Requêtes discriminantes pour l'exploration de données : Application à l'astrophysique

Julien Cumin, Jean-Marc Petit, Fabien Rouge, Christian Surace

Requêtes discriminantes pour l'exploration des données : Application à l'astrophysique

Julien Cumin[1] Jean-Marc Petit[1,2] Fabien Rouge[1]
Christian Surace[3]

[1]INSA Lyon, Villeurbanne
[2]LIRIS (UMR 5205 CNRS)
[3]LAM, CNRS, Marseille

July 3, 2015

Abstract

À l'ère du Big Data, il est essentiel de pouvoir explorer les données dont on dispose afin d'en faire éventuellement ressortir des connaissances nouvelles.

De part la diversité grandissante des profils d'utilisateurs, qui n'ont souvent qu'un vernis informatique, et la complexité des données accumulées, il est de plus en plus difficile de procéder à cette phase d'exploration.

Nous nous plaçons dans un contexte où un analyste a à sa disposition des données gigantesques accessibles en SQL.

Nous proposons de l'aider à formuler ses requêtes par une approche de réécriture (ou reformulation) répondant à cette nécessité d'exploration rapide et intuitive des données massives. Pour cela, nous introduisons la notion de *requêtes discriminantes*, une restriction syntaxique de SQL qui oblige de spécifier une condition de sélection qui permet de dissocier des exemples positifs et négatifs. Intuitivement, nous cherchons à construire un petit jeu d'apprentissage dans lequel les exemples positifs correspondent aux résultats souhaités par l'analyste et les exemples négatifs (les contre-exemples) sont ceux qu'il ne veut pas. Nous pouvons alors reformuler la requête initiale en utilisant des techniques d'apprentissage automatique. Nous proposons aussi des mesures pour évaluer la qualité de la réécriture, notamment vis-à-vis de sa diversité en terme de tuples retournés.
Un prototype, nommé iSQL, a été développé pour mettre en œuvre cette approche. Nous l'avons expérimenté sur des bases de données issues de l'astrophysique afin d'évaluer sa pertinence dans un cadre d'exploration des données, avec des résultats préliminaires très encourageants.

1 Introduction

Le phénomène *Big Data* est au centre de toutes les préoccupations des mondes scientifiques et économiques. La donnée, encore plus que par le passé, est au cœur des dynamiques de toutes les entreprises et instituts, de part son potentiel intrinsèque de connaissances nouvelles et donc de valeur pour ceux qui la détiennent.

L'un des aspects encore relativement confidentiel des difficultés du Big Data porte sur l'exploration interactive des données. Nous nous plaçons dans un cadre de données scientifiques relevant du Big Data et accessibles en SQL, comme c'est encore communément le cas dans de nombreux domaines, de l'astrophysique à la biologie. Sur les données scientifiques plus précisément, il est commun d'avoir des tables avec plusieurs centaines d'attributs, la plupart avec des valeurs numériques issues de mesures physiques. Définir des seuils adéquats sur ces attributs numériques dans une condition de sélection ou être sûr de ne pas manquer une condition simple sur tel ou tel attribut peut tourner au cauchemar pour les analystes de données. Dans ce contexte, l'écriture des requêtes SQL – aussi triviale soit elle sur des exemples simples – est un problème en soi et nécessite un haut niveau d'expertise. Les analystes de données sont alors confrontés à la difficulté de formuler les "bonnes" requêtes sur leurs données.
Si les problématiques de stockage et de débits sont importantes, celles d'exploration et d'exploitation des données le sont encore plus. Par ailleurs, un simple parcours de table gigantesque peut être rédhibitoire, rendant l'écriture de requêtes SQL parfois plus difficile. Par ailleurs, un analyste ne peut interpréter

