrement suivre les différences entre la Cardinalisation et la Discrétisation. Pour cela, on fait varier le nombre d'îlots de 50 à 500 avec un pas de 50 îlots. Chaque îlot possède 20 bâtiments. On fait donc 10 mesures par algorithmes. Voici, les courbes du test :



On peut voir que notre algorithme (la cardinalisation) **atteint les 100% de précision tandis que la discretisation plafonne à 95%.** De plus, les deux algorithmes sont identiques au niveaux du nombre de colonnes générées (23 colonnes chacun). De plus, les données resultats sont moins fiable que celle de la Cardinalisation, car le nombre de noeud de l'arbre de décision est suppérieur pour la Discretisation.

On peut donc conclure que pour ce jeu de données, notre algorithme s'en sort bien mieux que la Discrétisation ! Mais, il faudra vérifier cela dans d'autres tests.

Troisième partie

Retour d'expérience

Chapitre 7

Difficultés rencontrées

Un projet informatique n'est pas un long fleuve tranquille, il y a toujours des imprévus à gérer. En effet, durant mon stage, j'ai du faire face à de nombreux problèmes que ce soit durant l'analyse ou le développement.

Le contexte

Le stage s'inscrit dans une problématique complexe qu'est la fouille de donnée. En effet, il y a de nombreux concepts et principes à comprendre pour aborder sereinement le développement. Par ailleurs, il faux également savoir trier les informations et ne garder que celles qui sont utiles pour le développement.

Intégration des logiciels

Les logiciels (Proper, Weka, DataGenerator,etc...) ont du être utilisés directement par mon application. Cependant, ces logiciels ne sont pas tous optimisés pour une utilisation externe en ligne de commande, ils pouvaient uniquement être lancés à travers leur interface graphique respective, c'était pas super pratique. De fait, c'était parfois difficile de les intégrer dans mon application.

Cahier des charges incomplet

Le cahier des charges était trop évolutif. Bien qu'il y avais un cahier des charges au début du stage, il fut impossible de déterminer toutes les fonctionnalités dès le début, nous avions uniquement les grandes idées.

Au fur et à mesure du développement, il fallait rajouter beaucoup de nouvelles fonctionnalités. Il fallait donc revoir constamment le code de l'algorithme (moteur) de l'application. C'était pas toujours facile de le rendre compatible avec la nouvelle fonctionnalité à implémenter.

Exécutable JAR

J'ai aussi eu du mal pour générer mon premier exécutable Jar, car ce dernier possédait plusieurs niveaux de librairie, la solution était donc de laisser les librairies en dehors du JAR. J'ai perdu une bonne journée à cause de ce problème.

Chapitre 8

Le bilan

Ce stage a été une première expérience professionnelle dans le domaine très enrichissante sur tous les plans.

D'un **point de vue technique**, j'ai utilisé une grande partie des connaissances acquises pendant mes deux années de DUT informatique. Je me suis perfectionné dans la programmation orientée objet notamment dans l'utilisation des *librairies JDBC et Swing*. J'ai découvert une nouvelle librairie, *JFreeChart*, celle-ci permettant la création de diagrammes dans une application Java. J'ai aussi utilisé un nouvel environnement de développement *Eclipse* et le gestionnaire de base de données *PgAdminIII*.

Mon projet étant basé sur l'existant, j'ai du mettre en pratique les principes d'analyse et de modélisation (UML) que l'on a appris précédemment. Cela m'a permis d'éviter de me perdre dans le code source et de faire un découpage de mes classes plus efficace.

D'un **point de vue humain**, j'ai pu me faire une idée d'un projet informatique dans un milieu de recherche. La grosse différence, c'est qu'en recherche, il n'y a pas de cahier des charges définis au début d'un projet, on ajoute des fonctionnalités au fur et à mesure du développement pour tendre vers l'objectif de départ. De ce fait, l'application est un perpétuel prototype.

Il m'a permis de *développer mon esprit d'initiative, mon autonomie, d'acquérir le sens des responsabilités*.

Malgré que le domaine de la fouille de données est très vaste, j'ai commencé à comprendre certains concepts clés comme la propositionnalisation, la validation croisée, etc...

