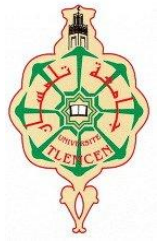
République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique



IRISA
Institut de recherche
en informatique et systèmes aléatoires

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Modèle Intelligent et Décision(M.I.D)*

Thème

Méthodes d'apprentissage pour améliorer la QoS d'une flotte de logiciels embarqués

Réalisé par :

Somia RAHMOUN

Présenté le 15 Septembre 2011 devant la commission composée de :

Président : - *Mohammed LEHSAINI*

Encadreurs : - *Abdelkrim BENAMAR*

- *Marie-Odile CORDIER*

Examinateurs : - *Nawel ILES* - *Mourtada BENAZOUZ*

- *Yaghmoracen BENZIANE* - *Fethellah HADJILA*

- *Smail SMAHI*

Année universitaire : 2010-2011

Je dédie ce modeste travail

A mes chers parents

En témoignage de ma profonde gratitude et de mon incontestable reconnaissance, pour tous les sacrifices qu'ils me contentent, toute la confiance qu'ils m'accordent et tout l'amour dont ils m'entourent.

A mes chers frères et sœur

Mohamed, Hatem et Manal que je ne trouverais jamais assez de mots pour leurs exprimer mon amour, En leurs espérant le plein succès.

A la famille

Mes grands-parents paternels et maternels, mes chères tantes et oncles à qui je souhaite le parfait bonheur

Aux amis

Chihab que je remercie pour son soutien et ses encouragements, à ma très chère amie Wisseme, à tous les collègues du Master MID, et à tous les amis de l'université de Tlemcen. Aux amis de Rennes, à mon orchestre Ahbab Cheikh Larbi Ben Sari et à tous ceux qui me sont chers.

Remerciements

Ce travail a été effectué au sein de l'équipe DREAM (Diagnostic, REcommandation d'Actions et Modélisation) du laboratoire IRISA (Institut de recherche en informatique et systèmes aléatoires) de l'université de Rennes1, France.

Je tiens tout d'abord à remercier le Professeur Marie-Odile Cordier de m'avoir accueilli au sein de son équipe de recherche DREAM et de m'avoir guidé tout le long du stage. Je remercie également les Professeurs Bernard Philippe et Kadi Bouatouch pour leurs recommandations. Je souhaite aussi remercier le doyen de la faculté des sciences Mr Befeldja Tabti, ainsi que le vise doyen Mr Said Ghalem et toutes personnes ayant participé à la réalisation de ce stage.

Je remercie mon encadrant et chef de département d'informatique, Mr Abdelkrim Benamar ainsi que mes deux responsables de stage Laurence Roze et Sophie Robin pour leurs disponibilités et la qualité de leurs encadrements.

Je tiens à remercier également tous les membres de la commission d'examination des projets de fin d'étude du Master MID.

Je remercie mon encadrement à l'Université de Tlemcen ainsi que l'ensemble de mes professeurs qui m'ont permis d'en arriver ici aujourd'hui.

Je remercie toute l'équipe DREAM du laboratoire IRISA pour leurs sympathie. Je pense aussi aux nombreux autres stagiaires et doctorants que j'ai rencontré pendant cette période et qui sont devenus des amis. Merci à Billal Merabti pour son aide et ses orientations.

Le plus grand remerciement revient à mes parents, mes deux frères et ma petite sœur qui m'ont encouragé en cette période de stage et tout le long de mes études.

Enfin, j'ai une pensée spéciale pour Alain et Cecile Mignot qui m'ont accompagné pendant ces quatre mois.

Somia RAHMOUN

Table des matières

Liste des figures

Liste des algorithmes

Liste des exemples

Introduction générale

Introduction

Actuellement avec le progrès de la technologie et des innovations, principalement dans le domaine de la communication, les téléphones portables sont de plus en plus répandus. Le développement de ces derniers vers l'aspect intelligent donne naissances aux Smartphones. En effet un public de plus en plus considérable est intéressé par ces appareils qui offrent diverses applications, utiles aussi bien dans le domaine professionnel que dans la vie courante. Comme toutes technologies ces derniers ne sont pas exempts de problèmes techniques, leur aspect multitâche, capable aussi bien de recevoir des appels que d'exécuter des taches diverses, pose des problèmes de stabilité et de fiabilité [1]. Il semblerait alors nécessaire d'effectuer des diagnostiques sur ces appareils, dans le but de comprendre ces erreurs de plantage qui peuvent être rencontrés lors de la manipulation de ces Smartphones, de les corriger et de prévenir de éventuelles situations de plantage. C'est en quoi consiste le projet « *ManageYourself* ».

Présentation du projet *« ManageYourself »*

Manage Yourself est un projet de diagnostic et surveillance de plates-formes embarquées[1] [2]. Le projet a réunis les deux organismes TELELOGOS et l'équipe de recherche DREAM-IRISA de l'université de Rennes 1, dans le cadre d'une collaboration. TELELOGOS est une entreprise éditant des logiciels professionnels qui automatisent, administrent et optimisent les processus d'échanges entre système informatique central et ceux des utilisateurs distants. DREAM (Diagnostic, recommandation d'actions et modélisation) est une équipe de recherche de l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires) spécialisée dans l'aide à la surveillance et au diagnostic de systèmes évoluant dans le temps. Le besoin de développer une telle application provient de l'entreprise TELELOGOS.

[1] On désigne par le terme embarqué tout système fonctionnant à l'intérieur d'un équipement n'ayant pas une vocation purement informatique

Le but du projet *Manage Yourself* est de développer une plateforme de surveillance sur des appareils de type Smartphone et PDA (Personal Digital Assistant). L'application de surveillance sera embarquée sur l'appareil mobile. Elle surveillera le système et sera constamment au courant de son état et sera capable de détecter les situations problématiques de plantage et de les corriger. L'application saura reconnaitre et réagir à une situation problématique grâce à un ensemble de règles[2]. Ces règles auront été au préalable téléchargées sur un serveur distant. D'autre part si l'application de surveillance n'a pas réussi à empêcher la défaillance de l'appareil, celle-ci envoi au serveur un bilan de la situation de l'appareil juste avant le problème encouru. L'objectif étant, à terme, de pouvoir apprendre de ces données remontées sur le serveur afin de générer de nouvelles règles permettant de traiter les problèmes ayant provoqués des erreurs.

Un premier logiciel a été développé par un groupe d'étudiants de l'INSA (Institut National des Sciences Appliquées) de Rennes lors de l'année 2009-2010. Le projet finalisé présentait certaines lacunes, d'où le besoin de développer un deuxième logiciel qui serait plus performant. Le projet *Manage Yourself* 2010-2011dans lequel nous intervenons, est une optimisation de l'ancien. Notre travail de mémoire concerne la partie serveur du projet, la conception et la réalisation de l'application embarquée sont assurées par des étudiants de l'INSA.

Le mémoire se divise en quatre chapitres: dans le premier chapitre nous présentons le projet *ManageYourself* 2009-2010 et nous analysons son mécanisme dans le but de déterminer les problèmes de fonctionnement du système. Le deuxième chapitre est une étude bibliographique sur deux grandes méthodes de l'apprentissage supervisé ; l'apprentissage par arbre de décision, et l'apprentissage par règles de classification. Une étude comparative entre les deux approches clôture le chapitre. Le troisième chapitre présente l'architecture du serveur pour le projet 2010-2011 que nous avons mis en œuvre après étude et amélioration de l'ancienne version (*ManageYourself* 2009-2010). Le fonctionnement global y est détaillé. Le dernier

[2] Un exemple de règles serait « Si il n'y a pas assez de mémoire alors supprimer les fichiers temporaires ».

chapitre présente l'analyse des tests effectués sur le système proposé et la discussion des différents résultats obtenus.

Chapitre 1

Projet Manage Yourself

I. Projet ManageYourself

Introduction

Le projet *Manage Yourself* est un projet de diagnostic et surveillance de plates-formes embarquées. Le but est de faire de la prévention de pannes sur des appareils de type Smartphone et PDA (Personal Digital Assistant). Un premier logiciel a été développé lors de l'année universitaire 2009-2010 par des étudiants de l'INSA (Institut National des Sciences Appliquées) de Rennes. Sur ce logiciel, l'apprentissage au niveau du serveur se faisait grâce à l'utilisation d'arbres de décision. Le but de notre travail est d'apporter des améliorations sur cette partie du projet. Nous présentons dans ce chapitre, le projet *Manage Yourself* 2009-2010, et nous expliquons le fonctionnement global du serveur dans le but de comprendre et analyser ce qui a été fait afin de proposer des améliorations pour le projet *Manage Yourself* 2010-2011.

I.1. Architecture globale

Le système est constitué de deux grandes parties : d'une part la partie embarqué qui fonctionne sur le Smartphone, et d'autre part la partie serveur qui est sur un ordinateur distant. Le système embarqué est une application de surveillance qui est constamment au courant de l'état du Smartphone ; celle-ci détecte les situations problématiques et doit être capable d'y remédier grâce à des règles qui ont été au préalable téléchargés depuis le serveur. Dans le cas où l'application de surveillance n'arrive pas à détecter et donc à empêcher un dysfonctionnement, elle devra envoyer le rapport de la situation juste avant le plantage. Le serveur reçoit ces rapports dans le but de faire de l'apprentissage de règles afin de détecter les situations problématiques. Les solutions apportées constituent la base de règles

correctives[3] qui sera ensuite renvoyé au Smartphone. L'architecture du projet est présentée dans la Figure I-1 Architecture globale

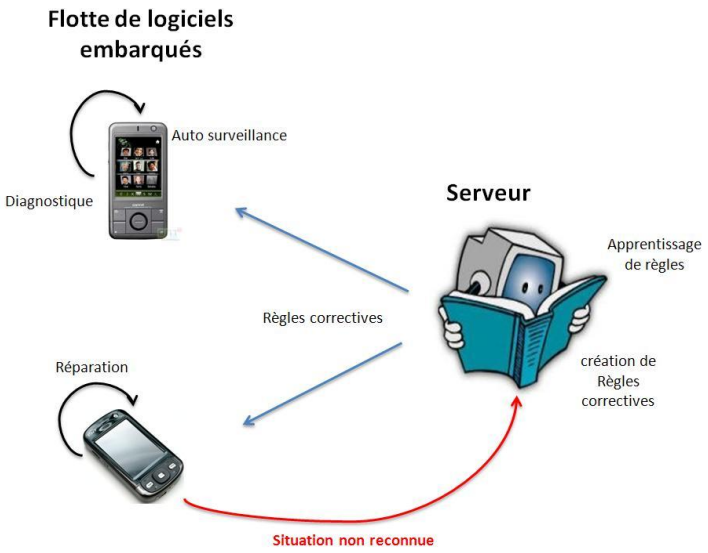


Figure I-1 Architecture globale

Les deux parties communiquent par le logiciel Mediacontact [3] développé par l'entreprise partenaire : TELELOGOS.

I.2. Partie embarquée

Le matériel mobile est doté de moyens d'autosurveillance afin de pronostiquer un dysfonctionnement et d'y remédier localement pour maintenir au mieux la qualité de service attendue. L'architecture de la partie embarquée est présentée dans la Figure I-2. Le module de surveillance surveille le système de manière périodique [2], en générant des rapports d'état du système contenant des informations telles que la place mémoire disponible, les applications lancées…, ces rapport sont envoyés au module de diagnostic. Le module de diagnostic possède des règles qui ont été téléchargées au préalable depuis le serveur ; ces règles vont permettre de déterminer si l'appareil est dans une situation problématique et d'effectuer les actions nécessaires afin d'éviter le plantage de l'appareil. Si toutefois l'appareil rencontre un

[3] Les règles correctives sont des règles aux quelle des actions ont été ajoutées :
Règle apprise : Si mémoire >200 ALORS Plantage = oui
Règle correctives : Si mémoire >200 ALORS ViderFichierTemporaire ;
ViderFichierTemporaire est une action

problème pour lequel aucune règle n'a été reconnue, il envoie au serveur les derniers rapports d'état du système dans le but d'apprendre à partir de ces rapports des règles de plantage et d'en associer des actions correctives, pour éviter par la suite le problème rencontré.

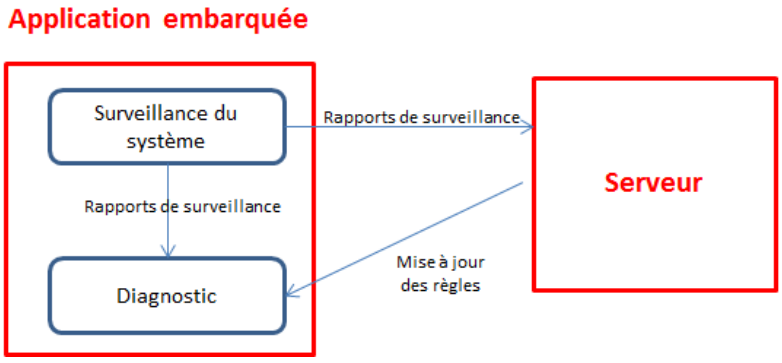


Figure I-2 Architecture du système embarqué

I.3. Partie serveur

L'application présente sur le serveur permet de recevoir les rapports des appareils ayant rencontré une défaillance. Le module d'apprentissage est capable de déduire les causes de défaillances à partir de ces rapports. Enfin, elle fournit une interface administrateur permettant d'associer des actions aux règles apprises et de les ajouter à la base de règles existante.

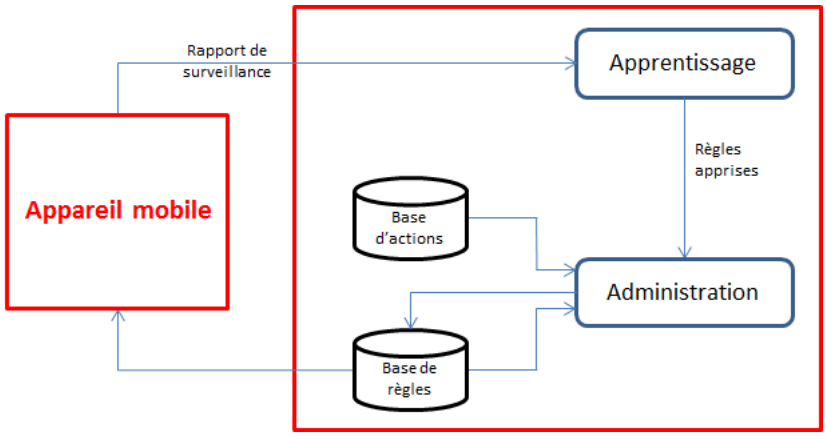


Figure I-3 Architecture serveur du projet Manage Yourself 2009-2010

I.3.2. Apprentissage

L'apprentissage est la partie du projet « *ManageYourself* » qui permet l'évolution du système, en terme de règles correctives par la découverte de nouvelles

règles (apprentissage). A partir des rapports reçus contenant la situation du Smartphone, de nouvelles règles de plantage seront apprises qui permettront à l'appareil mobile dans un temps futur de prévenir les situations problématiques correspondantes. Pour pouvoir mettre en place et développer la partie apprentissage du système, un générateur d'exemples a été mise en place dans le but de simuler les situations problématiques que peut rencontrer l'appareil mobile, et donc de générer des rapports susceptibles d'être envoyés par le Smartphone. Pour développer cette partie, la bibliothèque Weka a été utilisée. Weka implémente la plupart des algorithmes d'apprentissage existants.

I.3.2.1. Générateur d'exemples

Le simulateur a permis aux étudiants de l'INSA de tester l'algorithme d'apprentissage choisi, et d'analyser les résultats. Il prend en entrée des règles et génère une quantité d'exemples voulue en se rapprochant d'un cas réel d'utilisation. Les composantes de l'appareil mobile surveillées sont: la mémoire du téléphone, les différentes applications installées sur le Smartphone: application-A, application-B, et application-C qui peuvent être lancées ou non, dont on sait si on possède les dernières versions ou non, et enfin l'état de la batterie de l'appareil mobile. Plusieurs situations ont été simulées que nous détaillerons plus loin.

a) Les attributs

Les attributs sont définis par un nom et un domaine de valeurs. Le domaine peut être de deux types : discret ou numérique [4].

```
@attribute memoire {normale,saturee,presqueSaturee}
@attribute applicationA {lancee,nonLancee}
@attribute applicationB {lancee,nonLancee}
@attribute applicationC {lancee,nonLancee}
@attribute derniereVersionA {oui,non}
@attribute derniereVersionB {oui,non}
@attribute dernieVersionC {oui,non}
@attribute batterie {normale,faible,vide}
@attribute plantage {oui,non}
```

b) Les règles de simulation

Les règles que prend le générateur d'exemples en entrée sont de la forme : Condition ⇒ plantage =oui. Les conditions d'une règle peuvent être : des tests d'égalité ou d'inégalité sur une valeur d'attribut ou des tests d'appartenance à un

domaine, qu'il soit discret ou continu. À partir de ces règles fixées par les concepteurs du simulateur (les étudiants de l'INSA), celui-ci va tirer des exemples aléatoirement, sans aucune contrainte sur les valeurs des attributs [4].

c) Les exemples

Les exemples générés sont stockés dans un fichier au format .arff[4]. Ces rapports permettent de faire de l'apprentissage sous Weka en utilisant l'algorithme choisi. Voici un exemple de rapport simulé:

Exemple I-1 *Rapport simulé par le générateur d'exemples*

```
@relation smartphone1
@attribute memoire {normale,saturee,presqueSaturee}
@attribute applicationA {lancee,nonLancee}
@attribute applicationB {lancee,nonLancee}
@attribute applicationC {lancee,nonLancee}
@attribute derniereVersionA {oui,non}
@attribute dernieereVersionB {oui,non}
@attribute dernieereVersionC {oui,non}
@attribute batterie {normale,faible,vide}
@attribute plantage {oui,non}
@data
normale,lancee,lancee,nonLancee,oui,oui,non,faible,oui
presqueSaturee,lancee,nonLancee,nonLancee,oui,non,oui,normale,non
presqueSaturee,lancee,lancee,nonLancee,oui,non,non,normale,oui
presqueSaturee,lancee,lancee,lancee,oui,oui,non,vide,oui
saturee,nonlancee,lancee,Lancee,non,oui,oui,faible,non
presqueSaturee,nonlancee,nonlancee,Lancee,non,oui,oui,normale,non
normale,lancee,nonlancee,lancee,oui,non,oui,normale,oui
presqueSaturee,lancee,lancee,lancee,non,oui,non,vide,non
```

Sur la première ligne du fichier généré, on y trouve le nom du concept à apprendre (smartphone1) puis dans la suite, la description de chacun des attributs, sous la forme nom, domaine, et enfin les exemples avec la valeur des attributs dans l'ordre de leur déclaration.

d) Les différentes situations simulées

Dans un premier temps le simulateur utilise uniquement des données discrètes (Smartphone1), ensuite avec des données numériques (Smartphone2) ce qui rend les règles plus compliquées, puis avec réduction sensible du nombre de plantages (Smartphone 3). Ensuite pour le Smartphone 4, on introduit des données bruitées ne

[4] Format utilisé par la bibliothèque Weka

respectant pas les règles du Smartphone, et enfin avec des règles indéterministes où le Smartphone peut planter ou non (Smartphone5).