EMP	Empno	Lastname	Workdept	Job	Educlevel	Sex	Sal	Bonus	Comm	Mgrno
	10	SPEN	C01	MANAGER	18	F	52750	900	4220	20
	20	THOMP	-	MANAGER	17	M	41250	800	3300	-
	80	KWAN	-	FINANCE	20	F	38250	950	3060	10
	50	GEYER	-	MANAGER	16	M	40175	850	3214	20
	60	STERN	D21	SALE	14	M	32250	1000	2580	30
	70	PULASKI	D21	SALE	16	-	36170	700	2893	100
	90	HENDER	D21	SALE	12	F	29750	500	2380	10
	100	LAUREL	C01	FINANCE	18	-	26150	800	2092	50

Figure 1: Exemple illustratif

qu'un nombre réduit de résultats d'une requête, il est donc important de travailler sur un échantillon pour arriver à une "bonne" requête qui, elle, pourra être exécutée sur l'ensemble des données.

Exemple 1 Quoi qu'il soit difficile d'élaborer un exemple concis relevant du Big Data, nous considérons l'exemple donné en figure 1 avec une seule relation, *EMP* [6]. L'attribut *Educlevel* représente le nombre d'années d'éducation formelle, *Sal* le salaire annuel, *Bonus* le bonus annuel et *Comm* la commission annuelle. La signification des autres attributs est immédiate.

Imaginons que nous cherchons les femmes qui ont un bonus supérieur à leur responsable hiérarchique. On obtient que SPEN et KWAN vérifient cette condition.

L'idée que nous allons développer dans ce papier est de dire que nous aurions pu écrire cette requête comme suit :

```
Select Empno, Lastname, Bonus
From EMP
Where EducLevel > 17
```

Plusieurs choses sautent au yeux:

- Cette requête est très différente de la requête initiale : elle fait intervenir un nouvel attribut `EducLevel` et ne comporte pas d'imbrication.
- Elle est plus efficace : un seul parcours de la table est nécessaire. Sur des très gros volume de données, c'est primordial.
- Elle ne lui est pas équivalente, introduisant une forme de diversité dans les résultats : le dernier tuple, LAUREL, se retrouve dans le résultat de la nouvelle requête.

$\Diamond$

Problème abordé Le problème consiste donc à construire une interaction permettant à un analyste d'explorer ses données, sans supposer qu'il ait des connaissances avancées en bases de données. Le processus doit être itératif afin non seulement de le guider mais aussi de prendre en compte ses retours pour guider le processus lui-même. En effet, l'intuition de l'homme apparaît aussi importante que les outils techniques pour analyser les données. Aujourd'hui, les principaux outils d'analyse ou de fouille de données manquent d'interactions et apparaissent comme des tunnels à l'intérieur desquels il n'est pas possible de savoir ce qu'il se passe. Le pré-traitement des données est supposé fait (bien souvent avec SQL) et ensuite seulement, les techniques de fouille de données peuvent se mettre en œuvre avec un outil différant, obligeant de passer d'un environnement à l'autre sans réelle continuité. Nous pensons qu'il est très utile de rapprocher ces deux étapes et de permettre une interaction beaucoup plus fluide entre l'écriture de la requête SQL et la fouille de données.

Plus précisément, nous posons une question très simple, qui ne semble pas avoir été traitée dans le passé. Etant donnée une requête Q sur une base de données d, on peut facilement définir des exemples positifs à partir des réponses de Q sur d, i.e. ils correspondent aux tuples recherchés par l'analyste. La question est alors de définir ce que serait un exemple négatif, i.e. des tuples que l'analyste ne voudrait pas voir apparaitre dans le résultat. Autrement dit, comment définir une requête $\overline{Q}$ qui s'apparenterait à la négation de Q ?