Conclusion

Au terme de ce stage, j'ai eu la satisfaction d'avoir réalisé une application répondant à l'objectif initial à savoir la réalisation de tests de certains algorithmes de fouille de données. Mais plus que cette satisfaction, j'ai eu le plaisir de travailler dans un domaine qui me passionne.

En effet, ce stage m'a permis non seulement d'approfondir mes connaissances en informatique mais aussi d'acquérir une expérience extrêmement valorisante d'un point de vue personnel.

Dans la mesure où il reflète parfaitement le domaine dans lequel j'aimerais poursuivre mes études, j'estime être heureux d'avoir pu effectuer ce stage entouré de personnes compétentes qui ont su me guider dans mes démarches tout en me laissant une certaine autonomie.

Annexe A

Présentation de la base «îlot»

Table : îlot		
champs	type	description
idîlot	numeric	identifiant unique de l'îlot
aire	numeric	aire de l'îlot [1]
classe	varchar	classe de l'îlot [2]

Table : bâtiment		
champs	type	description
idbat	numeric	identifiant du bâtiment
aire	numeric	aire du bâtiment
idîlot	numeric	référence vers l'îlot d'appartenance

TABLE A.1 – Structure de la base «îlot»

Table : îlot		
idîlot	aire	classe
1	400	yes
2	375	yes

Table : bâtiment		
idbat	aire	idîlot
1	60	1
2	25	1
3	70	1
4	90	2
5	30	1
6	15	2
7	80	1
8	55	2

TABLE A.2 – Jeu de données de la base «îlot»

2. ce champs n'est pas important dans le projet

2. la valeur de ce champs (yes ou no) est déterminée lors du processus d'étiquetage, pour savoir si l'îlot répond ou non à une condition.

Annexe B

Planning prévisionnel (phase 1)

RÉNOVATION ERGONOMIQUE DE PROPER :			
Échéance	Description	Difficulté	Etat
11/04/11	Lancement du projet, sujet et étude de l'environnement de travail	Facile	OK
12/04/11	Analyse d'une nouvelle interface de connexion (analyse de l'existant)	Moyen	OK
13/04/11	Développement de l'interface	Moyen	OK
14/04/11	Intégration de l'interface dans «Proper»	Facile	OK
15/04/11	Tests, Correction des double boutons et commentaire du code	Facile	OK

Annexe C

Planning prévisionnel (phase 2)

ANALYSE DE L'EXISTANT ET PRÉ-TRAITEMENTS :			
Échéance	Description	Difficulté	Etat
18/04/11	Trouver comment lancer le Runner de Proper en ligne de commande et étudier le code du DataGenerator	Moyen	OK
20/04/11	Afin de comprendre la structure de l'application finale, il faut commencer à réaliser un script qui permet de repondre au sujet du stage (mais, sans interface utilisateur)	Difficile	OK
28/04/11	Comprendre et trouver le fonctionnement des classeurs WEKA (explorer) et intégrer le choix du classeur dans le script (avec paramètre variable)	Moyen	OK
28/04/11	Première maquette graphique de l'application sous Netbeans	Facile	OK
29/04/11	Rendu du cahier des charges de l'application	Moyen	OK
29/04/11	Finalisation du script Java permettant l'automatisation des tests	Difficile	OK
01/05/11	Intégration dans le script des autres algorithmes comme RELAGGS et Discretizor	Moyen	OK
DÉVELOPPEMENT ET INTÉGRATION :			
Échéance	Description	Difficulté	Etat
03/05/11	Développement de l'interface graphique (sans événement/action) sous Eclipse	Facile	OK
05/05/11	Intégration de l'algorithme de Cardinalisation	Moyen	OK
09/05/11	Intégration de l'algorithme de RELAGGS	Moyen	OK
10/05/11	Présentation du prototype	Facile	OK
13/05/11	Intégration de l'algorithme de Discretisation	Moyen	OK
FINALISATION			
Échéance	Description	Difficulté	Etat
16/05/11	Tests et débogage	Facile	OK
17/05/11	Commenter le code et génération de la JavaDoc	Moyen	OK
25/05/11	Amélioration des fonctionnalitées	Moyen	OK
30/05/11	Rendu du projet	Facile	EN COURS
30/05/11	Article d'aide à l'utilisation (ang/fr)	Moyen	-

Annexe D

UML des classes du projet

FIGURE D.1 – Aperçu des différents packages et classes du projets (attributs et methodes absentes)

Annexe E

Espace de travail

Ce schéma nous informe des **différentes fichiers et données résultats** que l'on retrouvent dans notre espace de travail après le passage du test :

FIGURE E.1 – Project workspace

Annexe F

Pourquoi automatiser les tests ?