Smartphone 1

Le simulateur génère des rapports d'exemples pour des données discrètes à partir des règles suivantes :

memoire = saturee ∧ applicationC=lancee ⇒ plantage=oui
memoire = presqueSaturee ∧ applicationC=lancee ⇒ plantage=oui
memoire = saturee ∧ applicationA=lancee ⇒ plantage=oui
applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

La Figure I-4 montre une représentation de la distribution de 100 exemples, générés par le simulateur à partir des règles précédentes:

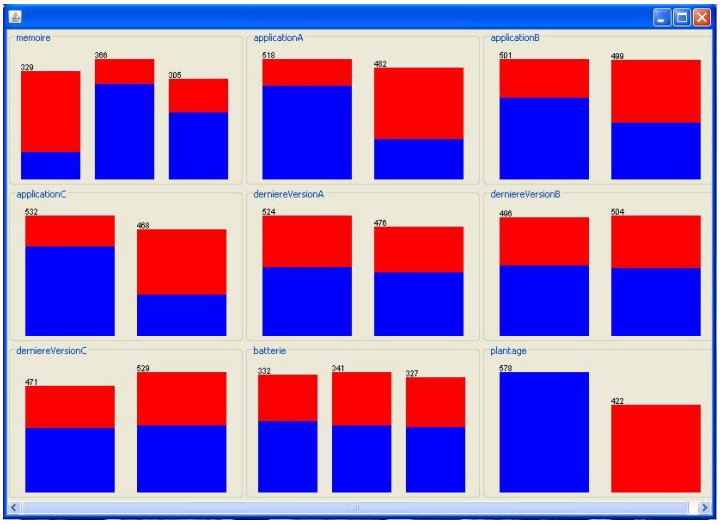


Figure I-4 Répartition des 100 exemples testés sur le Smartphone 1

Une fenêtre pour chaque attribut, en bleu le nombre de plantages, en rouge le nombre de non plantage. Pour la mémoire, par exemple, dont le domaine de valeurs est {normal, saturée, presque saturée}, nous pouvons voir qu'il y a 329 exemples avec mémoire normale et nous remarquons ici que le nombre d'exemples de non plantage (en rouge) est supérieur à celui des exemples avec plantage (en bleu), 366 exemples avec mémoire saturée et 305 exemples pour mémoire presque saturée.

Smartphone 2

L'idée du deuxième simulateur est de traiter la mémoire non plus comme une valeur discrète mais comme une donnée numérique: mémoire ∈ [0, 256] :

memoire ∈ [200, 256] ∧ applicationC=lancee ⇒ plantage=oui
memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui
applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

Smartphone 3

En analysant les deux premières situations simulées, les concepteurs se sont aperçus qu'il y a beaucoup plus d'exemples de plantage que dans la réalité. Dans cette troisième situation simulée, les exemples ne vont plus être tirés complètement aléatoirement. Partant du fait que beaucoup de personnes ne lancent pas plusieurs applications en même temps, les concepteurs ont considéré que lorsque qu'une application est lancée les deux autres n'ont plus qu'une chance sur dix d'être lancées [4].

memoire ∈ [200, 256] ∧ applicationC=lancee ⇒ plantage=oui
memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui
applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

Les règles sont identiques à celle du Smartphone 2, la différence se trouve au niveau du tirage des exemples.

Smartphone 4

Des données bruitées sont introduites dans les rapports, ne respectant pas les règles du simulateur, l'idée est de simuler une panne matérielle c'est-à-dire des plantages de l'appareil qui ne rentrent dans aucune des règles. Un exemple sur 100, étiqueté « plantage = non » est transformé en « plantage = oui ».

memoire ∈ [200, 256] ∧ applicationC=lancee ⇒ plantage=oui
memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui
applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

Smartphone 5:

L'idée est ici et de simuler des situations où le Smartphone peut planter ou non en introduisant une nouvelle règle : lorsque la mémoire est entre 200 et 240 et que l'application C est lancée, le Smartphone à une chance sur deux de se planter.

memoire ∈ [240, 256] ∧ applicationC=lancee ⇒ plantage=oui
memoire ∈ [200, 239] ∧ applicationC=lancee ⇒ plantage=ouiounon
memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui

applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

Les règles du simulateur sont légèrement différentes. Dans le cas où la mémoire est comprise entre 200 et 240 et où l'application C est lancée, le Smartphone va pouvoir planter ou non avec la même probabilité [4].

I.3.2.2. Algorithme d'apprentissage

L'apprentissage des règles de plantages à partir des données contenues dans les rapports simulés, est un apprentissage supervisé. L'algorithme qui a été utilisé pour le projet *Manage Yourself* 2009-2010 par les étudiants de l'INSA pour apprendre des arbres de décisions, est l'algorithme C4.5 de Quinlan [5] implémenté dans Weka sous le nom de J48 [6]. L'apprentissage se fait systématiquement à partir de tous les exemples simulés. Pour un rapport à 100 exemples simulés, l'arbre de décision généré lors de l'apprentissage des règles pour le concept « *Smartphone 1* » serait :

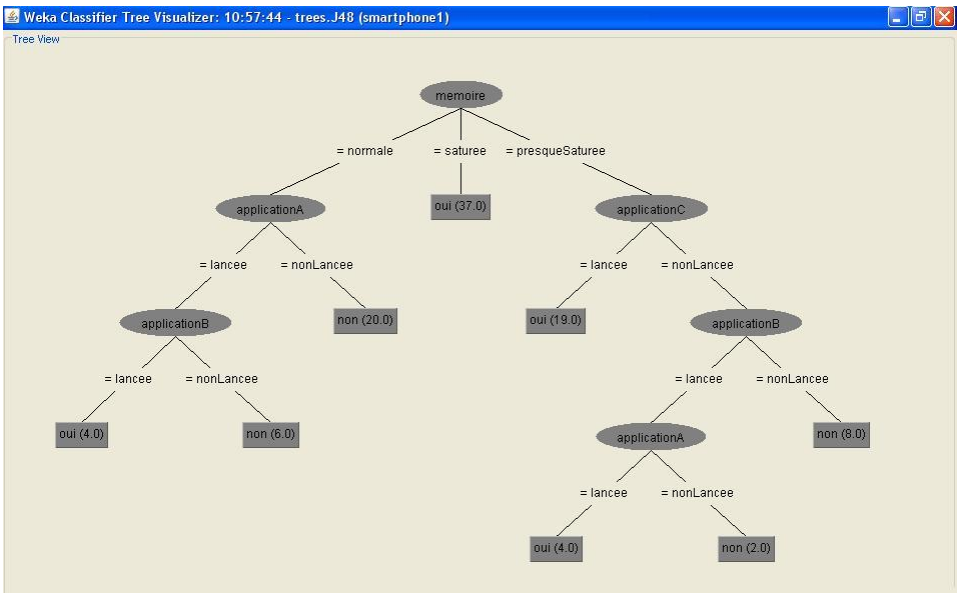


Figure I-5 Arbre de décision

Comme le montre l'arbre de la Figure I-5, les règles déduites sont les suivantes :

Ra_1 memoire = normale ∧ applicationA = lancee ∧ applicationB = lancee ⇒ plantage= oui

Ra_2 memoire = normale ∧ applicationA = lancee ∧ applicationB = nonLancee ⇒ plantage = non

Ra_3 memoire = normale ∧ applicationA = nonLancee ⇒ plantage = non

Ra_4 memoire = saturee ⇒ plantage =oui
Ra_5 memoire = presqueSaturee ∧ applicationC = lancee ⇒ plantage = oui
Ra_6 memoire = presqueSaturee ∧ applicationC = nonLancee ∧ applicationB = lancee ∧ applicationA = lancee ⇒ plantage = oui
Ra_7 memoire = presqueSaturee ∧ applicationC = nonLancee ∧ applicationB = lancee ∧ applicationA = nonLancee ⇒ plantage = non
Ra_8 memoire = presqueSaturee ∧ applicationC = nonLancee ∧ applicationB = nonLancee ⇒ plantage = non

Nous remarquons que l'algorithme d'apprentissage apprend 4 règles de plantage et 4 règles de non plantage, pour notre système qui fait la prévention de pannes, les règles de non plantage ne nous intéressent pas. Regardons de plus près les règles de plantage. En les comparants aux règles de simulation du « *Smartphone 1* » qui sont :

Rs_1 memoire = saturee ∧ applicationC=lancee ⇒ plantage=oui
Rs_2 memoire = presqueSaturee ∧ applicationC=lancee ⇒ plantage=oui
Rs_3 memoire = saturee ∧ applicationA=lancee ⇒ plantage=oui

Nous remarquons que la règle apprise (Ra_6) est une règle du simulateur (Rs_2) cela montre l'efficacité de l'algorithme à apprendre une situation simulée, mais la règle apprise (Ra_4) généralisant les deux règles du simulateur (Rs_1) et (Rs_3) prouve le contraire, en effet la règle est trop générale, en lui associant une action corrective, celle-ci risque d'être déclenché très souvent ce qui n'est pas très confortable pour l'utilisateur. Les deux autres règles apprises (Ra_1) et (Ra_6) qui n'ont aucune relation avec les règles du simulateur, prouvent l'efficacité de l'algorithme à apprendre de nouvelles situations de plantages. En leurs associant des actions de correction par la suite, celles-ci seront capables de prévenir ces situations problématiques.

I.3.3. Administration

Une interface administration est présente sur le serveur. Sur cette partie l'intervention d'un expert humain est obligatoire pour la manipulation des règles correctives. Le but de cette partie est d'attribuer une action corrective pour chaque règle de plantage apprise. Par exemple pour la règle Si mémoire > 200 ALORS plantage = oui, une action telle que ViderFichierTemporaire pourrait être affectée, ce qui donne naissance à une règle corrective qui serait: Si mémoire > 200 ALORS ViderFichierTemporaire. L'interface permet de visualiser facilement toutes les règles correctives présentes sur le système expert, elle permet également de supprimer des règles et d'en saisir de nouvelles.

I.3.4. Déroulement

Au lancement du générateur d'exemples (simulateur), celui-ci prend en entrée un fichier contenant les règles de simulation, cités précédemment pour les différentes situations simulées, le simulateur génère un fichier d'exemples (fichier .arff) qui est considéré jusqu'ici comme le rapport envoyé par l'appareil mobile, L'apprentissage se fait alors à partir de ce rapport, en récupérant le fichier généré par le simulateur et en appliquant l'algorithme d'apprentissage sous Weka, les règles apprises seront après étiquetés (affectation d'action correctives) via l'interface administration par un expert, et donc un fichier de règles correctives sera généré qui sera par la suite envoyé à l'appareil mobile pour y être utilisé, comme le montre la Figure I-6.

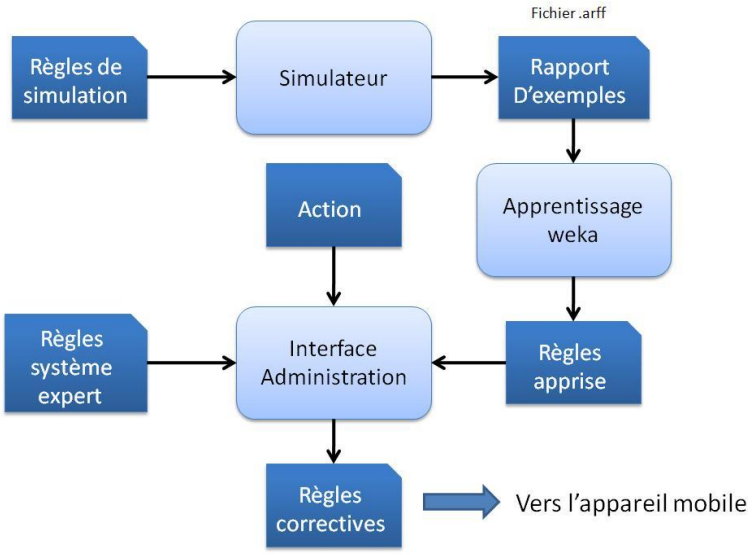


Figure I-6 Déroulement interne du serveur Manage Yourself 2009-2010

I.4. Constatations et problématique

Dans le projet *Manage Yourself* 2009-2010, lors des tests, l'apprentissage se faisait de manière naïve ; cela consistait à récupérer les rapports d'exemples simulés et à appliquer dessus l'algorithme d'apprentissage sans prendre en considération les règles correctives déjà déduites. Dans une situation réelle, dès qu'une règle corrective est installée sur l'appareil mobile, les exemples couverts par celle-ci ne risquent plus de figurer sur les rapports envoyés par l'application mobile. Nous proposons alors d'intégrer un moyen de filtrage qui simule l'exécution des règles correctives au niveau du Smartphone.

En analysant les règles apprises produites par l'algorithme d'apprentissage utilisé (C4.5), nous avons constaté que celui-ci apprend non seulement les règles de plantage mais aussi les règles de non plantage, notons que ce qui nous intéresse pour mettre en place un système de prévention de pannes est beaucoup plus les cas de plantages. Nous souhaitons aussi faire un apprentissage incrémentale, pour prendre en charge l'aspect évolutif du système. Pour cela nous avons pensé changer l'algorithme d'apprentissage pour un autre qui serait plus adapté à nos attentes. Nous allons dans le chapitre suivant étudier les deux approches de l'apprentissage supervisé ; l'apprentissage par arbres de décision et celui par règles de classification.

Dans la partie administration, la seule partie du système non automatique où l'intervention d'expert humain est obligatoire, le choix des règles correctives à envoyer à l'appareil mobile se fait de manière manuelle et sans fixer aucune contrainte de sélection, ce qui est une tache lourde pour l'administrateur demandant beaucoup de temps d'analyse et de réflexion, pour cela nous proposons d'introduire un moyen d'aide à la décision qui va aider l'expert humain à sélectionner les meilleures règles correctives à embarquer. Nous l'appelons ; *Module de sélection*, ce module proposera des mises à jour des connaissances embarquées à partir des nouvelles règles apprises, en fixant des contraintes sur ces deux ensembles.

Conclusion

Nous avons présenté dans ce chapitre, le projet *Manage Yourself* 2009-2010 implémenté par les étudiants de l'INSA et supervisé par Laurence Rozé, enseignante à l'INSA et chercheur à l'IRISA. Voir et comprendre le déroulement du projet, nous a permis de faire des constatations que nous voudrions prendre en considération pour d'éventuelles améliorations, et donc de proposer de nouvelles méthodes pour le projet *Manage Yourself* 2010-2011 dans lequel nous intervenons.

Chapitre 2

Apprentissage automatique

II. Apprentissage automatique

Introduction

Discipline et domaine vaste de l'intelligence artificielle, l'apprentissage automatique fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'évoluer grâce à un processus d'apprentissage [5]. L'apprentissage automatique consiste à tirer des règles générales à partir d'observations particulières. Les algorithmes d'apprentissage peuvent se catégoriser selon le type d'apprentissage qu'ils emploient : si les classes sont prédéterminées et les exemples étiquetés; on parle alors d'apprentissage supervisé. Quand le système ou l'opérateur ne disposent que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé [7]. Et enfin l'apprentissage semi-supervisé qui vise à faire apparaître la distribution sous-jacente des exemples dans leur espace de description. Il est mis en œuvre quand des données (ou étiquettes) manquent, le modèle doit utiliser des exemples non-étiquetés pouvant néanmoins renseigner. Plusieurs méthodes d'apprentissage automatique existent. L'utilisation de tel ou tel algorithme dépend fortement de la tâche à résoudre (classification, estimation de valeurs, etc.), et pour chaque tâche il existe toute une gamme d'algorithmes. Pour un problème d'apprentissage automatique il est souvent difficile de choisir la méthode à utiliser.

Nous allons dans ce chapitre poser la problématique liée à notre système dans le but d'exposer les besoins qui nous aideront à choisir l'algorithme d'apprentissage supervisé qui correspondra le mieux à nos attentes. Nous présenterons par la suite deux grandes approches de l'apprentissage artificiel ; les arbres de décisions qui ont été utilisés pour le projet ManageYourself 2009-2010, et les règles de classification. La dernière partie du chapitre consistera à faire une étude comparative entre les deux méthodes d'apprentissage présentées, ceci en testant les deux algorithmes sur les rapports d'exemples que nous utiliserons plus tard pour notre système.

II.1. Problématique d'apprentissage

Après avoir présenté et étudié le travail des étudiants de l'INSA pour le projet *ManageYourself* 2009-2010, nous avons fait des constatations (voir I.4. Constatations et problématique) au niveau du fonctionnement, que nous voudrions prendre en charge dans le but d'éventuelles améliorations. Nous avons observé au niveau du module d'apprentissage qui se fait par arbres de décision, que celui-ci se fait de manière très naïve ; cela consistait à récupérer les rapports d'exemples simulés et à appliquer dessus l'algorithme d'apprentissage sans prendre en considération les règles apprises pour les quelles des actions correctives ont déjà étaient attribuées, et cela empêchait le système d'évoluer. Pour cela nous avons pensé à changer l'algorithme d'apprentissage. Ce qui nous intéresse, est de faire un apprentissage incrémentale, ceci pour prendre en charge l'aspect évolutif de notre système la référence des algorithmes pour l'apprentissage incrémentale sont les algorithmes d'apprentissage par règles de classification, mais est-ce que le fait d'adopter cette méthode d'apprentissage nous fournirait un résultat meilleur. Nous allons étudier de plus près les deux approches ; arbres de décisions et règles de classification, nous les appliquons à notre système, nous comparons ensuite les résultats obtenus, pour pouvoir à la fin choisir l'approche qui nous intéresserait le plus.

II.2. Etat de l'art

II.2.1. Arbre de décision

II.2.1.1. Stratégie « Divide-and-conquer »

L'approche « divide-and-conquer » est une technique de conception d'algorithmes qui procède de façon récursive en réduisant un problème en un ensemble de sous-problèmes, jusqu'à atteindre un niveau de simplicité suffisant pour pouvoir le résoudre facilement. L'approche comporte trois étapes à chaque niveau de récursivité [5] : la première étape « *Divide »*: qui divise le problème en un ensemble de sous-problèmes différents. La seconde partie « *Conquer »* : celle-ci conquiert les sous-problèmes en les résolvant d'une manière récursive. Dans le cas où la taille des sous-problèmes est minimale, la résolution se fait d'une manière

directe. Et la dernière partie ; « *Combine* » : qui combine à la fin, les solutions des sous-problèmes obtenus afin de créer la solution au problème initial.

Exemple II-1 *l'approche « Divide-and-conquer »*

Nous expliquons l'approche par l'exemple du tri par fusion [5] qui consiste à trier une liste de chiffre et à les classer dans l'ordre croissant suivant l'approche « Divide-and-conquer » :

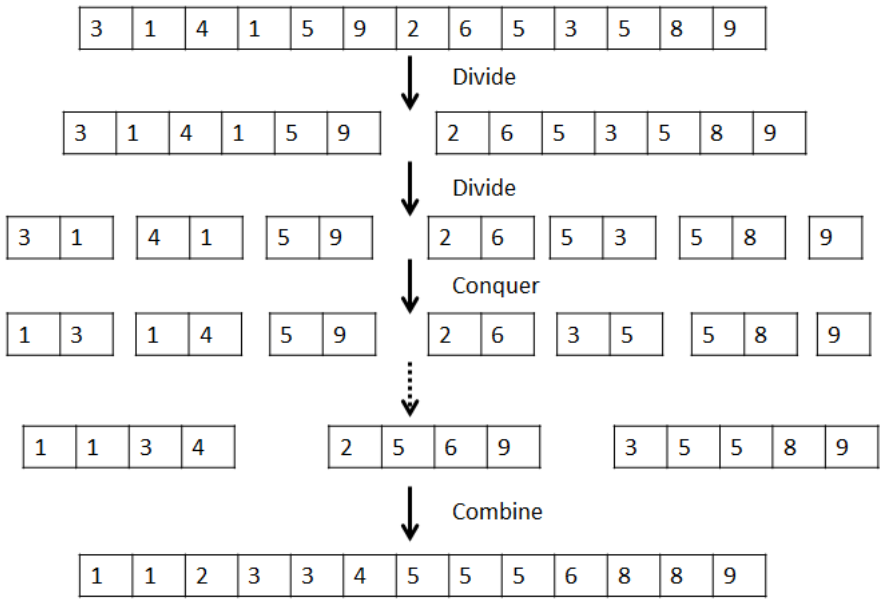


Figure II-1 Divide-and-conquer

Appliquée à l'apprentissage par arbre de décision, l'objectif général de cette approche est de générer une séquence hiérarchique de tests, aussi courte que possible, qui divise successivement l'ensemble des données d'apprentissage en sous-ensembles disjoints, tels que des sous-groupes de cas appartenant à la même classe soient rapidement détectés.