Si la génération de cette requête $\overline{Q}$ s'avère possible, nous disposerions alors d'un ensemble de tuples *exemples*, résultats de l'évaluation de Q, et d'un ensemble de tuples *contre-exemples*, issus de l'évaluation de $\overline{Q}$. Il est alors très naturel dans le cadre de la fouille de données d'utiliser ces deux ensembles dans une

approche d'apprentissage supervisé de motifs sur les tuples. En particulier, nous considérons les approches *d'arbres de décisions*, pour lesquels les modèles appris sont directement interprétables en conditions de sélection SQL. En alimentant un algorithme d'arbre de décisions avec nos deux ensembles de tuples dans un processus d'apprentissage supervisé, il semble alors possible d'aboutir à une condition de sélection traduisant les motifs découverts sur les tuples, et qui discrimine les exemples des contre-exemples, d'une manière toute autre que la condition de sélection initiale de l'utilisateur.

Contributions du papier Nous introduisons une nouvelle condition syntaxique dans une requête SQL qui permet de définir les exemples positifs et négatifs. A partir d'une telle requête, nous montrons qu'il est facile de construire un jeu d'apprentissage à partir duquel nous pouvons proposer automatiquement à l'analyste une nouvelle requête. Cette requête générée par le système peut être intéressante pour l'analyste dans son processus exploratoire, puisque les résultats de cette requête forment un ensemble proche mais néanmoins incluant de nouveaux tuples qui n'étaient pas à priori accessibles directement. Le processus peut alors se répéter un certain nombre de fois, jusqu'à ce que l'analyste soit satisfait du résultat obtenu. Nous proposons aussi des mesures pour évaluer la qualité de la réécriture, notamment vis à vis de sa diversité en terme de tuples retournés.
Un prototype, nommé iSQL, a été développé pour mettre en œuvre cette approche. Nous l'avons expérimenté sur des bases de données issues de l'astrophysique afin d'évaluer sa pertinence dans un cadre d'exploration des données. Nous détaillons notre validation sur des données issues du projet européen *CoRoT*[1], pour COnvection, ROtation et Transits planétaires, destinées à l'étude de la structure interne des étoiles et à la recherche d'exoplanètes extrasolaires. Les résultats préliminaires que nous avons obtenus sont très encourageants et démontrent que cette idée peut être très utile pour des analystes de données pour répondre à un des défis du Big Data.

Organisation du papier La section 2 présente un état de l'art sur les approches d'exploration des données à l'échelle du big data. La section 3 introduit les préliminaires et la notion de requêtes discriminantes et de leur négation. La réécriture proprement dite est décrite dans la section 4. La section 5 présente l'implémentation que nous avons réalisée et la section 6 la validation sur des données astrophysiques. Une conclusion et une discussion sont proposées en section 7.

2 Requêtes discriminantes et leur négation

2.1 Préliminaires

Nous introduisons brièvement les notations utilisées dans la suite de ce papier, voir [1] pour les détails. On considère une base de données (BD) d définie sur un schéma de BD R et une requête Q sur R. On note $ans(Q, d)$ le résultat de l'évaluation de Q sur d. On suppose aussi que les données peuvent comporter des valeurs nulles et qu'elles sont toutes de type réel pour simplifier la présentation. En pratique, les principaux types de données sont acceptés, voir Section 4.

On considère l'algèbre relationnelle comme langage relationnel ou de façon équivalente leur représentation en SQL sur des exemples. On supposera que les requêtes sont de la forme suivante : $Q = \pi_X(\sigma_F(r_1 \bowtie \ldots \bowtie r_n))$[2]. Les formules de sélection F sont définies par induction comme usuellement à partir des opérateurs logiques $\neg, \wedge, \vee$ et des parenthèses (). On admettra comme formule atomique les expressions de la forme $A\theta B, A\theta v$, A `IS NULL` ou A `IS NOT NULL` où A, B sont des attributs, v une valeur réelle et θ un opérateur binaire usuel $(<, \leq, >, gel, =)$. On note $attr(F)$ l'ensemble des attributs apparaissant dans une formule de sélection F.