Ci-dessous, la courbe résultat représentant le nombre de nœud de l'arbre de décision généré par WEKA en fonction du nombre d'îlots pour les quatre algorithmes. Nous fessons varier le nombre d'instances d'îlots de 50 à 500 avec un pas de 50 instances (total de 10 mesures par algorithmes).

Selon, le processus de test pour chaque point, nous avons 4 manipulations :
- La génération des données avec DataGenerator
- La propositionnalisation avec Proper
- L'interprétation avec Weka
- L'enregistrement des résultats

Avec 10 mesures par algorithme, nous devons déjà faire 160 manipulations. Afin d'affiner les courbes, nous répétons le test 10 fois, donc au final, nous devons effectuer 1600 manipulations ! C'est inconcevable à la main !

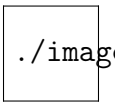


FIGURE F.1 – Calcul du nombre de manipulation réalisées lors d'un test

Glossaire

Fichier ARFF
Ce fichier est le résultat de l'extraction d'une base de données sous forme de fichier par Proper.

Fichier XML
Un fichier XML est en quelque sorte un langage HTML amélioré permettant de définir de nouvelles balises. Il s'agit effectivement d'un langage permettant de mettre en forme des documents grâce à des balises. Dans notre cas, le fichier XML contient les informations et les paramétres de l'algorithme à lancer.

Fichier CSV
C'est un fichier représentant des données tabulaires sous forme de « valeurs séparées par des virgules ». Il peut être lu directement par Excel ou OpenOffice Calc. Dans notre application, ce fichier contient les résultats des tests.

Propositionalisation des données
La propositionalisation est un concept important de fouille de données. L'objectif est de passer de plusieurs tables relationnelles à une seule table en remontant en profondeur (association un-à-plusieurs)

Discrétisation des données
La discrétisation consiste à découper une variable quantitative en intervalles. Il s'agit d'une opération de recodage.

Étiquetage des données
Cela permet de donner une valeur (par défaut, vrai ou faux) à une ligne de la table, si celle-ci respecte une condition de filtrage.

Classeur WEKA
Un classeur est un outil permettant de construire un arbre de décision et ainsi prédire la classe d'une donnée.

Arbre de décision
Un arbre de décision est un outil d'aide à la décision qui représente la situation plus ou moins complexe à laquelle on doit faire face sous la forme graphique d'un arbre de façon à faire apparaître l'extrémité de chaque branche les différents résultats possibles en fonction des décisions prises à chaque étape.

Réf. rapport :

Etudiant : HAEHNEL Jonathan

Entreprise : Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection

Mots clés de l'application :

- Fouille de données
- Propositionnalisation
- Tests
- Algorithmes
- Proper
- Processus de test
- Cardinalisation
- Discretisation

Matériel utilisé :

- Java 6
- Libraries JfreeChart, JDBC
- Composants Swing
- Bases de données Postgresql-Mysql
- LaTeX / Beamer

Logiciels utilisés *(comme support à l'analyse et/ou au développement, y compris les langages)* :

- Eclipse
- Proper, DataGenerator et Weka
- PgAdminIII
- Kile
- Dia (pour UML)

Enoncé du sujet :Réalisation des tests de certains algorithmes de fouille de données

Résumé :

J'ai effectué mon stage au Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection dans l'équipe de Fouille de données et classification du 11 avril au 17 juin 2011.

Durant ce stage, j'ai du réaliser des séries de tests permettant de caractériser certains algorithmes de fouille de données. Il fallait en particulier dégager les bénéfices des deux algorithmes (la Cardinalisation et les Quantiles) conçus et implémentés par le laboratoire par rapport aux autres. Autour du projet, l'existant était conséquent et il a demandé une grosse analyse. Nous nous sommes vite rendu compte que faire les tests manuellement était très long et peu efficace, c'est pourquoi il fut important de les automatiser avec une application en Java.

Outre la réalisation de cette application, j'ai du modifier l'interface de connexion du logiciel "Proper" qui était très peu ergonomique et trop répétitive.

Bonne lecture !