II.2.1.2. Arbres de décision

Un arbre de décision est un classifieur et outil supervisé d'aide à la décision, structuré comme son nom le suggère, tel un arbre. Un arbre de décision permet de répartir une population d'individus en groupes homogènes selon des attributs discriminants en fonction d'un objectif fixé et connu. Un arbre de décision est composé d'un nœud racine, un ensemble de nœuds internes et un ensemble de feuilles. Chaque nœud interne de l'arbre représente un attribut, les nœuds externes appelés feuilles, représentent les classes d'affectation, les arcs entre les nœuds

représentent quant à eux les tests sur ces derniers. Les nœuds internes d'un arbre de décision possèdent un seul nœud parent, et deux ou plusieurs nœuds descendants.

Exemple II-2 *Arbre de décision: Peut-on faire du sport ?*

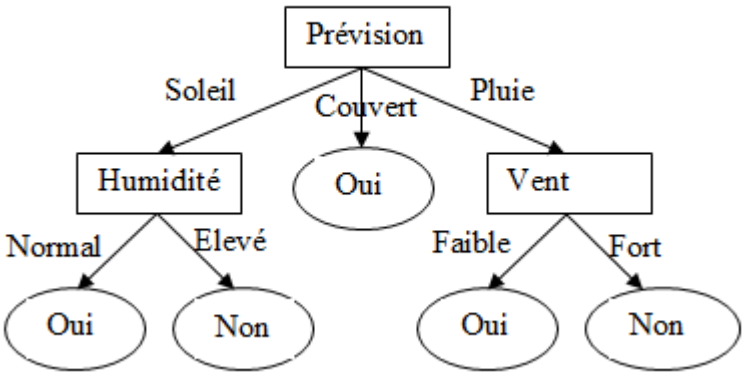


Figure II-2 Arbre de décision

Cet arbre de décision essaye de répondre à la Question : « Peut-on faire du Sport ? » [6]. Et donc la réponse à la question se trouve dans les feuilles de l'arbre, qui représentent les deux classes d'affectation : Oui et Non. L'attribut **prévision** situé sur le premier niveau, représente la racine de l'arbre, et prend ses valeurs dans le domaine {Soleil, Couvert, Pluie}. Pour chaque nouvel exemple dont on ne connaît pas l'étiquette, un parcours de l'arbre est effectué du nœud racine vers une feuille, la classe de la feuille atteinte est donc affectée à cet exemple.

II.2.1.3. Construction d'un arbre de décision

Un arbre de décision se construit de manière récursive, en divisant au fur et à mesure l'espace de données en sous-groupes de plus en plus pures en terme de classes, depuis sa racine jusqu'à ses feuilles : si tous les exemples sont dans une seule classe, on place une feuille de cette classe, sinon, on choisit l'attribut qui divise le mieux l'ensemble des données de la base d'apprentissage, une série de tests relatifs à l'attribut choisi est produite, afin de diviser l'ensemble de données (production des arcs) : pour les variables quantitatives, les tests sont généralement binaires : $X \leq$ ou $>$ seuil. Pour les variables qualitatives, chacune des valeurs possibles est considérée comme une alternative. L'objectif est de diminuer le plus possible le mélange des classes au sein de chaque sous-ensemble créé par les

différentes alternatives du test. Pour chaque nouvel ensemble, on construit un sous arbre de décision jusqu'à ce qu'un certain critère d'arrêt soit satisfait.

Condition d'arrêt

Le critère d'arrêt dépend du type de l'arbre, différentes possibilités :

- limite fixée de la profondeur de l'arbre ;
- pourcentage de cas d'appartenance à la classe majoritaire > seuil ;
- nombre de cas dans une feuille <seuil ;
- le nombre de feuilles a atteint un maximum fixé ;
- la qualité de l'arbre est suffisante ;
- la qualité de l'arbre n'augmente plus de façon sensible.

Algorithme II-1 Algorithme de construction d'un arbre de décision [8]

Entrées : - Un ensemble d'exemples E
Sorties : - Un arbre de décision

Si tous les exemples de E sont de classe c **ALORS**
    Retourner une feuille classe = c
Fin si

Choisir un attribut A
Construire une racine r représentant A
Si A est quantitatif **ALORS**
    Choisir une valeur v ∈ Domaine de A
    Soit E' les exemples qui vérifient A ≤ v
    Construire un arbre de décision pour E' et le rattacher à r
    Soit E'' les exemples qui vérifient A> v
    Construire un arbre de décision pour E'' et le rattacher à r
Sinon
    Pour chaque valeur v de Domaine de A **faire**
        Soit E' les exemples qui vérifient A=v
        Construire un arbre de décision pour E' et le rattacher à r
    Fin pour
Fin si
Retourner l'arbre de décision de racine r

Critères de séparation

Le choix de l'attribut de division et des valeurs de test, pour les variables quantitatives se fait à l'aide d'heuristiques et est souvent basé sur la théorie de

l'information, et notamment sur la notion d'entropie (mesure de l'hétérogénéité d'un mélange). Plusieurs critères de séparations sont utilisés pour choisir le meilleur attribut, les plus répandus sont : [8]

- le critère du χ^2 : utilisé dans l'algorithme *CHAID*, lorsque les variables sont qualitatives ou discrètes.
- le critère de ***Gini*** : pour tout type de variables, utilisé dans l'algorithme *CART*.
- le critère Twoing, pour tout type de variables, utilisé dans l'algorithme *CART*.
- l'entropie, ou information, pour tout type de variables, utilisé dans les arbres C4.5 et C5.0

Notons qu'il n'existe pas de différences significatives quant à la qualité des arbres utilisant différents critères de séparation, chaque arbre à ses points forts et ses points faibles [9]. Pour plus de détails, les explications des heuristiques sont données dans le livre [10].

L'entropie de Shannon

Nous allons expliquer la notion d'entropie, vu qu'une partie du travail de mémoire consiste à comparer deux algorithmes d'apprentissage, dont l'apprentissage par arbre de décision en utilisant C4.5. Le « Gain d'entropie » ou « information mutuelle », est utilisé dans les deux algorithmes de Quinlan, ID3 et C4.5

Soit un ensemble *S* d'exemples divisés en n classes ω_1, ..., ω_k, ..., ω_n. L'entropie de la distribution des classes, est la quantité moyenne d'information nécessaire pour identifier la classe d'un exemple de *S*, et se définit ainsi :

$$Ent(S) = -\sum_{k=1}^{n} (P(\omega_k)\log_2(P(\omega_k))$$

Où $P(\omega_k)$ est la probabilité a priori de la classe ω_k.

Soit un attribut A de S, le gain de A est:

$$\text{Gain(S, A)} = \text{Ent(S)} - \sum_{v \in Valeur(A)}^{\infty} \left(\frac{|Sv|}{|S|} Ent(Sv) \right)$$

Où v est une valeur de A, et Sv est l'ensemble des exemples de S pour lesquels A=v.

Lors de la construction de l'arbre, à chaque division le gain d'information est calculé pour tous les attributs candidats, l'attribut choisi est celui qui maximise le gain d'information. Les algorithmes les plus simples notamment CHAID s'en tiennent à cette première étape de construction, les plus sophistiqués comportent une seconde étape d'élagage

Elagage

L'objectif de l'élagage est de supprimer les parties de l'arbre qui ne semblent pas efficaces pour prédire la classe de nouveaux cas. Le principe est de remplacer les nœuds internes proches des feuilles, par un nœud terminal, associé à la classe majoritaire. Deux types d'élagage :

Le pré élagage : consiste à fixer une règle d'arrêt qui permet de stopper la construction de l'arbre.

Le post élagage : le processus est de type *bottom-up*[5], basé sur une estimation du taux d'erreur de classification : un arbre est élagué à un certain nœud si le taux d'erreur estimé à ce nœud (en y allouant la classe majoritaire) est inférieur au taux d'erreur obtenu en considérant les sous arbres terminaux.

Exemple II-3 *Construction d'un arbre de décision : Post élagage pour la*

Considérons l'arbre suivant :

[5]Du bas vers le haut: des extrémités vers la racine

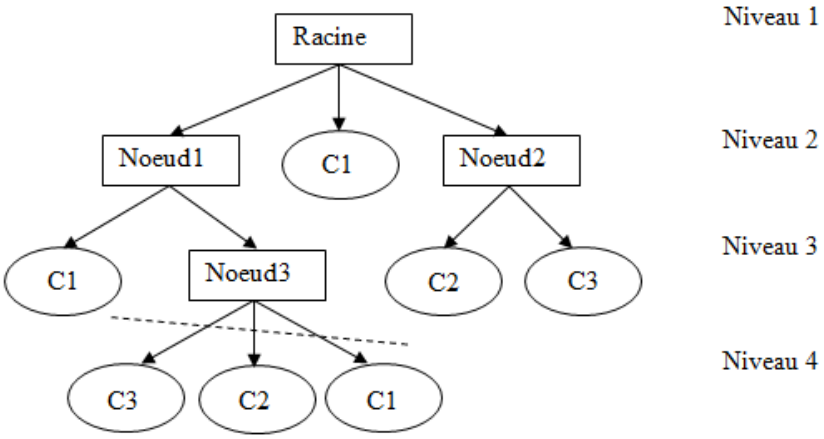


Figure II-3 Arbre de décision avant élagage

Le taux d'erreur estimé au niveau 4 (en considérant les 3 feuilles) est supérieur à celui du niveau 3 pour le nœud 3, et donc les trois branches sont élaguées et le nœud 3 devient une feuille, la classe qui lui est affectée est celle qui est majoritaire.

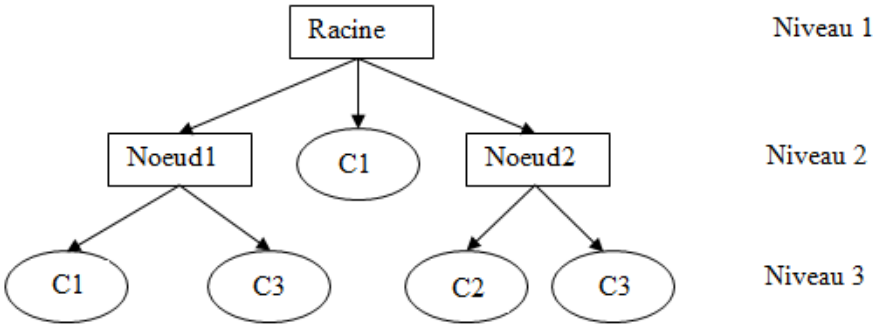


Figure II-4 Arbre de décision élagué

L'estimation de l'erreur se fait de plusieurs manières différentes, dépendantes des algorithmes utilisés, elle peut être sur la base de nouveaux exemples disponibles : le taux; d'erreur est le pourcentage d'exemples mal classés. via une validation croisée[6]. Sur base d'une estimation statistique : par exemple la borne supérieure d'un intervalle de confiance…

[6] Méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage. Pour une validation croisée à k tours, les données sont divisées en K blocs. Le processus suivant est répété k fois : apprentissage sur (k-1) blocs, test sur le k-ième bloc, l'erreur en test s'écrit ainsi e = 1/k ($\sum e_k$)

Construction de règles à partir d'arbre de décision:

Il est possible de construire un ensemble de règles à partir de l'arbre de décision, en générant une règle pour chaque feuille, et en faisant des conjonctions de tous les tests rencontrés dans le chemin depuis la racine jusqu'à cette feuille [11], Les règles construites sont de la forme : **SI** Condition (test ^ test ^ test …) **Alors** Classe. Les cinq règles générées à partir de l'arbre de décision, de la Figure I-5 sont :

SI Prévision=soleil ET humidité=élevé ALORS sport =non
SI Prévision=pluie ET vent=faible ALORS sport =oui
SI Prévision=couvert ALORS sport =oui
SI Prévision=pluie ET vent=fort ALORS sport =non
SI Prévision=soleil ET humidité=normal ALORS sport =oui

Pour un exemple donné, il existe un chemin depuis la racine jusqu'à une feuille de l'arbre, ce qui signifie qu'un exemple est couvert exactement par une seule règle.

Les algorithmes d'arbres de décision les plus répandus

ID3 (Inductive Decision Tree, Quinlan 1979)

ID3 construit un arbre de décision de façon récursive en choisissant l'attribut qui maxime le gain d'information selon l'entropie. Cet algorithme traite uniquement les variables qualitatives, un nœud est créé pour chaque valeur des attributs sélectionnés. ID3 est un algorithme basique facile à implémenter dont la première fonction est de remplacer les experts dans la construction d'un arbre de décision.

C4.5 (Quinlan 1993)

C4.5 est une amélioration d'ID3 qui permet de travailler à la fois avec des données discrètes et des données continues [6]. Il permet également de travailler avec des valeurs d'attribut absentes (valeurs manquantes). Un dernier élément de performance de C4.5 réside dans l'élagage de l'arbre construit afin de supprimer les règles inutiles et de rendre l'arbre plus compact.

C5.0 (Quinlan 1998)

L'arbre C5.0 est un perfectionnement par le chercheur australien J.Ross Quinlan de ses précédents arbres ID3 (1986) et C4.5(1993), expliqués précédemment. L'algorithme C5.0 fonctionne en cherchant à maximiser le gain d'information réalisé en affectant chaque individu à une branche de l'arbre. Il partage avec CART son adaptation à l'étude de tout type de variables ainsi que son dispositif d'optimisation de l'arbre par construction puis élagage d'un arbre maximum [8]. L'originalité de C5.0 comme de C4.5 est qu'il est accompagné d'une procédure transformant les arbres en ensemble de règles. Les règles redondantes sont supprimées, ce qui réduit la complexité de l'ensemble des règles. C5.0 généralise les règles ne menant pas à une diminution du taux d'erreur. C5.0 n'est pas binaire du fait de son traitement des variables qualitatives

II.2.2. Règles de Classification

II.2.2.1. Stratégie « Separate-and-conquer »

La stratégie « separate-and-conquer » repose sur la séquence d'opérations suivante : construire une règle qui prédit au mieux sur un ensemble d'instance (conquête) ; retirer les exemples couverts par la règle de l'ensemble d'apprentissage (séparer) [12], cette opération est répéter jusqu'à ce qu'il ne reste plus d'observations. La méthode produit ou bien des règles ordonnées, ou des règles indépendantes, ceci résulte de l'étape de séparation où nous pouvons retirer de la base toutes les observations couvertes par la règle construite, ou uniquement les observations qui sont correctement classées.

II.2.2.2. Règles de classification

L'induction de règles de classification fait partie des approches « separate-and-conquer » qui produisent de manière incrémentale un ensemble de règles de la forme : **Si** Condition **Alors** Conclusion ; où condition représente une suite de conjonctions de couples « attribut-valeur », et conclusion la classe d'affectation. Ces règles peuvent ou bien être ordonnées (appelé souvent liste de décision) ou indépendantes

II.2.2.2.1. Les listes de décision

La méthode produit une base de règles ordonnées. Lors du classement d'un individu, la première règle est évaluée. Si elle n'est pas déclenchée, on passe à la suivante, etc. Si aucune règle n'est activée, une règle par défaut est utilisée. Le modèle prend donc la forme suivante :

Si Condition 1 Alors Conclusion 1
Sinon Si Condition 2 Alors Conclusion 2
Sinon …
Sinon (Règle par défaut) Conclusion M

L'énorme avantage de cette représentation est qu'il ne peut pas y avoir de collision entre les règles, une et une seule sera activée lors du classement d'un individu [12].

Algorithme d'apprentissage des listes de décision

L'algorithme d'apprentissage repose sur le principe « separate-and-conquer ». Il peut être résumé en deux étapes imbriquées. Une première procédure encapsule la recherche des règles.

Algorithme II-2 Algorithme d'apprentissage d'une liste de décision

```
DecisionList (var.cible, var.prédictives, exemples)
Base de règles = ∅
Répéter
    Règle = Spécialiser (var.cible, var.prédictives, exemples)
    Si (Règle != NULL) Alors
        Base de Règles = Base de Règles + {Règle}
        Exemples = Exemples – {Individus couverts par la Règle}
    Fin Si
Jusqu'à (Règle = NULL)
Base de Règles = Base de Règles + {Règle par défaut (exemples)}
Renvoyer (Base de règles)
```

A chaque étape, la méthode cherche par spécialisation la règle la plus intéressante pour les observations disponibles. Si on en trouve une, elle est ajoutée à

la base de règles. Les observations couvertes sont retirées de l'ensemble de données. Le processus s'arrête lorsqu'il n'est plus possible de produire une règle [12]. Le système élabore alors la règle par défaut à partir des observations restantes. Il s'agit simplement de désigner la classe majoritaire.

La construction de la règle se fait via la fonction « Spécialiser » en cherchant la proposition qui explique le mieux une des modalités[7] de la variable cible[8], la fonction rajoute une seconde proposition qui va toujours dans le même sens, ainsi de suite jusqu'à ce qu'une règle d'arrêt soit déclenchée.

Algorithme II-3 Algorithme de construction d'une règle pour liste de décision

```
Spécialiser (var.cible, var.prédictives, exemples)
Règle = NULL
Max.Mesure = -∞
Tant Que (Règle.Arrêt(Règle) == FAUX)
    Ref.Mesure = -∞
    Pour Chaque Proposition Candidate
        Mesure = Evaluation (Règle, Proposition)
        Si (Mesure > Ref.Mesure)
        Alors
            Ref.Mesure = Mesure
            Proposition* = Proposition
        Fin Si
    Fin Pour
    Si (Ref.Mesure > Max.Mesure)
        Règle = Règle x Proposition*
        Max.Mesure = Ref.Mesure
    Fin Si
Fin Tant Que
Renvoyer (Règle)
```

[7] Valeur que prends la classe à prédire.
[8] La classe à prédire.

La fonction « spécialiser » correspond à une optimisation gloutonne[9]. Lorsqu'il n'est plus possible d'améliorer la mesure d'évaluation des règles, le processus est stoppé. La fonction « Evaluation » intervient pour tester si la règle formée avec l'adjonction de la proposition candidate est acceptable, et pour déterminer sa pertinence. Pour que l'ajout d'une nouvelle proposition dans une règle soit acceptable, il faut que celle-ci remplisse les deux conditions suivantes : le nombre d'exemples couvert soit supérieur ou égale à un seuil minimum fixé, et que son ajout apporte significativement de l'information au sens du test de KHI-2[10]. La pertinence de la règle est ensuite jugée suivant ces deux mesures : l'entropie de Shannon qui caractérise la pureté des groupes, et la J-Mesure qui caractérise la distorsion entre la distribution actuelle et la distribution initiale des classes.

Mesure d'évaluation

L'entropie de Shannon

Utilisée en générale pour les petites bases de données faiblement bruitées, la mesure produit des règles fortement spécialisées, et donc des bases de connaissances complexes, son calcule est détaillé dans la partie arbre de décision (voir Critères de séparation).

J-Mesure

Comparée à l'entropie de Shannon, la J-Mesure privilégie les solutions plus simples, avec comparativement moins de règles, celle-ci qui couvrent un ensemble plus élevée d'observations. La mesure est utilisée de préférence lorsque de grandes bases, fortement bruitées sont traités.