2.2 Requêtes discriminantes

Les langages relationnels (calcul, datalog, algèbre) fournissent un cadre rigoureux permettant d'exprimer des requêtes sur des bases de données relationnelles. La question qui nous intéresse ici est de définir l'ensemble des tuples qui ne vérifient *pas* une requête Q. Les langages relationnels logiques nous enseignent alors qu'il faut être prudent : la négation d'une formule peut nous faire perdre l'*indépendance du domaine* et donner des requêtes dont le résultat est infini. Accessoirement, en pratique, nous ne pourrions pas les exprimer en SQL !

[1] `http://smsc.cnes.fr/COROT/`

[2] En pratique, nous pourrions avoir des requêtes plus générales admettant par exemple des requêtes imbriquées. Les résultats que nous proposons restent valides, mais leur étude est cependant hors de portée de cet article.

Nous proposons donc de simplifier ce problème en introduisant de façon relativement triviale une nouvelle condition syntaxique sur les formules de sélection. Cette restriction syntaxique étant très simple, nous gardons une grande expressivité sur la classe de requêtes traitées.

La condition syntaxique vise la formule de sélection F : F doit s'exprimer comme une conjonction de deux formules, $F = F_1 \wedge F_2$, où F_1 est appelée *condition de discrimination* et F_2 une condition classique.

Définition 1 Une *condition de discrimination* F est une condition de sélection classique, excepté qu'elle ne comporte pas de formule atomique testant les valeurs nulles.

La présence de valeurs nulles ne permettraient pas de dissocier exemples et contre-exemples et sont donc exclues.

Définition 2 Une requête est dite *discriminante* si celle-ci s'exprime sous la forme $Q = \pi_X(\sigma_{F_1 \wedge F_2}(r_1 \bowtie \ldots \bowtie r_n))$ avec F_1 une condition discriminante et F_2 une formule quelconque, éventuellement vide.

Les exemples positifs, notés $E_+(Q)$ sont issus du résultat de l'évaluation de la requête discriminante. Pour garder tous les attributs possibles, nous supprimons juste la projection finale. On a donc : $E_+(Q) \subseteq \sigma_{F_1 \wedge F_2}(r_1 \bowtie \ldots \bowtie r_n)$.

Exemple 2 Sur l'exemple de l'introduction, il faut réécrire la requête pour la rendre compatible avec notre formalisation. On aurait :

```
Select E1.Empno, E1.Bonus
From EMP E1, EMP E2
Where E1.Sex = 'F' and
E1.Bonus > E2.Bonus AND E1.Mgrno=E2.Empno
```

Sur cet exemple, la condition de discrimination serait `Sex = 'F'` et les exemples positifs seraient les deux tuples de SPEN, KWAN. ◊

2.3 Négation de requêtes discriminantes

Nous sommes maintenant armés pour définir la négation de ces requêtes discriminantes. L'introduction de la restriction syntaxique rend cette étape très simple.

Définition 3 Soit $Q = \pi_X(\sigma_{F_1 \wedge F_2}(r_1 \bowtie \ldots \bowtie r_n))$ une requête discriminante sur un BD d. La négation de Q, notée $\overline{Q}$, est définie par :

$$\overline{Q} = \sigma_{\neg(F_1) \wedge F_2}(r_1 \bowtie \ldots \bowtie r_n)$$

On peut noter :

- F_1 est considérée comme étant la clause de discrimination des exemples et $\neg F_1$ celle des contre-exemples.
- F_2 est commune à Q et $\overline{Q}$. Elle permet de restreindre l'ensemble des tuples utilisables en tant qu'exemples et contre-exemples.
- Nous ne projetons plus le résultat sur X afin de garder tous les attributs possibles pour discriminer les valeurs.

De façon analogue aux exemples positifs, les exemples négatifs sont notés $E_-(Q)$ et vérifient : $E_-(Q) \subseteq ans(\overline{Q}, d)$

Exemple 3 Sur l'exemple précédent, la négation de la requête donnerait :

```
Select E1.*
From EMP E1, EMP E2
Where NOT (E1.Sex = 'F') and
E1.Bonus > E2.Bonus AND E1.Empno=E2.Mgr
```

Les employés GEYER, STERN sont considérés comme des exemples négatifs. ◊

On peut remarquer que l'on a les propriétés simples suivantes sur l'ensemble des tuples n'ayant pas de valeurs nulles sur les attributs de F_1.

Empno	Lastname	Workdept	Job	Educlevel	Sal	Bonus	Comm	Mgrno	Class
10	SPEN	C01	MANAGER	18	52750	900	4220	20	+
80	KWAN	-	FINANCE	20	38250	950	3060	10	+
50	GEYER	-	MANAGER	16	40175	850	3214	20	-
60	STERN	D21	SALE	14	32250	1000	2580	30	-

Figure 2: Jeu d'apprentissage

Propriété 1 Soient $F_3 = \wedge_{A \in attr(F_1)} A$ `IS NOT NULL` et $J = r_1 \bowtie \ldots \bowtie r_n$.
On a :

- $\sigma_{F_1}(J) \subseteq \sigma_{F_3}(J)$.
- $\sigma_{\neg(F_1)}(J) \subseteq \sigma_{F_3}(J)$.
- $\sigma_{F_1}(J) \cup \sigma_{\neg F_1}(J) \subseteq \sigma_{F_3}(J) \subseteq J$.

Les tuples de $J \setminus \sigma_{F_3}(J)$ sont des tuples pour lesquels il existe au moins une valeur nulle sur les attributs de la condition discriminante.

Exemple 4 Sur l'exemple introductif, l'avant avant dernier et le dernier tuple sont dans ce cas. $\Diamond$

En pratique, on peut aisément imaginer soit d'étendre la syntaxe de SQL en ajoutant une clause de discrimination, soit concevoir une interface graphique via des formulaires séparant F_1 et F_2 pour spécifier ces requêtes discriminantes (voir Section 4).

2.4 Construction d'un jeu d'apprentissage

Nous pouvons maintenant construire un jeu de données d'apprentissage. Il est important de noter que nous contrôlons absolument tout le processus. En fonction de la taille des données manipulées, nous pourrons choisir un échantillon. L'idée n'est pas de précisément répondre à la requête, mais bien d'aider l'analyste à formuler une requête qui corresponde mieux à ses attentes. Ces étapes de tâtonnement ne nécessitent pas de prendre en compte toutes les données. Une étude précise des garanties que nous pourrions fournir pour l'apprentissage est hors de la portée de cet article.

Définition 4 Etant donnée une requête discriminante Q, sa négation $\overline{Q}$, un *jeu d'apprentissage* est défini sur le schéma de $(r1 \bowtie \ldots \bowtie r_n) \setminus attr(F_1) \cup Class$. Ses tuples sont issus de $E_+(Q)$ (resp. $E_-(Q)$) avec l'ajout de la valeur $+$ (resp. $-$) pour l'attribut `Class`.

On ne garde pas les attributs de F_1 pour ne pas apprendre la condition déjà exprimée dans la requête discriminante.

Exemple 5 En continuant l'exemple, on obtient le jeux décrit dans la Figure 2.

L'attribut `Sex` a été supprimé et l'attribut `Class` a été ajouté avec les labels correspondants. $\Diamond$

Cette étape peut facilement s'implémenter en SQL, voir la Section 4. Sous réserve de disponibilité des données, on peut aussi contrôler le nombre d'exemples positifs et négatifs à intégrer dans le jeu d'apprentissage.

3 Réécriture de requêtes discriminantes

Nous présentons dans cette partie la phase d'apprentissage automatique de motifs sur les tuples de E_+ et E_-, et leurs transcriptions en condition de sélection relationnelle. Nous apportons également différentes métriques permettant d'évaluer la qualité de la réécriture de la requête initiale par ce processus.