[9] Technique d'optimisation mathématique. Algorithme itératif qui commence par une solution arbitraire à un problème, puis tente de trouver une meilleure solution en changeant de manière incrémental un seul élément de la solution. Si le changement produit une meilleure solution, un changement graduel est fait à la nouvelle solution, répéter jusqu'à ce que le système ne puisse plus trouver d'améliorations.

[10] Pour plus de détails, voir **Rakotomalala, Ricco.** *Induction de règles prédictives* : Laboratoire ERIC, Université Lyon 2.

Les algorithmes les plus répandus

L'algorithme PART (Frank & Witten, 1998)

Génère des listes de décision. La méthode crée un arbre de décision à chaque étape, sélectionne la: branche la plus intéressante, retire les observations associées, et réitère le processus jusqu'à épuisement de la bas [13].

L'algorithme RIDOR (Ripple-Down Rules, Gaines & Compton, 1995)

La méthode démarre avec la règle par défaut, puis il essaie de produire un système qui en explique les exceptions [14]. Il poursuit jusqu'à obtention de feuilles pures [15].

II.2.2.2.2. Les règles indépendantes

Les règles indépendantes sont apparues peu de temps après les listes de décision, dans le but de facilité leur interprétation [16]. En effet, lorsque les règles sont ordonnées, pour lire correctement la règle n°i, nous devons tenir compte des (i-1) règles précédentes. L'interprétation devient difficile dès que le classifieur contient un nombre important de règles. Les règles s'écrivent de la manière suivante :

Si Condition 1 Alors Conclusion 1
Si Condition 2 Alors Conclusion 2
…
(Règle par défaut) Conclusion M

Le classement d'un nouvel individu, se fait en testant toutes les règles du classifieur. Si aucune d'entre elles n'est activée, la règle par défaut sera déclenchée. Il arrive que parfois plusieurs règles, avec des conclusions contradictoires, peuvent être déclenchées dans ce cas on s'intéresse au support de ces règles afin de déterminer la classe de l'exemple. La stratégie adoptée pour gérer ces collisions est celle que les auteurs (P. Clark and T. Niblet) [16] proposent : si lors du classement d'un individu deux règles se déclenchent, nous effectuons l'addition des fréquences observées pour chaque règle, et nous affectons à l'individu la classe majoritaire.

Algorithme d'apprentissage des règles indépendantes

L'algorithme de construction des règles procède en prenant tour à tour les classes de la variable à prédire en commençant par la plus rare, la classe la plus fréquente correspond à la conclusion par défaut. Dans la phase de construction de la règle, nous cherchons la proposition qui explique le mieux une des modalités de la variable à prédire, nous procédons par spécialisation jusqu'à ce qu'une règle d'arrêt soit déclenchée qui est en général la taille max.

Algorithme II-4 Algorithme d'apprentissage d'une base de règles indépendantes

Induction Règles (var.cible, var.prédictives, exemples)
Base de règles = ∅
Trier les classes par ordre de fréquence croissante
Pour chaque classe « c » sauf la dernière
    Sample = Exemples
    Tant que Règle != NULL
        Règle = Spécialiser (c, var.cible, var.prédictives, sample)
        Si (Règle != NULL) Alors
            Base de Règles = Base de Règles + {Règle}
            Sample = Sample – {Individus « positifs » couverts par la Règle}
        Fin Si
    Fin Tant Que
Fin Pour
Exemples = Exemples – {Individus couverts par au moins une règle}
Base de Règles = Base de Règles + {Règle par défaut (exemples)}
Renvoyer (Base de règles)

La fonction Spécialiser construit la règle comme suit :

Algorithme II-5 Algorithme de construction d'une règle pour base de règles indépendantes

```
Spécialiser (classe, var.cible, var.prédictives, exemples)
Règle = NULL
Max.Mesure = - ∞
Répéter
    Ref.Mesure = - ∞
    Proposition* = NULL
    Pour Chaque Proposition Candidate
        Mesure = Evaluation (classe, Règle, Proposition)
        Si (Mesure > Ref.Mesure)
        Alors
            Ref.Mesure = Mesure
            Proposition* = Proposition
        Fin Si
    Fin Pour
    Si (Ref.Mesure > Max.Mesure)
        Règle = Règle x Proposition*
        Max.Mesure = Ref.Mesure
    Fin Si
Jusqu'à (Proposition* = NULL)
Si Significative(Règle) =VRAI Alors Renvoyer (Règle)
Sinon Renvoyer(NULL)
```

Pour la fonction « Spécialiser », une classe de la variable à prédire est ciblée durant le processus. Nous ne retiendrons donc que les règles ayant pour conclusion « Variable à prédire = classe » à chaque étape. Ici également, la fonction « Evaluation » joue un rôle central. Elle sert dans un premier temps à vérifier si la règle couvre suffisamment d'observations en comparant le support[11] de la règle avec

[11] Le support d'une règle désigne le nombre d'observations couvertes par la règle

le seuil « Support Min » demandé par l'utilisateur. Puis à quantifier la pertinence de la règle en comparant la distribution courante de la classe et la distribution observée au début du processus d'optimisation[12]

Les algorithmes les plus répandus

Le système CN2 (Clark & Niblett, 1989)

Pour construire un ensemble de règles *H*, et pour une classe *c*, le système adopte la stratégie « separate-and-concquer ». Il construit une première règle *R* pour la classe *c* qu'il ajoute à *H* en utilisant l'ensemble des exemples *E*. il retire de *E* tous les exemples de la classe *c* couvert par la premier règle et construit une deuxième règle à partir des exemples restant. Il itère ainsi jusqu'à ce que l'ensemble E ne possède plus aucun exemple de classe *c*. cette stratégie est adopter pour chacune des classes d'apprentissage. L'algorithme construit ses règles par spécialisation, en ajoutant les meilleurs complexes[13] au sens de l'estimateur de Laplace. Le meilleur complexe est celui dont le rapport de vraisemblance est supérieur à un paramètre fixé. Nous ne détaillons pas le fonctionnement vu que nous ne utilisons pas l'algorithme pour notre système, par ailleurs, les intéressés peuvent se référé à l'article [17].

RIPPER (Repeated incremental pruning to produce error reduction algorithm, Cohen, 1995)

Développé en 1995 par Cohen, RIPPER construit un ensemble de règles indépendantes selon l'approche « separate-and-conquer », sa particularité c'est qu'il ajoute une heuristique de post élagage sur les règles, cette heuristique est appliquée au classifieur comme une phase d'optimisation [18].

a) Construction du classifieur

RIPPER construit l'ensemble des règles comme suit : Tout d'abord, pour une classe ci donnée, on différencie l'ensemble P des exemples positifs (de classe ci) de

[12] Pour une étude approfondie des mesures d'évaluation des règles, voir « Mesures d'intérêt des règles dans A PRIORI MR » :
http://tutoriels-datamining.blogspot.com/2009/02/mesures-dinteret-des-regles-dans-priori.html.
[13] Un complexe est formé de conjonction de conditions. Exp : (A=const1) ∧(B≥const2)

l'ensemble N des exemples négatifs. Les exemples de la base sont ensuite répartis aléatoirement en respectant la taille des classes en deux sous-ensembles ($P_{app} \cup N_{app}$) qui sera utilisé pour la construction d'une règle et ($P_{test} \cup N_{test}$) l'élagage de cette dernière [19]. Noter que ($P_{app} \cup N_{app}$) représente 2/3 de la base courante. Après construction d'une règle R (voir ci-dessous) en tenant compte de ($P_{app} \cup N_{app}$), celle-ci est immédiatement élaguée en utilisant ($P_{test} \cup N_{test}$) de la manière suivante :

Considérons la mesure suivante pour une règle R construite :

$$V(R, P_{test}, N_{test}) = \frac{p_{test} - n_{test}}{p_{test} + n_{test}}$$

Où p_{test} (n_{test}) est le nombre d'objets de P_{test} (N_{test}) couverts par R.

En partant de la dernière condition « c » ajouté à R, si V (R'/ c) ≥ V (R) alors on élimine « a ». Et ainsi de suite pour les autres conditions. Lorsqu'une règle est ajoutée à l'ensemble des règles, tous les exemples couverts par celle-ci sont retiré de la base.

b) Construction des règles avec RIPPER

La construction de la règle se fait en partant de la règle vide et en recherchant la condition qui maximise la mesure de gain d'informations suivante [20] :

$$\text{Gain}(R', R) = S \times (\log_2 \frac{N'_+}{N'} - \log_2 \frac{N_+}{N})$$

R : la règle initiale

R' : la règle candidate après ajout d'une condition

N (N'):le nombre d'exemple couverts par R (R')

N_+ (N'_+): le nombre de vrais positifs couverts par R (R')

S : le nombre de vrais positifs couverts par R et R' (après l'ajout d'une condition)

L'ajout se fait jusqu'à ce qu'il n'y ait plus d'exemples négatifs couverts ou critère heuristique MDL (Minimal Description Length). MDL est une techniques d'optimisation supplémentaires basées sur la longueur minimale de description Pour plus de détail veuillez consulter l'article original [18].

c) Critère d'arrêt de RIPPER

RIPPER dispose de deux conditions d'arrêt. Première condition : après chaque construction de règle R, si le taux d'erreur de R excède 50% dans ($P_{app} \cup N_{app}$), alors R n'est pas rajoutée à l'ensemble de règles et RIPPER s'arrête là. L'ensemble construit pour la classe « c_i » jusqu'alors est l'ensemble de règles finales pour « c_i ». La deuxième condition : si tous les exemples positifs sont couverts, alors RIPPER s'arrête. Dans les deux cas, RIPPER est appliquée aux classes restantes.

II.2.3. Comparaison des deux approches

Performances

Les deux approches ; arbres de décision et règles de classification fournissent des méthodes effectives qui obtiennent en un temps de calcule raisonnable de bons résultats dans la pratique [10]. Nous avons confirmé cela, en étudiant les résultats obtenues lors de notre étude comparative entre les deux méthodes (voir II.3. Comparaison expérimentale)

Optimalité

Les méthodes sont non optimales : un arbre de décision à 10 feuilles produit n'est pas le meilleur des arbres de décision à 10 feuilles. En effet, les choix dans la construction des arbres ne sont jamais remis en question; pas de backtraking [12]. De plus ces méthodes sont basées sur de nombreuses heuristiques (décider si un nœud est terminal, choix du test, choix de la classe par défaut, technique d'élagage). Pour ces algorithmes, il est possible de régler les choix de paramètre et il faut faire le bon choix des paramètres, ce qui n'est pas toujours facile.

La faiblesse des systèmes à base de règles induites, est due au fait que les exemples d'apprentissage ne sont couverts qu'une seule fois, ce qui résulte en un petit ensemble de règles inductives. En raison de la nature même de la procédure de construction de la règle qui sélectionne successivement le meilleur attribut pour accroitre une règle, certaines règles importantes peuvent être oubliées [20]. En effet, la sélection du meilleur attribut cache d'autres attributs qui peuvent être intéressants (mais un peu moins). De même, la nature de l'algorithme de couverture séquentielle ne garantit pas que l'ensemble final de règles soit le meilleur. Le fait de retirer les

objets couverts implique que les valeurs de gain calculées par la suite ne sont plus globalement optimales.

Structure du classifieur

Pour les arbres de décision, la définition des nœuds au niveau n+1 dépend énormément de celle au niveau n [10], ce qui rend la structure très rigide ; toutes les règles induites de l'arbre sont contraintes de commencer par l'attribut racine, cette rigidité donne naissance à des duplications de sous arbres à l'intérieur de l'arbre de décision [12]. L'ajout ou la modification d'une seule variable, peut entièrement modifier l'arbre.

Les arbres de décision possèdent l'avantage d'être compréhensible par tout utilisateur (si la taille de l'arbre produit est raisonnable) et d'avoir une traduction immédiate en terme de règles de décision ; le nombre de règles ainsi obtenue est le nombre de feuilles de l'arbre, une règle pour chaque feuille de l'arbre. La traduction des règles de classification en arbre de décision est moins évidente, vu que les arbres n'expriment pas facilement les disjonctions.

Par contre, l'ajout d'informations au niveau des arbres de décision n'est pas aussi simple qu'il en est pour les règles de classification, en effet, l'ajout d'une règle oblige la redéfinition de l'arbre entier. Quant aux règles de classification, leur principale avantage est la modification incrémentale en ajoutant de nouvelles règles, ou en incluant des exceptions à l'ensemble des règles sans avoir à toucher aux anciennes, ce qui rends la structure très flexible et ouverte à tous changement.

Classification des exemples

Pour les arbres de décision un exemple est couvert exactement par une seule règle, ce qui évite toute ambiguïté. Lors du classement pour les règles de classification, le traitement d'un exemple se fait à l'aide de toutes les règles couvrant l'exemple, la difficulté se trouve lorsque plusieurs règles, avec parfois des conclusions contradictoires peuvent être déclenchées, dans ce cas on s'intéresse au support de ces règles afin de déterminer la classe de l'exemple, et donc l'exemple sera étiqueté de la classe majoritaire parmi l'ensemble des exemples d'apprentissage couverts pas ces règles.

II.3. Comparaison expérimentale

Pour pouvoir choisir la méthode qui serait intéressante pour le fonctionnement de notre système, nous faisons une étude comparative entre ces deux grandes approches de l'apprentissage automatique qui ont fait leurs preuves dans le monde de la recherche : les arbres de décision que l'on classe dans la catégorie des méthodes « divide-and-conquer » et les règles de classification qui font partie de l'approche « separate-and-conquer ». Nous faisons cette comparaison dans le but de voir et d'étudier les résultats obtenus. Nous comparons les deux méthodes en utilisant l'algorithme RIPPER, qui fait l'induction de règles de classification indépendantes, et C4.5 utilisé pour le projet 2009-2010 qui construit un arbre de décision.

II.3.1. Apprentissage

Nous faisons les tests sur les rapports[14] que construit le générateur d'exemple du projet *Manage Yourself* 2009-2010. Nous avons vu que celui-ci simule plusieurs situations (voir I.3.2.1. Générateur d'exemples). Nous utilisons le concept Smartphone 4 qui introduit des données bruitées dans les rapports, c'est-à-dire des exemples ne respectant pas les règles du simulateur, l'idée est de simuler une panne matériel c'est-à-dire des plantages de l'appareil qui ne rentrent dans aucune des règles, ceci en modifiant aléatoirement un exemple de non plantage en plantage avec une probabilité de une sur cent. Nous choisissons de comparer les deux approches à travers ce concept pour tester et étudier leur capacité à apprendre les bonne règles, et ceci malgré les bruits introduits. Les rapports que nous exploitons contiennent des exemples utilisant les attributs suivants :

memoire ∈ [0,256]
applicationA ∈ {lancee,nonLancee}
applicationB ∈ {lancee,nonLancee}
applicationC ∈ {lancee,nonLancee}
dernieVersionA ∈ {oui,non}
dernieVersionB ∈ {oui,non}
dernieVersionC ∈ {oui,non}
batterie ∈ {normale, faible, vide}
plantage ∈ {oui,non}

[14] Voir **Exemple I-1** Rapport simulé par le générateur d'exemples

Rappelons les règles de simulation qu'utilise le générateur d'exemple pour le Smartphone 4:

memoire ∈ [200, 256] ∧ applicationC=lancee ⇒ plantage=oui
memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui
applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

Ces règles génèrent des exemples sous la forme :

```
169,lancee,nonLancee,lancee,oui,oui,non,vide,non
250,nonLancee,nonLancee,lancee,non,non,non,vide,oui
92,nonLancee,nonLancee,nonLancee,oui,non,non,normale,non
239,nonLancee,lancee,lancee,non,non,non,normale,oui
222,nonLancee,lancee,lancee,oui,oui,oui,vide,oui
34,nonLancee,nonLancee,nonLancee,non,non,oui,faible,oui
173,lancee,nonLancee,nonLancee,oui,oui,non,normale,non
```

II.3.1.1. Paramétrages

Pour notre étude comparative, nous utilisons le logiciel Weka[15], ceci pour la richesse de sa bibliothèque de calcul. En effet Weka intègre toute une panoplie de méthodes introuvables par ailleurs. De plus, chacune d'entre elles correspondent à une référence publiée, et donc reconnue scientifiquement. Pour l'apprentissage, nous utilisons les valeurs par défaut des paramètres des algorithmes : deux exemples au minimum par feuille, et un facteur de confiance de 0.25 pour l'élagage. L'apprentissage s'effectue avec une validation croisée à 10 tours. Un rapport à 1000 instances au format ARFF sera étudié :

[15] Logiciel libre qui propose un ensemble d'algorithmes d'apprentissage automatique. Son code java est ouvert à tous et disponible à l'adresse http ://www.cs.waikato.ac.nz/ml/weka/.

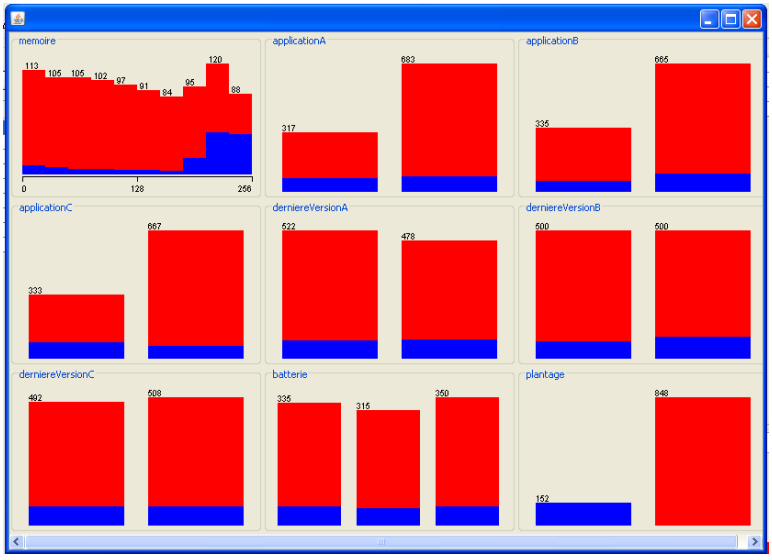


Figure II-5 Répartition des 1000 exemples testés sur le Smartphone 4

II.3.1.2. Apprentissage par arbre de décision

Nous appliquons sur le fichier d'exemples, l'algorithme C4.5 de Quinlan [6] pour l'apprentissage d'arbre de décision, celui-ci est implémenté dans le logiciel de Weka sous le nom J48.

II.3.1.2.1. Résultats et discussion

Arbre construit

Après avoir fixé les paramètres d'exécution et lancé l'algorithme d'apprentissage sur le rapport d'exemple que le générateur avait fournis, l'arbre de décision appris est donné par la Figure II-6:

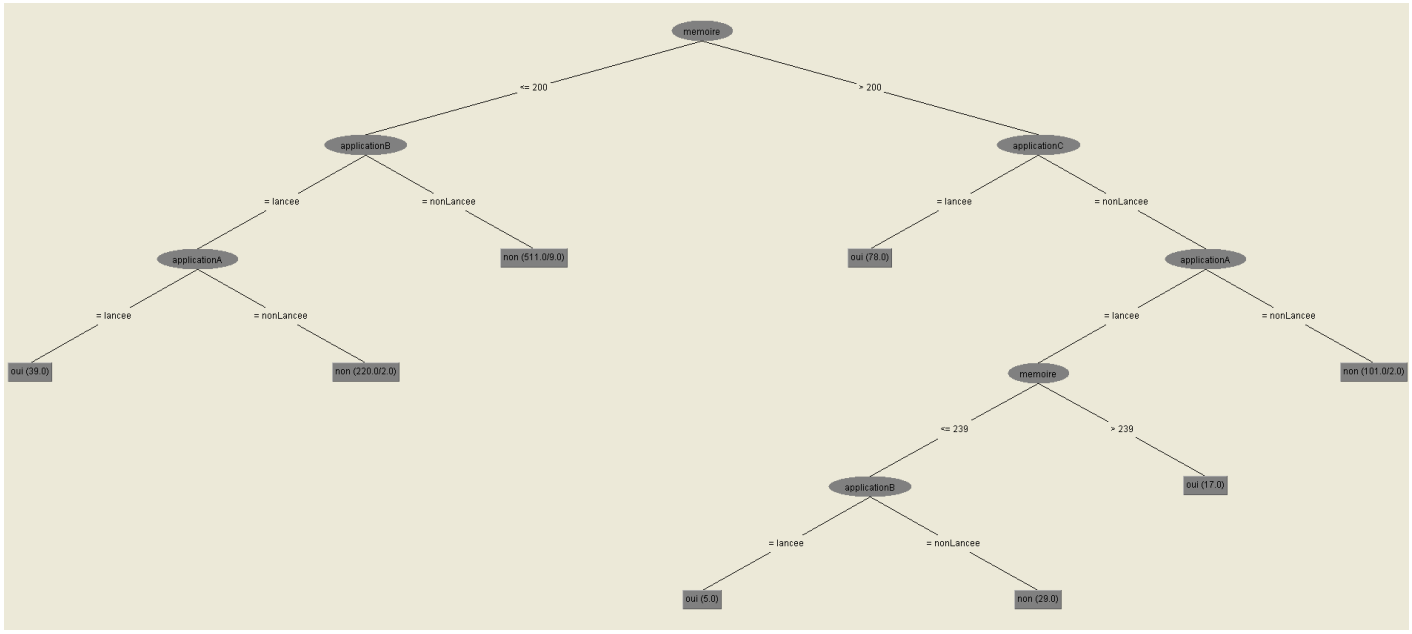


Figure II-6 Arbre de décision Smartphone 4

Règles apprises

Comme le montre l'arbre de la Figure II-6 huit règles sont apprises dont quatre de plantage, ceci en un temps de 0.05 secondes.

RA1 memoire ≤ 200 ∧ applicationB = lancee ∧ applicationA = lancee ⇒ plantage=oui

RA2 memoire > 200 ∧ applicationC = lancee ⇒ plantage=oui

RA3 mémoire ∈] 200, 239] ∧ applicationC = nonLancee ∧ applicationA = lancee ∧ applicationB = lancee ⇒ plantage=oui

RA4 memoire > 239 ∧ applicationC = nonLancee ∧ applicationA = lancee ⇒ plantage=oui

RA5 memoire ≤ 200 ∧ applicationB = nonlancee⇒ plantage= non

RA6 memoire ≤ 200 ∧ applicationB = lancee ∧ applicationA = nonlancee ⇒ plantage=non

RA7 mémoire ∈] 200, 239] ∧ applicationC = nonLancee ∧ applicationA = lancee ∧ applicationB = nonlancee ⇒ plantage=oui

RA8 SI memoire > 200 ∧ applicationC = nonLancee ∧ applicationA = nonlancee ⇒ plantage=non

Pour notre système qui fait la prévention de pannes, les règles de non plantage ne nous intéressent pas. Regardons de plus près les règles de plantage. Pour pouvoir juger l'efficacité de l'algorithme d'apprentissage utilisé, nous comparons les règles de plantages apprises aux règles du simulateur. Leur rapprochement en terme

de couverture d'exemple prouverait l'efficacité de l'algorithme. Pour un apprentissage parfait, les deux ensembles seraient identiques, cela montrerait que l'algorithme a réussis à apprendre toute les situations simulées.

Règles de simulation :

RS1 memoire ∈ [200, 256] ∧ applicationC=lancee ⇒ plantage=oui
RS2 memoire ∈ [240, 256] ∧ applicationA=lancee ⇒ plantage=oui
RS3 applicationA=lancee ∧ applicationB=lancee ⇒ plantage=oui

En comparant les deux ensembles de règle, nous remarquons qu'une seule règle du simulateur a été apprise (Rs_1), toutes les autres règles apprises sont des spécialisations des règles du simulateur :

RA1 est une spécialisation[16] de *RS3*
RA2 est équivalente à *RS1*
RA3 est une spécialisation de *RS3*
RA4 est une spécialisation de *RS2*

Performances

Le taux de reconnaissance[17] de l'arbre généré n'étant pas de 100%, il est intéressant ici d'analyser un peu plus finement les résultats obtenus :

Nombre d'exemples	% de reconnaissance	Nombre d'ex mal classés	Nombre de plantages non détectés	Nombre de normal non détectés	Nombre d'ex modifiés
1000	98,5	15	14	1	13

L'algorithme réussit à classer correctement 98.5% d'exemples, contre 1.5% d'exemples mal classées, sachant que 1.3% des exemples représente les exemples bruités introduit aux rapports. On s'aperçoit alors que les exemples mal classés, à deux exceptions près, sont les exemples bruités.

[16] L'ensemble d'exemple couvert par la règle RA1 est inclus dans celui couvert par la règle RS3
[17] Pourcentage des exemples bien classés

II.3.1.3. Apprentissage par règles de classification

Sur le même fichier d'exemples nous faisons l'apprentissage avec l'algorithme RIPPER de Cohen [19]. L'algorithme produit des règles indépendantes. Implémenté dans le logiciel Weka sous le nom de JRip [15]

II.3.1.3.1. Résultats et discussion

Règles apprises

Trois règles apprises en un temps de 0.03 secondes, plus une quatrième règle qui est la règle par défaut.

RA1 memoire ≥ 201 and applicationC = lancee ⇒ plantage=oui
RA2 applicationA = lancee and applicationB = lancee⇒ plantage=oui
RA3 memoire ≥ 240 and applicationA = lancee ⇒ plantage=oui
Règle par défaut : Plantage=non

Nous remarquons que l'apprentissage par règles de classification apprend trois règles de plantage et une règle de non plantage, ceci pour la nature de l'algorithme qui construit le classifieur en prenant tour à tour chaque classe, en commençant par la plus rare, la plus fréquente correspond à la règle par défaut. Ceci est intéressent pour notre système vu que l'objectif est de prévenir tout dysfonctionnement des appareils mobiles grâce au système d'apprentissage, apprendre les cas de non plantage nous intéresse moins. En comparant les règles de simulation avec les règles apprises nous remarquons que celles-ci sont identiques :

RA1 est la même que RS1
RA2 est la même que RS3
RA3 est la même que RS2

Ce qui nous mène à comprendre que l'algorithme d'apprentissage a réussis à apprendre toutes les situations problématiques qui ont été simulé, ce qui laisse penser que l'algorithme choisit s'avère très efficace.

Performances

Nombre d'exemples	% de reconnaissance	Nombre d'ex mal classés	Nombre de plantages non détectés	Nombre de normal non détectés	Nombre d'ex modifiés
1000	98,6	14	14	0	13

Les résultats sont les mêmes que ceux de l'apprentissage par arbre de décision, nous remarquons que les exemples mal classés sont les exemples bruités qui ont été introduit.

II.3.2. Comparaison des résultats

Récapitulons les résultats obtenus.

Rapport	Méthodes d'apprentissage	Nombre de règles apprises	Taux d'erreur (%)	Temps de calcul (sec)
1000 Instances, dont 13 bruits	Arbre de Décision	8	1.5	0.05
	Règle de classification	4	1.4	0.03

En analysant les ensembles de règles appris par les deux approches, nous avons remarqué que l'apprentissage par règles de classification contrairement à celui par arbre de décision, déduit les règles de plantages, et une seul règle de non plantage qui correspond à la règle par défaut, ce qui est intéressent à utiliser pour notre système qui fait de la prévention de pannes. Nous avons remarqué aussi que l'apprentissage par arbre de décision génère de nouvelles règles, qui sont des spécialisations des règles de simulation, par contre pour celui par règles de classification, les règles apprises sont similaires aux règles de simulation cela prouve la force de l'algorithme à apprendre toutes la situation problématiques simulées. Coté performances, nous ne remarquons pas de grandes différences entre les deux méthodes (taux d'erreur, Temps de calcul).

Nous avons effectué des apprentissages avec 2000 et 3000 instances et nous avons observé les même résultats, et ce pour les deux méthodes.

Conclusion

Nous avons présenté dans ce chapitre de manière générale le domaine de l'apprentissage automatique. Nous avons étudié et détaillé le fonctionnement de deux grandes méthodes de l'apprentissage supervisé ; « l'apprentissage par arbres de décision » qui fait partie de l'approche « divide-and-conquer », et « l'apprentissage par règles de classification » que celui-ci fait partie de l'approche « separate-and-conquer ». Pour pouvoir choisir la méthode qui serait intéressante pour le

fonctionnement de notre système, nous avons fait une étude comparative (voir II.3. Comparaison expérimentale) entre les deux techniques. Les résultats ne montrent pas une grande différence en ce qui concerne l'efficacité des algorithmes, mais nous nous sommes aperçus que ce qui nous intéresse le plus est un apprentissage par règle de classification, pour la nature des règles déduites ainsi que pour son aspect incrémentale. Dans le chapitre suivant nous présenterons le système que nous avons mis en place pour le projet *Manage Yourself 2010-2011*, tout en détaillant les différents modules proposés.

Chapitre 3

Modèle proposé

III. Modèle proposé

Introduction

Le projet *Manage Yourself* est un projet de diagnostic et surveillance de plates-formes embarquées son objectif est de faire de la prévention de pannes sur mobiles. Deux parties constituent le projet ; l'application embarquée, et le serveur sur lequel nous travaillons. Quand l'appareil mobile rencontre une situation problématique qui n'arrive pas à reconnaitre celui-ci envoie au serveur le rapport de la situation juste avant le problème, le serveur récupère ce rapport dans le but de déduire des règles de dysfonctionnements par apprentissage et d'y remédier en définissant des règles de corrections, ces correctif seront par la suite renvoyé à l'appareil mobile. Un premier logiciel a été développé l'année universitaire 2009-2010 par des étudiants de l'INSA de Rennes. Nous avons étudié son fonctionnement au niveau du serveur dans le chapitre I, dans le but de proposé des améliorations pour le nouveau logiciel (*Manage Yourself* 2010-2011) qui est en cours de réalisation.

III.1. Problématique

Le but de notre système est de déduire à partir des rapports reçus un ensemble de règles correctives *RC* qui soit capable de traiter le maximum de situations problématiques que l'appareil mobile pourrait rencontrer, et ce de manière efficace.

Considérons : $\mathbb{C}$: espace de données (exemples que contient le rapport[18] envoyé par l'appareil mobile), $\mathbb{C}^*$: espace de donnée sans plantages et $\mathbb{C}' = \mathbb{C} - \mathbb{C}^*$: espace de donnée avec plantages. L'objectif de notre système est de diminuer l'espace $\mathbb{C}'$ en définissant RC^* : l'ensemble des règles correctives délimitant $\mathbb{C}^*$ comme le montre la figure.

[18] Un exemple de rapport est décrit dans la section : I.3.2.1. Générateur d'exemples

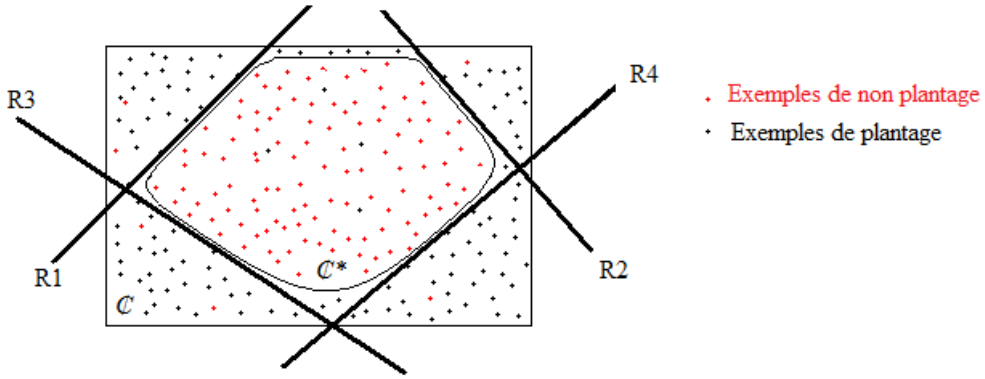


Figure III-1 Modélisation mathématique de la problématique

Les règles R_1, R_2, R_3, R_4 Délimitent l'espace $\mathbb{C}$* de la manière la plus pertinente, et donc les 4 règles constituent l'ensemble *RC**

A chaque réception de l'ensemble des rapports de situation par le serveur, l'apprentissage, déduit un ensemble de règles R_i délimitant un espace de donnée $\mathbb{C}_i$, cet ensemble sera étiqueté (affectation d'actions) et renvoyé à l'application mobile, nous l'appelons l'ensemble RC_i, le but de notre système est que $\mathbb{C}_i$ converge vers $\mathbb{C}$*, pour cela RC_i doit converger vers *RC**. La question que traite notre système est : comment RC_i converge vers *RC** de la manière la plus efficace. Pour cela nous avons introduit un moyen pour traiter l'ensemble des règles *RC* avant de le renvoyer à l'appareil mobile. Nous l'avons appelé module de sélection celui-ci permet à chaque itération d'améliorer l'ensemble *RC* en sélectionnant les meilleurs règles afin qu'il converge vers *RC**.

III.1. Fonctionnement globale

La partie serveur du projet *Manage Yourself* (2009-2010) comprend un module d'apprentissage qui apprend les règles de plantages par arbre de décision (C4.5) et une interface administration qui permet d'affecter des correctifs à chaque règle apprise. Pour le serveur du projet *Manage Yourself* 2010-2011 nous reprenons l'architecture en modifiant l'algorithme d'apprentissage et en intégrant le module de sélection qui permet la mise à jour des connaissances embarquées. Le projet comprend aussi une application embarquée sur l'appareil mobile. Pour pouvoir développer le traitement au niveau du serveur nous avons simulé le fonctionnement de l'application embarquée, pour cette simulation nous avons repris le générateur d'exemples du projet *Manage Yourself* (2009-2010) et nous avons intégré un

moyen pour simuler l'exécution des règles correctives au niveau de l'appareil mobile, nous l'avons appelé module de filtrage. Nous illustrons par la suite les deux architectures proposées

III.1.1. Architecture du système

Dans une situation réelle, l'architecture finale du projet que nous avons mis en place et qui sera exploitée par les utilisateurs serait :

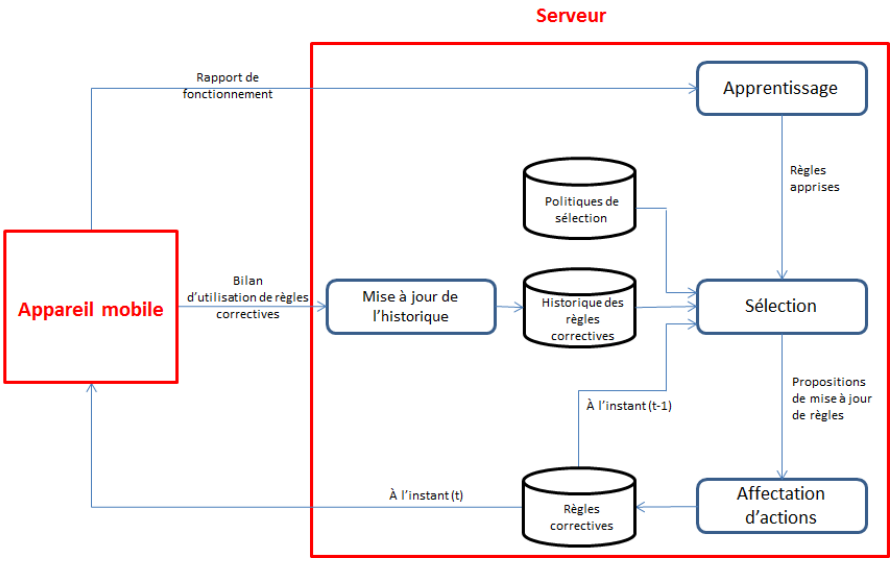


Figure III-2 Architecture du système

L'application présente sur l'appareil mobile surveillera le système de manière continue. Celle-ci sera capable de détecter des situations problématiques (situations amenant à un plantage) et de les corriger grâce à un ensemble de règles, téléchargé au préalable depuis le serveur, nous appelons cet ensemble: ensemble de règles correctives. Pour un disfonctionnement non détecté l'appareil mobile devra envoyer le rapport de la situation juste avant le plantage au serveur, ce rapport contiendra l'état du système (niveau de batterie, état de la mémoire …), voir Exemple **I-1** Rapport simulé par le générateur d'exemples, le but est d'apprendre à partir de ces rapports, des situations susceptibles de conduire à un plantage et d'en apporter les solutions, ceci en créant de nouvelles règles correctives qui seront ensuite envoyées au Smartphone.

III.1.2. Architecture simulée

Nos contributions concernent la partie serveur du projet. Nous avons donc simulé le fonctionnement de l'appareil mobile dans le but de pouvoir faire les tests nécessaires. Cette simulation nous permettra d'optimiser le temps d'expérimentation. Notre système qui est expliqué dans la suite se présente ainsi:

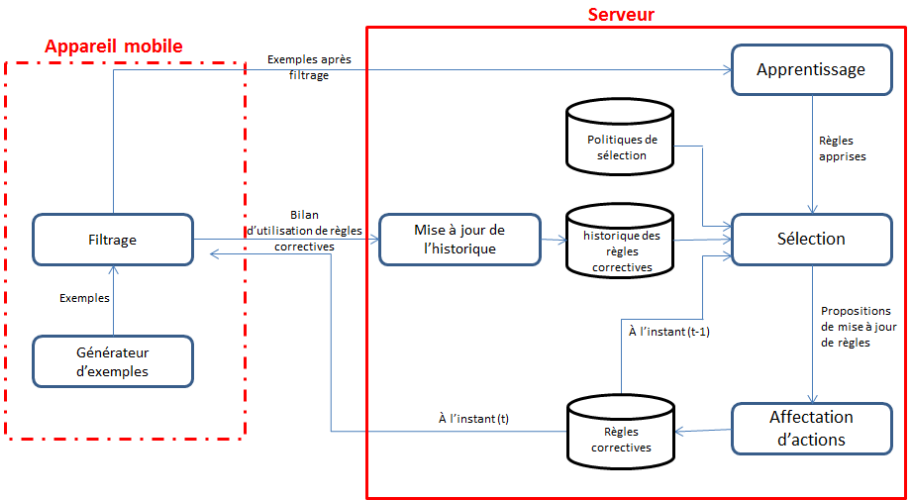


Figure III-3 Architecture simulé

Nous avons simulé la correction des plantages au niveau du Smartphone en intégrant un module de filtrage (voir III.2.2. Filtrage)

III.2. Description de l'architecture du système proposé

Le système que nous avons mis en œuvre pour le projet Manage Yourself 2010-2011, comprend 5 modules de traitement de l'information et de gestion de connaissances à savoir : le générateur d'exemples que nous avons repris du projet 2009-2010 celui-là a pour objectif de simuler l'envoie des rapports de plantage. Le module de filtrage que nous avons conçus et qui simule le traitement de l'appareil mobile. Le module d'apprentissage que nous avons développé en modifiant l'algorithme d'apprentissage, celui-ci permet l'évolution du système par la découverte des règles correctives. Le module de sélection que nous avons conçus et mis en place pour trier les connaissances à embarquer et enfin l'affectation d'actions que nous avons repris du logiciel existant, qui permet à un expert humain d'apporter des corrections aux situations problématiques que rencontre le mobile. Le fonctionnement de chaque module est détaillé ci-dessous :

III.2.1. Générateur d'exemples

Dans une situation réelle, les appareils mobiles enverront au serveur des rapports d'état du système. Le générateur d'exemples a été développé par les étudiants de l'INSA, dans le but de simuler les rapports que pourraient générer des Smartphones. Le fonctionnement est détaillé dans le chapitre *Manage Yourself* (voir I.3.2.1. Générateur d'exemples). Ce module nous fournit les données qui nous permettront de faire l'étude de l'apprentissage.

III.2.2. Filtrage

Sur le projet *Manage Yourself* de l'année universitaire 2009-2010 l'apprentissage se faisait de manière très naïve, celui-ci récupérait les rapports produits par le simulateur et appliquait dessus l'algorithme d'apprentissage, sans prendre en considération les règles correctives déjà attribués. Cette modélisation empêchait le système d'évoluer dans le temps. Pour se rapprocher d'une situation réelle nous avons introduit un module de filtrage qui simule l'exécution des règles correctives au niveau du Smartphone. Dans une situation réelle, l'appareil mobile empêche les situations de plantage connues grâce aux règles correctives présentes sur le mobile, et envois au serveur un rapport contenant les situations non connues. Dès qu'une règle corrective est installée sur l'appareil mobile, les exemples couverts par celle-ci ne risquent plus de figurer sur les rapports de plantage.

Dans cette partie, le traitement consiste à récupérer les rapports d'exemples générés par le module de simulation à l'instant 't', et à appliquer dessus les règles correctives construite à l'instant 't-1'. L'application du filtre consiste à retirer tous les exemples couverts par les règles correctives. On dit que la règle *R* couvre l'exemple *E* si *E* remplit les conditions de *R*. Et donc si l'on a par exemple, une règle corrective de la forme : Si la mémoire > 199 alors viderMémoire, le filtrage retire tous les exemples dont la mémoire >200. A la fin le module retourne le rapport d'exemples qui va être traité par la partie apprentissage, et le bilan d'exécution de chaque règle corrective.

Exécution des règles correctives

Le bilan d'exécution de chaque règle corrective résulte de la partie filtrage. Lorsqu'on applique le filtre sur les rapports produits par le générateur d'exemples, le système supprime tout exemple couvert par au moins une règle corrective présente sur l'appareil mobile. Comme nous ignorons la manière dont les règles correctives vont être exécutées sur le Smartphone, nous interprétons l'exécution des règles corrective par la couverture d'exemples, ce qui est une simplification peu réaliste mais qui nous permet de mettre en place notre plateforme sans attendre la fin du développement du nouveau simulateur (projet 2010-2011) beaucoup plus réaliste. Un exemple peut être couvert par une ou plusieurs règles correctives, comme il peut ne pas être couvert par aucune de ces règles.

Exemple III-1 *Exécution des règles correctives*

Supposons que le générateur d'exemples a fourni le rapport suivant :

```
@relation smartphone4
@attribute memoire numeric
@attribute applicationA {lancee,nonLancee}
@attribute applicationB {lancee,nonLancee}
@attribute applicationC {lancee,nonLancee}
@attribute dernieereVersionA {oui,non}
@attribute dernieereVersionB {oui,non}
@attribute dernieereVersionC {oui,non}
@attribute batterie {normale,faible,vide}
@attribute plantage {oui,non}
@data
E1    228,nonLancee,nonLancee,nonLancee,oui,oui,oui,faible,non
E2    249,lancee,lancee,lancee,non,oui,oui,faible,oui
E3    169,lancee,nonLancee,lancee,oui,oui,non,vide,non
E4    10,lancee,lancee,nonLancee,oui,oui,oui,normale,oui
E5    250,nonLancee,nonLancee,nonLancee,oui,non,non,normale,oui
E6    92,nonLancee,nonLancee,nonLancee,oui,non,non,normale,oui
E7    34,nonLancee,nonLancee,nonLancee,non,non,oui,faible,non
E8    173,lancee,nonLancee,nonLancee,oui,oui,non,normale,non
E9    135,lancee,nonLancee,nonLancee,oui,oui,non,normale,oui
```

Et que l'ensemble des règles correctives servant pour le filtrage à l'instant 't'est le suivant :

R1 SI applicationA=lancee ET demandeLancementApplicationB ALORS couperApplicationA

R2 SI memoire ≥ 209 ALORS libererEspaceMemoire

La règle corrective *R1* couvre les exemples : *E2* et *E4*, quant à la règle *R2*, celle-ci couvre les exemples : *E1*, *E2* et *E5*. Nous remarquons que l'exemple *E2* est couvert en même temps par les deux règles correctives *R1* et *R2*. Après l'exécution du filtre

qui retire tout exemple couvert par au moins une règle, le rapport sur lequel nous appliquons l'apprentissage serait :

```
@relation smartphone4
@attribute memoire numeric
@attribute applicationA {lancee,nonLancee}
@attribute applicationB {lancee,nonLancee}
@attribute applicationC {lancee,nonLancee}
@attribute dernieereVersionA {oui,non}
@attribute dernieereVersionB {oui,non}
@attribute dernieereVersionC {oui,non}
@attribute batterie {normale,faible,vide}
@attribute plantage {oui,non}
@data
E3    169,lancee,nonLancee,lancee,oui,oui,non,vide,non
E6    92,nonLancee,nonLancee,nonLancee,oui,non,non,normale,oui
E7    34,nonLancee,nonLancee,nonLancee,non,non,oui,faible,non
E8    173,lancee,nonLancee,nonLancee,oui,oui,non,normale,non
E9    135,lancee,nonLancee,nonLancee,oui,oui,non,normale,oui
```

Et le bilan d'exécution de l'ensemble des règles correctives serait :

R1 couvre 2 exemples et *R2* couvre 3 exemples

III.2.4. Apprentissage

L'apprentissage est la partie cœur de notre système, celle-ci permet d'apprendre à partir des rapports envoyés par les appareils mobiles, les situations problématiques que peuvent rencontrer ces derniers. Des solutions seront proposées par la suite et renvoyées au Smartphone. L'apprentissage est la partie qui permet l'évolution du système par la découverte des nouvelles règles de plantages. Le module d'apprentissage consiste à récupérer les rapports envoyés par les appareils mobiles et à appliquer dessus l'algorithme d'apprentissage. Après la découverte des règles de plantages, celles-ci passe par le module de sélection (voir III.2.5. Sélection) où l'ensemble des règle a étiquetée par l'expert humain sera sélectionné. Dans l'architecture simulée l'apprentissage se fait ainsi :

Apprentissage (i) : E_i, = Générateur (RS)

E_i^F = Filtre (E_i, RC_{i-1})

R_i = Learn (E_i^F)

Avec : R_i : ensemble des règles apprise a l'itération 'i', E_i^F : ensemble des exemples filtrés à l'itération 'i', E_i : l'ensemble des exemples simulé par le générateur d'exemple à l'itération 'i', RC_{i-1} : ensemble des règles correctives à

l'itération 'i-1' et *RS* : l'ensemble des règles de simulation. L'apprentissage est supervisé. Nous utilisons la bibliothèque Weka qui contient de nombreux algorithmes.

III.2.2.1 Algorithme d'apprentissage

Nous avons fait toute une étude bibliographique pour pouvoir choisir l'algorithme d'apprentissage que nous utiliserons (voir Apprentissage automatique). Après avoir étudié chacune des deux approches : apprentissage par arbres de décision et apprentissage par règles de classification (voir II.2. Etat de l'art) nous avons fait une étude comparative entre les deux grandes méthodes, et après avoir analysé les résultats obtenus (voir II.3. Comparaison expérimentale), nous décidons d'appliquer un apprentissage par règles de classification, ceci pour sa cohésion avec nos objectifs et pour la qualité de ses performances observées. L'algorithme que nous utilisons pour l'apprentissage des règles de classification, est l'algorithme RIPPER de Cohen [19] L'algorithme est implémenté dans la bibliothèque Weka sous le nom de JRip [7]. L'algorithme produit de manière incrémentale un ensemble de règles de classifications dites indépendantes.

III.2.5. Sélection

Pour éviter tout dysfonctionnement, l'appareil mobile est doté de moyens d'autosurveillance, qui lui permet de détecter et de corriger les situations problématiques, ceci se fait grâce à un ensemble de règles correctives installées sur le Smartphone. Ces connaissances embarquées, doivent d'être remises à jour régulièrement pour assurer la fiabilité du système et pour répondre aux besoins de l'utilisateur. Cette mise à jour consiste à ajouter de nouvelles règles à l'ensemble des règles correctives, et à supprimer les anciennes règles qui ne sont plus pertinentes. Pour cela nous avons mis en place un module de sélection au niveau du serveur qui va permettre au système de sélectionner les règles correctives qui seront exploitables par l'appareil mobile. Le but de la sélection est d'éviter la surcharge du Smartphone, de remédier à tout conflit entre règles correctives installées, et d'assurer la fiabilité du système de surveillance embarqué.

Le principe est de sélectionner un ensemble de règles correctives qui soit le plus pertinent possible, cet ensemble sera consulté dans la partie *« affectation d'actions »* de notre système où un expert humain pourrait encore le modifier. Ces

règles seront envoyées à l'appareil mobile et seront prêtes à être exécutées. La sélection se fait entre l'ensemble des règles correctives déjà présentes sur l'appareil mobile (règles correctives à l'instant t-1), et l'ensemble des nouvelles règles apprises, ceci en prenant en considération l'historique de chaque règle corrective. La sélection se fait suivant les différentes politiques de sélection proposées (voir III.2.5.3. Politiques de sélection).

III.2.5.1. Historique des règles correctives

Pour pouvoir sélectionner les règles correctives qui répondent le mieux aux besoins de l'utilisateur, nous avons mis en place un moyen de sauvegarde qui garde en mémoire le bilan d'exécution de chaque règle corrective depuis son installation sur l'appareil mobile jusqu'à sa suppression. Nous avons conçus ce moyen de sauvegarde pour pouvoir voir l'évolution dans le temps de chaque règle corrective en terme d'exécution, ceci nous aidera à décider de quelle règle garder et de quelle règle ne pas garder.

règles \ *itération*	T_1	.	.	.	.	.	T_m
R_1	x_{11}						x_{1m}
.	.						.
.	.						.
R_n	x_{n1}						x_{nm}

Figure III-4 Historique des règles correctives

X_{nm} représente le bilan d'utilisation de la rège R_n à l'itération T_m

III.2.5.2. Notion de généralisation et de spécification entre les règles correctives

Nous définissons la notion de généralisation et de spécification entre les règles correctives en terme de couverture d'exemples ; on dis que la règle *R1* est plus générale que la règle *R2* si l'ensemble d'exemples couvert par *R1* est inclus dans celui couvert par *R2*.

Soit *E* un espace où nous représentons les exemples que contient les rapports envoyés par l'application mobile. Soit *R1* et *R2* deux règles correctives, nous représentons leurs espaces de couverture par des cercles. Voir Figure III-5.

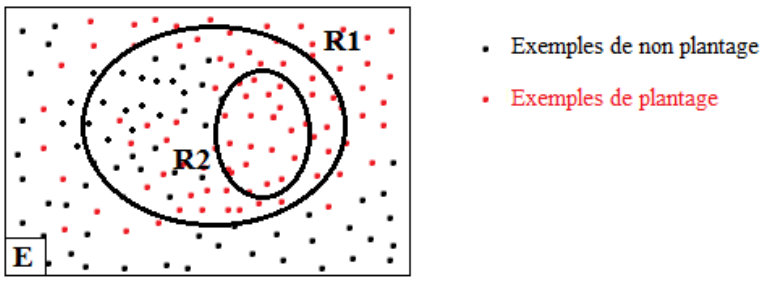


Figure III-5 Généralité et spécificité entre règles correctives

R1 est plus générale que *R2*, ce qui revient à dire que *R2* est plus spécifique que *R1*.

III.2.5.3. Politiques de sélection

La sélection consiste à choisir un ensemble de règles correctives *RC* qui soit capable de traiter les situations problématiques que l'appareil mobile pourrait rencontrer. Considérons $\mathcal{C}$: espace de données (exemples que contient le rapport envoyé par l'appareil mobile), $\mathcal{C}^*$: espace de données sans plantage et $\mathcal{C}' = \mathcal{C} - \mathcal{C}^*$: espace des données avec plantage. Pour pouvoir séparer les deux espaces ; $\mathcal{C}^*$ et $\mathcal{C}'$ de la manière la plus pertinente, nous devons définir RC^* : l'ensemble des règles correctives délimitant $\mathcal{C}^*$ (voir III.1. Problématique). A chaque réception du rapport de situation par le serveur, l'apprentissage, déduit un ensemble de règles de plantages R_i. Le module de sélection définit à chaque itération 'i' l'ensemble RC_i en sélectionnant les meilleures règle depuis RC_{i-1} et R_i pour délimiter un espace $\mathcal{C}_i$. Les politiques de sélection fixent des contraintes sur les deux ensembles de règles pour que $\mathcal{C}_i$ converge de la manière la plus efficace vers $\mathcal{C}^*$. Nous proposons deux types de politiques ; politiques pour l'ajout de règles et politique pour la suppression de règles.

III.2.5.3.1. Politiques de sélection pour l'ajout de règles

Ces politiques fixent des contraintes sur l'ensemble des nouvelles règles de plantage apprises. Dès que le module d'apprentissage retourne l'ensemble des règles apprises R_i, le module de sélection choisit parmi toutes ces règles, celles qui seront

intégrées à l'ensemble des règles correctives RC_i, et ce à l'aide des politique de sélection pour l'ajout de règles.

Ajout automatique

Cette politique permet de sélectionner toutes les règles apprises sans exceptions. L'ensemble des règles de plantage apprit sera automatiquement transféré au module d'affectation d'actions où l'expert humain attribuera une action corrective à chaque règles, celui-ci peut aussi en ajouter d'autres ou supprimer quelque unes. L'ensemble qui résulte sera transféré à l'appareil mobile.

Ajout par généralité

Les nouvelles règles apprises ne sont pas ajoutées automatiquement à l'ensemble des règles correctives, un test de généralité est appliqué entre les règles des deux ensembles (l'ensemble des règles apprises et celui des règles correctives). Si pour une règle apprise *R,* il existe une règle corrective *R'* plus générale, la règle *R* ne sera pas ajouté à l'ensemble des règles corrective. Mais si la règle apprise *R* est plus générale qu'une règle corrective *R'; R* sera ajouté à l'ensemble des règles corrective e*t R'* sera supprimée. Cette politique a été mise en place pour éviter l'installation sur le Smartphone des règles inutiles.

III.2.5.3.2. Politique de sélection pour la suppression de règles

Ces politiques traitent les règles correctives déjà présentes sur l'appareil mobile, en fixant des contraintes sur l'ensemble RC_{i-1}. Nous les avons mis en places dans le but de retirer toute règle jugée plus efficace. Si par exemple la règle *R* n'est plus utilisée depuis l'installation d'une règle *R',* l'idéal serait de retirer *R* pour éviter la surcharge du Smartphone.

Par seuil de couverture :

Dès qu'une règle corrective est installée sur l'appareil mobile, les exemples couverts par celle-ci, ne risquent plus de figurer sur les rapports de plantage. Alors qu'à partir de ces exemples on pourrait apprendre de nouvelles situations problématiques. Cette politique retire une règle couvrant un nombre élevé

d'exemples dans le but de découvrir des règles plus spécifiques[19] délimitant un espace plus proche de l'espace optimal $\mathcal{C}^*$.

Supposons qu'à l'itération 'i' nous avons l'ensemble des règles correctives *RCi* qui délimite l'espace $\mathcal{C}i$ par les trois règles *R1*, *R2* et *R3* comme le montre la Figure III-6.

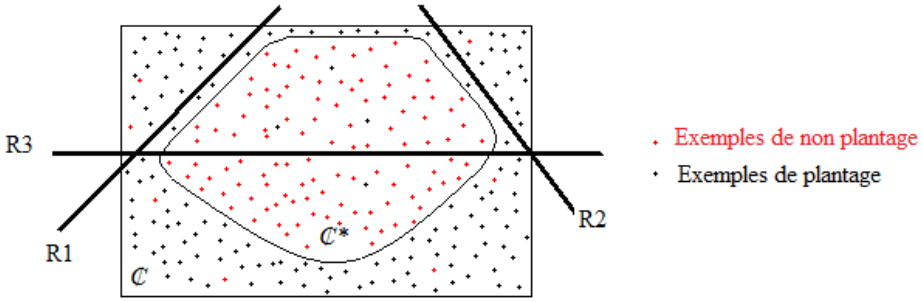


Figure III-6 couverture avant l'utilisation de la politique (seuil de couverture)

Remarquons que R_3 couvre un pourcentage élevé d'exemples, en la retirant nous donnons la possibilité de découvrir par apprentissage de nouvelles règles qui serait plus spécifiques que R_3 et qui délimite un espace plus proche de $\mathcal{C}^*$ comme le montre la Figure III-7.

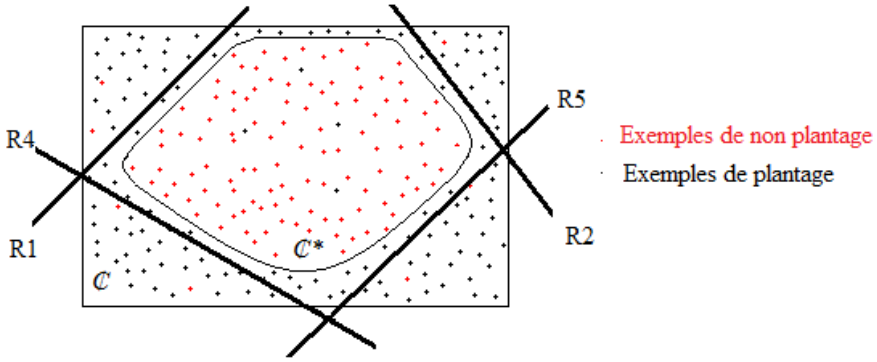


Figure III-7 couverture après l'utilisation de la politique (seuil de couverture)

Nous avons fixé le seuil de couverture à 50%,

[19] Voir III.2.5.2. Notion de généralisation et de spécification entre les règles correctives

Roue de la chance : méthode probabiliste et aléatoire

Après avoir mis en place la politique *« par seuil de couverture »* qui retire les règles couvrant un seuil fixé d'exemples, nous avons pensé qu'i serait intéressant de voir ce qu'il en est en retirant les autres règles. Cette politique permet de donner à chacune des règles une chance d'être retirée à un certain moment suivant une loi à la fois probabiliste et aléatoire. Nous calculons le pourcentage de la moyenne d'utilisation de chaque règle depuis 't_0' jusqu'à 't_i '. Supposons qu'à l'instant 't' nous avons 4 règles correctives installées sur l'appareil mobile :

R_1 : 40% d'utilisation depuis 't_0' jusqu'à 't'
R_2 : 20% d'utilisation depuis 't_0' jusqu'à 't'
R_3 : 10% d'utilisation depuis 't_0' jusqu'à 't'
R_4 : 30% d'utilisation depuis 't_0' jusqu'à 't'

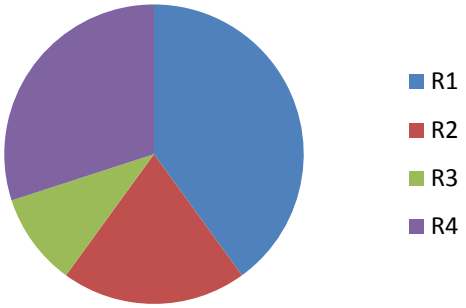


Figure III-8 Pourcentages d'utilisation des règles (Roue de la chance)

Nous tirons aléatoirement un nombre *N* entre 0 et 1 :

Si $N \in [0, 0.4]$ nous retirons la règle 1 de l'ensemble des règles correctives
Si $N \in [0.4, 0.6]$ nous retirons la règle 2
Si $N \in [0.6, 0.7]$ nous retirons la règle 3
Si $N \in [0.7, 1]$ nous retirons la règle 4

Sélection naïve

Cette politique consiste à retirer à un moment donné la règle corrective qui a la moyenne d'utilisation la plus élevée comparé avec le reste des règles. A partir de l'historique d'utilisation des règles correctives, nous calculons la moyenne d'utilisation de chaque règle depuis 't_0' jusqu'à 't_i ' et nous retirons la règles qui a la moyenne d'utilisation la plus élevée. Ceci pour les mêmes raisons que traitent les deux politiques précédentes.

Politique des pannes rares

Pour des règles correctives installées sur l'appareil mobile qui serait rarement utilisées depuis le début, celles-ci correspondent à des pannes rares de l'appareil mobile. Ça serait intéressant de les traiter, nous proposons de les négliger en supprimant ces règles, et voir si elles seront réapprises.

III.2.6. Affectation d'actions

Cette partie consiste à récupérer l'ensemble des règles sélectionnées et à leur attribuer une action de correction (voir I.3.3. Administration), nous obtiendrons ainsi l'ensemble des règles correctives utilisable dans la prochaine itération par le module de filtrage. Dans une situation réelle, l'ensemble des règles correctives sera transmis à l'appareil mobile, ces règles vont permettre à l'appareil de corriger les situations problématiques correspondantes. L'intervention d'un expert humain est obligatoire pour la manipulation de ces règles. Pour nos expérimentations, nous ignorons pour l'instant cette partie, du fait que les actions nous importent peu, pour les règles correctives nous ne prenons en compte que la partie condition.

III.3. Démonstration

Au lancement du système et après avoir fixé les paramètres d'exécution, le générateur d'exemples simule le rapport contenant le nombre d'exemples fixé, ceci à l'aide de l'ensemble des règles de simulation (voir I.3.2.1. Générateur d'exemples). Ce rapport qui est considéré jusqu'ici comme le rapport envoyé par l'appareil mobile est directement transmis au module de filtrage où l'ensemble des règles correctives y sera exécutée. Après avoir retiré les exemples couverts par l'ensemble des règles correctives, le module d'apprentissage récupérera le rapport résultant pour y appliquer l'algorithme d'apprentissage. L'apprentissage se fait sous Weka [7] en utilisant l'algorithme d'induction de règles de classification [15], RIPPER (24). L'ensemble des règles apprises passera par le module de sélection pour pouvoir décider des règles qui serviront dans la prochaine itération à filtrer les nouveaux exemples. Nous ignorons pour l'instant la partie affectation d'actions qui nécessite l'intervention d'expert humain, uniquement la partie condition des règles correctives nous intéresse. La Figure III-9 montre l'interface qui nous permettra de lancer le système et de visualiser les résultats obtenues.

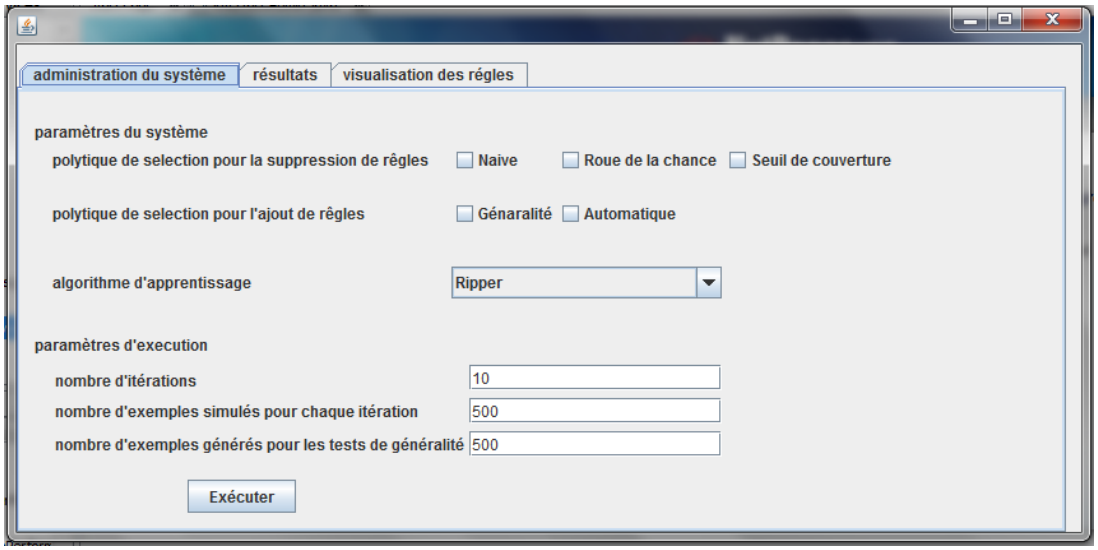


Figure III-9 Administration du système

Le choix des politiques de sélection, l'algorithme d'apprentissage et les différents paramètres d'exécution se fait via *« l'administration du système »*. A l'exécution, et après avoir fixé les différents paramètres, l'onglet *« résultats »* nous permet de visualiser les règles correctives qui ont servi au filtrage à chaque itération, avec pour chacune son historique d'utilisation (voir Administration du système).

Pour voir de plus près les résultats des règles correctives (voir Figure III-10), à chaque itération choisi, la fenêtre de gauche affiche l'ensemble des règles correctives qui ont servis au filtrage, avec entre parenthèse son bilan d'exécution (nombre d'exemples couverts). La fenêtre de droite quant à elle, affiche l'historique d'utilisation de la règle sélectionnée depuis la fenêtre de gauche :

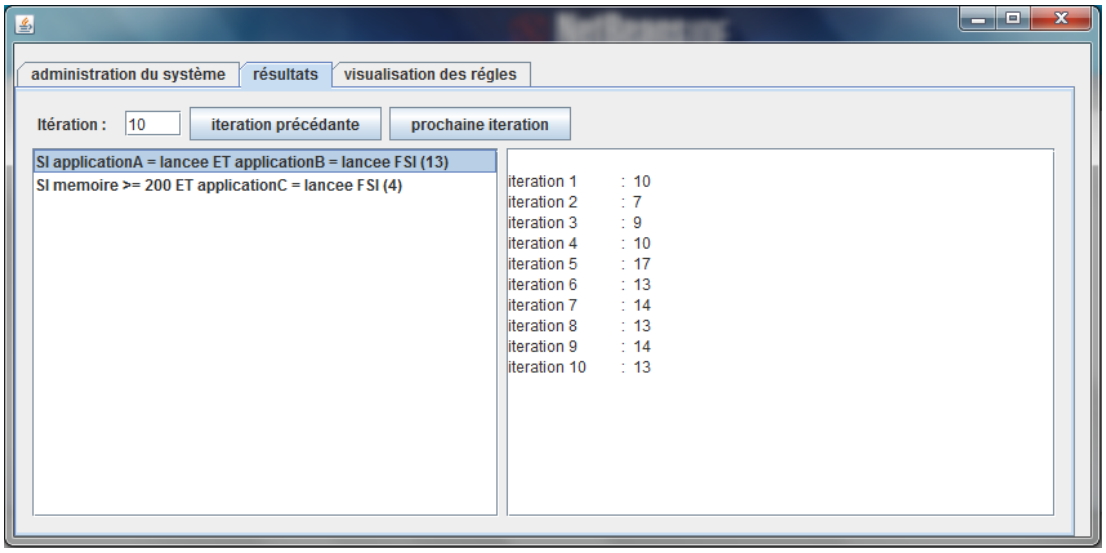


Figure III-10 Visualisation des règles correctives à chaque itération

Le troisième onglet *« visualisation des règles »* permet de voir de plus près l'évolution du système, celui-ci comprends trois fenêtres : la fenêtre en haut à gauche affiche les règles qui ont servis au filtrage, dans la situation réelle cet ensemble serait l'ensemble des règles correctives installées sur l'appareil mobile, celle à droite affiche les nouvelles règles apprises après le filtrage et la troisième fenêtre en bas affiche les règles sélectionnées qui serviront au filtrage à l'itération suivante, qui seront envoyé à l'appareil mobile dans l'architecture vrais vie (voir Figure III-2).

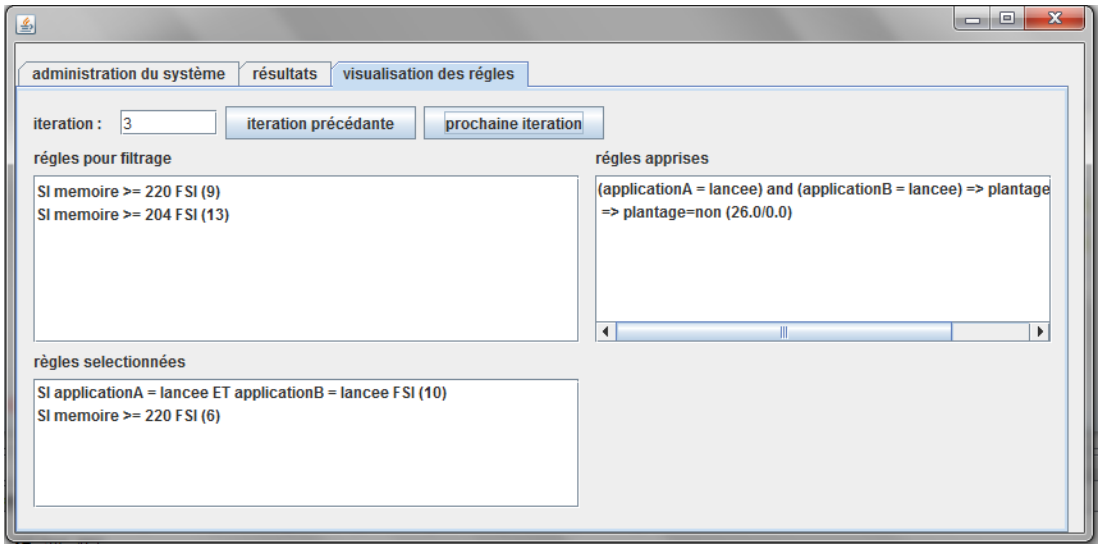


Figure III-11 Visualisation des ensembles de règles

Conclusion

Nous avons présenté dans ce chapitre le système qui a fait l'objet de ce travail, tout en détaillant le fonctionnement de chacun des modules. Pour pouvoir juger la pertinence de notre application, nous testons par la suite le système et nous expliquerons les résultats obtenus. Le chapitre suivant consistera donc à étudier l'efficacité du modèle proposé.

Chapitre 4

Résultats et Discussion

IV. Résultats et discussion

Introduction

Le projet *Manage Yourself* est un projet de diagnostic et surveillance de plates-formes embarquées son objectif est de faire de la prévention de pannes sur mobiles. Nous avons donc conçus un système dans le cadre de ce projet. Nous détaillons le système dans le chapitre *« Modèle propos ».* La prévention de panne se fait en apprenant à partir des rapports envoyés par l'application embarquée au serveur. Ces rapports contiennent la situation de l'appareil mobile avant le plantage. Le but de notre système est de déduire à partir de ces rapports un ensemble de règles correctives qui soit capable de traiter toutes les situations problématiques que l'appareil mobile pourrait rencontrer. Notre travail se situe dans la partie serveur du système, pour pouvoir faire les tests expérimentaux, nous avons simulé le traitement de l'appareil mobile. Les résultats que nous discutons dans ce chapitre sont ceux de l'architecture simulée de notre système (voir III.1.2. Architecture simulé). Le système proposé répond à la problématique suivante :

IV.1. Problématique

Considérons : $\mathbb{C}$: espace de données (exemples que contient le rapport[20] envoyé par l'appareil mobile), $\mathbb{C}^*$: espace de donnée sans plantage et $\mathbb{C}' = \mathbb{C} - \mathbb{C}^*$: espace de donnée avec plantages. L'objectif de notre système est de diminuer l'espace $\mathbb{C}'$ en définissant RC^* : l'ensemble des règles correctives délimitant $\mathbb{C}^*$.

A chaque itération, l'algorithme d'apprentissage déduit à partir des rapports reçus un ensemble de règles R_i délimitant un espace de donnée $\mathbb{C}_i$, cet ensemble sera étiqueté (affectation d'actions) et renvoyé à l'application mobile, nous l'appelons ; l'ensemble RC_i, le but de notre système est que $\mathbb{C}_i$ converge vers $\mathbb{C}^*$, pour cela RC_i

[20] Un exemple de rapport est décrit dans la section : I.3.2.1. Générateur d'exemples

doit converger vers *RC**. La question que traite notre système est : comment RC_i converge vers *RC** de la manière la plus efficace.

IV.2. Solution proposée

Pour répondre au problème posé, nous avons défini une étape de traitement dans notre système que nous avons appelé; module de sélection (voir III.2.5. Sélection). Ce module définit à chaque itération 'i', l'ensemble RC_i (règles correctives) à transmettre au mobile, à partir de R_i (ensemble des nouvelles règles déduites par apprentissage) et RC_{i-1} (règles correctives présentes sur l'appareil mobile). Ce module détermine RC_i en fixant des contraintes sur les deux ensembles (Ri, RCi_{-1}) à travers les différentes politiques de sélection proposées (Voir III.2.5.3. Politiques de sélection)

IV.3. Tests expérimentaux

Nos expérimentations consistent à tester la capacité du système à déduire l'ensemble pertinent des règles correctives ; *RC**. Ainsi que sa capacité à prévenir de nouvelles pannes. Pour notre étude nous utilisons les rapports d'exemples simulant la situation *« Smartphone 4 »*[21] qui traite la mémoire comme une valeur numérique, et qui introduit quelques données bruitées[22]. Nous avons choisis ce concept pour son rapprochement d'une situation réelle d'utilisation. Rappelons les règles de simulations générant ces rapports:

*R1** memoire $\in$ [200, 256] $\wedge$ applicationC=lancee $\Rightarrow$ plantage=oui
*R2** memoire $\in$ [240, 256] $\wedge$ applicationA=lancee $\Rightarrow$ plantage=oui
*R3** applicationA=lancee $\wedge$ applicationB=lancee $\Rightarrow$ plantage=oui

Cet ensemble de règles constitue l'ensemble *RC**. Nous comparons alors à chaque expérience, l'ensemble des règles déduite par le système à *RC**. Rappelons que pour les règles correctives, nous ne tenons compte que de la partie condition. Pour toutes les expériences qui suivent, nous fixons le nombre d'exemples que

[21] Le générateur d'exemples simule plusieurs situations, celles-ci sont détailler dans la partie (I.3.2.1. Générateur d'exemples).
[22] Les données bruitées simulent une panne matérielle : des exemples de plantage couvert par aucune des règles de simulation

contient le rapport simulé pour chaque itération, à 1000 exemples. A noter que les règles déduites pour chaque expérience sont rangé par ordre d'arrivée.

IV.3.1. Testes des politiques d'ajout

Deux politiques d'ajout proposé dans le module de sélection de notre système (voir III.2.5.3. Politiques de sélection) ; « *Ajout automatique »* et *« par généralité »*. Celles-ci traitent les règles apprises avant de les intégrer à l'ensemble des règles correctives.

Ajout automatique

Chaque règle apprise est automatiquement intégrée à l'ensemble des règles correctives. Après 43 itérations, le système n'apprend plus de nouvelles règles, nous déduisons à partir de ça que le système arrive à filtrer toutes les situations problématiques que contiennent les rapports d'exemples simulés. L'ensemble des règles correctives (*RC*) déterminé, est le suivant :

R1 SI applicationA = lancee ET applicationB = lancee FSI
R2 SI memoire ≥ 200 ET applicationC = lancee FSI
R3 SI memoire ≥ 252 FSI
R4 SI memoire ≥ 242 FSI
R5 SI memoire ≥ 240 FSI

Rappelons que nous ne considérons que la partie conditions des règles correctives, nous ne traitons pas dans ce mémoire l'efficacité des actions[23].

En comparant les règles de l'ensemble déduit *RC,* avec ceux de l'ensemble des règles de simulation *RC*,* nous remarquons que le système arrive à apprendre les deux règles *R3* R1*.* Le système n'apprend pas la troisième règle du simulateur *R2** ceci parce les règles déduites *R3*, *R4*, et *R5* sont plus générales que *R2*,* celle-ci retire lors du filtrage, toutes les pannes simulés par *R2*,* et donc *R2** ne peut être apprise par la suite. En regardant les règles de plus près nous constatons la présence d'une règle *R5* couvrant deux autre règles *R3* et *R4*. Dans une situation réelle, l'installation de la règle *R5* annule l'exécution des deux autres vu que celle-ci corrige

[23] Prenons la R2 par exemple, avec action corrective, la règle serait par exemple : SI memoire ≥ 199 ET applicationC = lancee alors viderfichiertemporaires. Nous avons changé 200 par 199 pour éviter le plantage à 200

toute les situations que corrige *R3 et R4*, et donc à ce moment-là, *R3* et *R4* ne feront que surcharger l'appareil mobile, celles-ci ne serviront plus à rien.

Ajout par généralité

Après avoir remarqué dans l'expérience précédente, la présence de règles plus générales que d'autres qui les couvrent et les rendent inutile surchargent le mobile, nous souhaiterons tester la politique qui ne garde que les règles les plus générales à chaque itération. Après 50 itérations le système se stabilise et n'apprend plus de règles. L'ensemble des règles correctives résultant est :

R1 SI applicationA = lancee ET applicationB = lancee FSI
R2 SI memoire ≥ 210 ET applicationA = lancee FSI
R3 SI memoire ≥ 240 FSI
R4 SI memoire ≥ 200 ET memoire ≤ 203 FSI
R5 SI memoire ≥ 204 ET memoire ≤ 204 FSI
R6 SI memoire ≥ 202 ET derniereVersionB = oui FSI

Nous remarquons à première vu qu'il n'y a pas de règle plus générale qu'une autre. Nous remarquons aussi l'apprentissage d'une nouvelle règle *R6* ne faisant pas partie de l'ensemble des règles de simulation, cela prouve la capacité du système à prévenir des situations de plantage. En comparant les règles de l'ensemble déduit *RC,* avec ceux de l'ensemble des règles de simulation *RC*,* nous remarquons que le système arrive à apprendre les deux règles *R3* R2*.* Le système n'apprend pas la troisième règle du simulateur *R1** ceci parce la règle déduite *R3* est plus générales que *R1*,* celle-ci retire lors du filtrage, toutes les pannes simulés par *R1*,* et donc *R1** ne peut être apprise par la suite. Nous avons remarqué la même chose dans l'expérience précédente, ceci montre l'importance d'utiliser les politique de suppression qui permettent le retour arrière ; supprimer une règles pour apprendre une autre plus spécifique, pour à la fin arriver à déduire un ensemble de règles plus fin convergeant vers *RC** l'ensemble pertinent.

IV.3.2. Testes des politiques de suppression

Après avoir prouvé la pertinence de la politique d'ajout par généralité, nous passons ensuite aux tests des différentes politiques de suppression proposées. Nous testons chaque politique séparément ; Deux politiques du même type ne peuvent pas être déclenchés en même temps. Nous déclenchons à chaque expérience la politique d'ajout par généralité.

Expérience 1 : Sélection Naïve

L'ensemble des règles apprises à l'itération 1 est le suivant :

SI applicationA = lancee ET applicationB = lancee FSI
SI memoire ≥ 205 ET applicationC = lancee FSI
SI memoire ≥ 250 FSI

Après 46 itérations le système arrive à déduire un ensemble de règles *RC* plus spécifique que *RC** :

R1 SI applicationA = lancee ET applicationB = lancee FSI
R2 SI memoire ≥204 ET applicationC = lancee FSI
R3 SI memoire ≥ 201 ET applicationA = lancee FSI

En comparant l'ensemble *RC* déduit avec l'ensemble des règles pertinent *RC** nous remarquons que les règles *R1* et *R3** sont identiques, *R2* est plus spécifique que *R1** et *R3* est plus spécifique que *R2**. Le système n'améliore plus par la suite l'ensemble *RC*, à chaque itération le système supprime une règle, celle-ci est tout de suite réapprise dès l'itération suivante.

Expérience 2 : Roue de la chance

L'ensemble des règles correctives déduit dès la première itération est :

SI memoire ≥201 ET memoire ≤ 205 FSI
SI memoire ≥ 210 ET applicationC = lancee FSI
SI applicationA = lancee ET applicationB = lancee FSI
SI memoire ≥ 243 FSI

Nous remarquons à la troisième itération, l'apprentissage de la règle suivante :

SI memoire ≥ 200 ET derniereVersionA = oui ET derniereVersionC = oui FSI

Cette règle ne figure pas dans l'ensemble des règles de simulation, nous déduisant que le système arrive à apprendre de nouvelles règles et donc à prévenir des situations non simulées. A l'itération 69 le système apprend un ensemble de règles *RC* plus spécifique que l'ensemble des règles de simulation *RC**.

R1 SI applicationA = lancee ET applicationB = lancee FSI
R2 SI memoire ≥ 242 ET applicationA = lancee FSI
R3 SI memoire ≥ 213 ET applicationC = lancee FSI

En comparant les deux ensembles (*RC* et *RC**) nous constatons que R1 est identique à *R3**, *R2* et *R3* sont plus spécifique que *R2** et *R1** respectivement. Cet ensemble est amélioré d'une itération à une autre pour stagner à l'itération 81. Par la suite l'ensemble n'est plus amélioré, et donc le meilleur ensemble des règles *RC* déduit en comparant avec *RC** est le suivant :

SI applicationA = lancee ET applicationB = lancee FSI
SI memoire ≥ 240 ET applicationA = lancee FSI
SI memoire ≥ 202 ET applicationC = lancee FSI

Expérience 3 : Seuil de couverture

Les règles apprises dès la première itération sont :

SI applicationB = lancee ET applicationA = lancee FSI
SI memoire ≥ 203 ET applicationC = lancee FSI

Dès la quatrième itération, le système déduit un ensemble de règles correctives *RC* plus spécifique que *RC** :

SI applicationB = lancee ET applicationA = lancee FSI
SI memoire ≥ 203 ET applicationC = lancee FSI
SI memoire ≥ 249 ET applicationA = lancee FSI

A l'itération 52 le système apprend une nouvelle règle qui qui est:

SI memoire ≥ 201 ET derniereVersionB = oui FSI

L'ensemble *RC* des règles correctives déduit à l'itération 4, est amélioré d'une itération à une autre jusqu'à atteindre l'ensemble des règle pertinent *RC**, à l'itération 97:

SI applicationB = lancee ET applicationA = lancee FSI
SI memoire ≥ 200 ET applicationC = lancee FSI
SI memoire ≥ 240 ET applicationA = lancee FSI

Pour la quatrième politique de suppression proposé *« politique des pannes rares »*, nous ne l'avons pas implémenté.

IV.3.3. Comparaison des politiques de suppression

Nous comparons les différents résultats obtenues pour les trois politiques de suppression, en se basant sur les critères suivants : capacité à prévenir de nouvelles pannes, le temps que met le système pour apprendre un ensemble de règles

correctives traitant toutes les pannes simulées, et le temps que met le système pour améliorer cet ensemble afin qu'il converge vers *RC** l'ensemble des règles de simulation.

La politique de suppression Naïve déduit un ensemble de règles correctives *RC* plus spécifique que l'ensemble pertinent *RC** après 42 itérations, cette ensemble n'est plus amélioré par la suite.

La Roue de la chance détermine à la 69eme itération un ensemble de règles correctives *RC* plus spécifique que *RC**, cette ensemble est amélioré d'une itération a une autre jusqu'à l'itération 81.

Pour la politique de suppression par seuil de couverture, celle-ci découvre l'ensemble de règles *RC* plus spécifique que *RC** dès la 4eme itération, cette ensemble est amélioré jusqu'à la 97eme itération où le système arrive à apprendre exactement les mêmes règles que l'ensemble *RC**.

En terme de temps ; la politiques de suppression « par seuil de couverture » est la plus efficace, celle-ci arrive à déduire un ensemble de règles plus spécifique de *RC**, en à peine 4 itérations. En comparant cet ensemble avec les ensembles déduits par les deux autres politiques, on s'aperçoit que c'est l'ensemble le plus proche de l'ensemble *RC**. La politique de suppression « naïve » n'améliore plus l'ensemble déduit. En comparons les capacités à améliorer l'ensemble des règles déduit par les deux politiques ; « Roue de la chance » et « par seuil de couverture », on s'aperçoit que la roue de la chance améliore plus rapidement que la politique de suppression par seuil de couverture, les deux ensembles résultant, sont équivalent à quelque exemples prés, notons que la politique de suppression par seuil de couverture arrive à déterminer un ensemble de règles identique à *RC**.

Nous avons fait varier le nombre d'exemples que contiennent les rapports, et nous avons remarqué que plus celui-ci augmente, plus l'ensemble des règles correctives converge de manière optimal.

Conclusion

Pour notre système qui fait la prévention de pannes sur appareil mobile par apprentissage, nous avons proposé des méthodes de sélection permettant de définir l'ensemble des règles correctives à envoyer au mobile, ces règle une fois installées sur l'appareil mobile, doivent être capable de résoudre tout problème de dysfonctionnement. Nous avons testé ces méthodes de sélections dans ce présent chapitre, dans le but de déterminer celle qui permet de définir le meilleur ensemble de règles correctives. Nous nous sommes aperçus après les différentes expériences menées que toutes les méthodes fournissent des résultats que nous jugeons satisfaisants, ceci dis les meilleur résultats ont été observés pour l'expérience 3 (politique de suppression par seuil de couverture) qui arrive à déduire un ensemble de règles correctives traitant toutes les situations problématique simulé.

Conclusion Générale

Conclusion et perspectives

Nous nous somme intéressé dans ce mémoire aux systèmes de diagnostic et surveillance de plates-formes embarquées. Ce travail entre dans le cadre du projet *Manage Yourself* qui réunit les deux organismes TELELOGOS et l'équipe de recherche DREAM-IRISA de l'université de Rennes 1. Le projet comprend deux grandes parties, d'une part la partie embarquée qui fonctionne sur l'appareil mobile, et d'autre part la partie serveur qui est sur un ordinateur distant. Un premier logiciel a été développé par des étudiants de l'INSA. Et le deuxième où nous intervenons dans la partie serveur, est en cours de réalisation. Après étude du premier logiciel, nous avons fait des constations de mauvais fonctionnement au niveau du serveur que nous améliorons par la suite dans le nouveau logiciel, l'étude de la partie d'apprentissage nous a fait ressentir le besoin d'améliorer cette partie vers l'aspect incrémentale, permettant l'évolution dans le temps du système. Après étude bibliographique sur les méthodes d'apprentissage et après une étude comparative entre arbres de décision et règles de classification, nous avons adopté un apprentissage par règles de classification. Cette méthode construit l'ensemble de règles de manière incrémentale.

Sur le projet 2009-2010, une fois les règles de plantage apprises, des actions correctives sont associées à chacune d'entre elles pour former l'ensemble des règles correctives à embarquer, cela se faisait par un expert humain, ce qui est une tache lourde pour l'administrateur pouvant véhiculer des erreurs de manipulation (voirI.4. Constatations et problématique). Nous avons pour cela, introduit dans la nouvelle architecture (projet 2010-2011) un moyen d'aide à la décision pour aider l'expert humain à sélectionner les meilleures règles correctives à embarquer. Nous l'avons appelé ; *Module de sélection*, ce module propose des mises à jour des connaissances embarquées à partir des nouvelles règles apprises, en fixant des contraintes sur ces deux ensembles, nous avons donné le nom de *Politiques de sélection* à ces contraintes. Nous avons mis en place plusieurs politiques de sélection (voir III.2.5.3. Politiques de sélection) à savoir ; « l'ajout par généralité » qui ne sélectionne que les règles apprises les plus générales, « la suppression naïve » qui supprime la règle

corrective ayant la moyenne d'utilisation la plus élevé tenant compte de l'historique d'utilisation de règles, « roue de la chance » cette politique supprime des règles correctives suivant une lois probabiliste et aléatoire, et enfin la politique « suppression par seuil de couverture » qui supprime les règles correctives dépassant un certain nombre d'exécutions. Nous avons testé le système en activant à chaque expérience la politique d'ajout par généralité pour enrichir l'ensemble des règles correctives, plus une des trois politiques de suppression afin d'affiner cet ensemble vers l'ensemble optimal. Les tests ont montré qu'à chaque expérience l'ensemble des règles correctives déterminé, convergeait vers l'ensemble optimal, cependant la politique qui déduisait l'ensemble de règles correctives le plus proches de l'ensemble optimal, comparé aux autres politique, est la « suppression par seuil de couverture » associé à « l'ajout par généralité ».

Perspectives

La continuité de notre travail serait de prendre en considération plusieurs aspects ; un premier point et qui serait intéressent de développer, est l'automatisation complète du système, jusqu'ici l'affectation d'actions correctives se fait par un expert humain, il serait idéal de pouvoir intégrer dans l'architecture un moyen d'étudier l'efficacité des actions de correction pour pouvoir les affecter aux règles apprise sans l'intervention d'un tiers.

Pour notre étude nous avons utilisé un générateur d'exemples qui simule des rapports de plantage via un ensemble de règles de simulation, cet ensemble reste fixe tout le long du travail c'est-à-dire simulant toujours les mêmes situations, il serait intéressent que ces situations changent à chaque itération c'est dire à chaque découverte de nouvelles règles correctives ; si par exemple une situation vient d'être corriger il ne serait plus logique de la simuler dans le rapport qui sera exploiter dans la prochaine itération.

Une perspective intéressante pour le projet *Manage Yourself* serait de s'intéresser au profile des utilisateurs, ceci en personnalisant le type de correction à apporter, ces correction seront adaptées aux habitudes de chaque type d'utilisateur, un utilisateur qui n'utilise son Smartphone que pour appeler n'a pas besoin de correctifs traitant les différentes applications installées.

Un autre point à étudier, est la gestion de valeurs manquantes ; les rapports de situation qu'envoie l'application embarquée peuvent contenir des données manquantes qui se sont par exemples perdues accidentellement lors du transfert, il serait intéressant de pouvoir mettre en place une machine learning permettant de prendre en considération cet aspect.

Pour le système de diagnostic et surveillance développé dans ce mémoire, nous nous somme intéressé beaucoup plus aux exemples de plantage, il serait intéressent d'étudier les exemples de non plantage afin d'essayer de comprendre les pannes.

Bibliographie

1. **O.Corridor, R.Boillon, Q.Decré, V.Le Biannic, G.Lemasson, N.Renaud, F.Tollec, L.Rozé.** *Systeme Expert pour Smartphones: Rapport de Pré-étude. INSA, Rennes* 2011.

2. **E.Alibert, F.Barbedette, P.Chesneau, M.Deshayes, S.Hartunians, M.Poignet, L.Rozé.** *Manage Yourself: Rapport de pré-étude. INSA, Rennes* 2009.

3. **E.Alibert, F.Barbedette, P.Chesneau, M.Deshayes, S.Hartunians, M.Poignet, L.Rozé.** *Manage Yourself: Rapport de spécifications. INSA, Rennes* 2009.

4. **E.Alibert, F.Barbedette, P.Chesneau, M.Deshayes, S.Hartunians, M.Poignet, L.Rozé.** *Manage Yourself: Rapport final. INSA, Rennes* 2009.

5. **J.R.Quinlan.** *C4.5 : programs for machine learning* . 1993.

6. Weka documentation.
[http://weka.sourceforge.net/doc/weka/classifiers/rules/package-summary.html]

7. **J.R.Quinlan.** *Induction of Decision Trees.* 1985, Machine Learning, Vol. 1, pp. 81-106.

8. **D.MacKay.** *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003 .

9. Machine learning. *Wikipédia, the free encyclopedia.*
[http://en.wikipedia.org/wiki/Machine_learning]

10. **H.M.Pandey.** *Design Analysis and Algorithm.* Machine Learning, Vol. 2, pp.1 1-112.

11. **S.Tuffery.** *Data Mining et statistique décisionnelle. L'intlligence des données*

12. **L.Breiman, J. H. Friedman, R.A.Olshen, and C.J.Stone.***Classification and Regression Trees,* Wadsworth 1984.

13. **G.CALAS.** *Etudes des principaux algorithmes de data mining: Spécialisation Sciences Congnitives et Informatique Avancée,.*

14. **R.Gilbert.** *CHAID and Earlier Supervised Tree Methods.* 2010.

15. **R.Rakotomalala.** *Induction de règles prédictives.* Laboratoire ERIC, Université Lyon 2.

16. **F.Witten.** *Generating Accurate Rule Sets without Global Optimization.* 1998, ICML, pp. 144-151.

17. **P.Clark, T.Niblett.** *The CN2 Induction Algorithm.* Machine Learning, Vol. 3, 1989.

18. **F.Thabtah, P.Cowling.** *Mining the data from a hyperheuristic approach using associative classification.* 2008, Expert Systems with Applications, Vol. 34, pp. 1093–1101.

19. **D.J.GAY.** *Calcul de motifs sous contraintes pour la classification supervisée.* thèse de doctorat, Institut National des sciences appliquées de Lyon, L'université de la nouvelle calédonie. Thèse de doctorat, 2009 .

20. **J.Widmer, G.Furnkranz.** *Incremental reduced error pruning.* ICML, 1994, pp. 70-77.

21. **P.E.Utgoff.** *Incremental induction of decision trees.* Machine Learning, Vol. 4, pp. 161–186. 1989,

22. **C.Decaestecker.** *Decision tree.* Machine Learning, Vol. 5, pp. 110–125. 1990,

23. **C.Gaines.** *Induction of Ripple Down Rules Applied to Modeling Large Databases.* 1995, Journal of Intelligent Information Systems, Vol. 5, pp. 211-228.

24. **W.Cohen.** *Fast effective rule induction.* Machine Learning.

25. **M.Friedl.** *Decision tree Classification of land Cover From Remotely Sensed Data.* 1997.

26. **R.Rakotomalala.** *Arbre de Décision.* Laboratoire ERIC Université Lumière Lyon 2.

27. **I.H.Witten, E.F.Mark A.Hall.** *Data Mining: Practical Machine Tools ans Techniques.*

28. **F.Luger, George.** *Artificial intelligence : structures and strategies for complex problem solving* .

29. **J.Fisher, C. Schlimmer and H.Douglas.** *A case study of incremental concept induction.* 1986, AAAI, pp. 496-501.

30. **A.Cornuéjols, L.Miclet, Y.Kodratoff.** *Apprentissage artificiel: Concepts et algorithmes.* Eyrolles, 2002.

31. **R**.**Trépos.** *Apprentissage symbolique a partir de données issus de simulation pour l'aide à la décision. Gestion d'un bassin versant pour une melleur qualité de l'eau. Thèse de Doctorat* , 2008.

32. **I.H. Witten, E.Frank, L.Trigg, M.Hall, G.Holmes, S.J.Cunningham.** *Weka: Practical Machine Learning Tools and Techniques.*

33. **A.Osmani.** *Arbre de décision – Evaluation de l'apprentissage - Elagage.*

Résumé

Le projet ManageYourSelf dans lequel intervient le travail de ce mémoire, a pour objectif d'augmenter la robustesse et l'adaptativité d'un ensemble de logiciels embarqués. L'idée est d'abord de doter ces matériels mobiles de moyens d'autosurveillance afin de pronostiquer un dysfonctionnement et y remédier localement pour maintenir au mieux la qualité de service attendue. Ces outils d'autosurveillance, qui incluent pronostic, détection et réparation, sont embarqués et doivent être régulièrement remis à jour afin d'améliorer leur comportement. Des informations sur le contexte des dysfonctionnements sont transmises régulièrement, sous forme de rapports de fonctionnement, à un serveur. Il s'agit alors de capitaliser ces informations, de les analyser et d'améliorer, par apprentissage, les connaissances existantes, avant de les redistribuer, de manière pertinente, aux différents éléments de la flotte. L'objectif est d'actualiser les connaissances distribuées afin d'améliorer la robustesse et les performances des logiciels embarqués. Un premier logiciel a été développé par un ensemble d'étudiant de l'INSA de Rennes, nous avons étudié le fonctionnement du serveur, où nous avons constaté des problèmes de fonctionnement. Pour le deuxième logiciel dans lequel nous intervenons, nous avons modifié l'architecture du serveur en proposant des méthodes d'analyse et d'apprentissage afin de tirer parti, de manière incrémentale, des connaissances récupérées au niveau du serveur. Nous avons aussi mis en place un système d'aide à la décision permettant la mise à jour des connaissances embarquées à partir des connaissances apprises. Quant à la partie embarquée du nouveau système, celle-ci est assuré par un ensemble d'étudiants de l'INSA.

Mots clés : diagnostic, surveillance, apprentissage incrémental, système d'aide à la décision.

Abstract

The project ManageYourSelf which involve the work of this work, aims to increase the robustness and adaptivity of a set of embedded software. The idea is first to provide mobile equipment such means of self-monitoring to predict and remedy malfunctions locally to maintain the best quality of service expected. The tools of self-monitoring, which include prognosis, detection and repair, are embarked and must be regularly updated to improve their behavior. Information about the context of malfunctions is provided regularly, as reports of operation, to a server. Processing at the server is to collect information, analyze and improve by learning, existing knowledge, before redistributing them so relevant to the various elements of the fleet. The goal is to update the distributed knowledge to improve the robustness and performance of embedded software. The first software was developed by a group of students from INSA in Rennes; we studied the operation of the server, where we found the optimization problems. For the second program in which we operate, we changed the server architecture by providing methods of analysis and learning to build knowledge incrementally retrieved at the server. We also set up a system of decision support for updating embedded knowledge from the knowledge learned. The embedded software of the new system (Manage Yourself 2010-2011), is provided by a group of students from.

Key words: diagnostic, monitoring, incremental learning, decision support system.

ملخص

المشروع "مناج يور سلف" يهدف إلى زيادة متانة وقابلية التكيف لمجموعة من البرمجيات المدمجة. الفكرة هي أولا توفير لهذه الأجهزة المتنقلة, وسائل المراقبة الذاتية للتنبؤ ومعالجة الأعطال محليا للحفاظ على أفضل نوعية الخدمة المتوقعة. يتم شحنها أدوات الرصد الذاتي، والتي تشمل التشخيص والكشف والإصلاح، ويجب أن يتم تحديثه بشكل منتظم لتحسين تصرفاتهم. يتم توفير المعلومات حول سياق الأعطال على شكل تقارير بصفة منتظمة إلى ملقم. مسألة تكديس هذه المعلومات وتحليلها وتحسين طريقة التعلم المعرفة القائمة، وذلك قبل إعادة توزيعها إلى مختلف عناصر الاسطول .الهدف من ذلك هو تحديث المعرفة الموزعة لتحسين قوة وأداء البرمجيات تم تطوير أول برنامج من قبل مجموعة من الطلاب من المعهد الوطني للعلوم التطبيقية INSA في رين، حيث وجدنا مشاكل وضيفيه .في البرنامج الثاني قمنا بتغيير بنية الملقم من خلال توفير طرق التحليل والتعلم, من أجل بناء وبشكل متزايد،استرجاع المعارف في الملقم, ووضعنا ايضا نظام دعم القرار لاستكمال المعرفة المضمنة من المعارف المكتسبة .

الكلمات الرئيسية :التشخيص، المراقبة، تعلم تدريجي، نظام دعم القرار.