3.1 Rappel sur les arbres de décision

Un arbre de décision est une méthode classique de l'apprentissage automatique, dont le but est de construire un arbre (au sens de la théorie des graphes) permettant de décider une variable de classe en fonction des valeurs de variables d'entrées. Le parcours de l'arbre de la racine jusqu'à une feuille constitue la conjonction des conditions du modèle sur les variables d'entrées pour aboutir à la feuille, labellisée de la classe à décider pour cette conjonction.

La construction supervisée du modèle (i.e. la structure de l'arbre et les conditions posées sur chaque nœud interne) à partir d'un ensemble d'apprentissage est dépendante des algorithmes utilisés et sort du

cadre de cet article. La méthode utilisé dans le prototype iSQL présenté dans la suite est l'algorithme *C4.5* [11].

3.2 Apprentissage supervisé de condition de sélection

À partir d'un arbre de décision, il est relativement direct de construire une condition de sélection relationnelle en le parcourant en profondeur. En effet, une branche (un parcours direct de la racine à une feuille) peut être vue comme une conjonction de conditions booléennes sur les valeurs des attributs d'un tuple, pour laquelle la classe du tuple est celle dont est labellisée la feuille de la branche. L'ensemble des branches aboutissant à la classe des tuples positifs de E_+ peut donc être vu comme une disjonction des clauses conjonctives issues de ces branches, et donc utilisé comme nouvelle condition de sélection relationnelle.

Définition 5 Soient r_0 un jeu d'apprentissage issu d'une requête discriminante Q avec une formule de sélection de la forme $F_1 \wedge F_2$ et AD l'arbre de décision appris à partir de r_0. Soit b une branche positive de AD (de la racine à une feuille labellisée $+$). On note

$$F_{new} = \bigvee_{b \in AD} \bigwedge_{e \in b} e$$

où e est de la forme $A_i \; bop \; v$, avec A_i un attribut de r_0 qui n'est pas dans $attr(F_1)$, bop est un opérateur binaire usuel et v une valeur numérique.

Exemple 6 Sur l'exemple précédent, un arbre de décision va trouver la condition '`Educlevel` > 17". ◊

Définition 6 Soit Q une requête discriminante. La nouvelle requête issue de Q, notée ${}^tQ$, est définie par:

$${}^tQ = \pi_X(\sigma_{F_{new}}(r_1 \bowtie \ldots \bowtie r_n)).$$

On l'appellera *requête transmutée* par la suite.

Cette nouvelle requête a un certain nombre d'atouts, comme expliqué dans l'introduction :

- La condition de sélection est complètement nouvelle : elle peut porter sur des attributs non identifiés comme utiles dans la requête initiale.
- La requête est nécessairement plus simple et plus rapide en SQL: les requêtes imbriqués avec par exemple les constructeurs `EXIST` ou `bop ANY` sont mécaniquement simplifiées par une seule sélection (un seul parcours des données).

On peut noter que nous avons une *propriété d'absorption* avec cette nouvelle requête vis à vis des auto-jointures.

Exemple 7 Sur l'exemple, la requête transmutée est celle donnée dans l'introduction avec la condition de l'exemple précédent. ◊

3.3 Mesure de qualité entre requête initiale et requête transmutée

S'il est difficile de garantir des relations précises entre une requête initiale Q et sa réécriture ${}^tQ$, puisque celle-ci dépend des motifs découverts dans la phase d'apprentissage, on peut en revanche illustrer graphiquement les ensembles de tuples mis en jeu au cours du processus de réécriture ainsi que les résultats souhaitables sur ces ensembles.

Nous présentons sur la Figure 3 les ensembles de tuples suivants :

- $\boldsymbol{Z}$: ensemble des tuples de $r_1 \bowtie \ldots \bowtie r_n$.
- $\boldsymbol{ans(Q, d)}$: ensemble des tuples résultat de la requête initiale Q de l'utilisateur dans d.
- $\boldsymbol{ans(\overline{Q}, d)}$: ensemble des tuples résultat de la négation $\overline{Q}$ de Q dans d.
- $\boldsymbol{E_+}$: ensemble des tuples positifs.

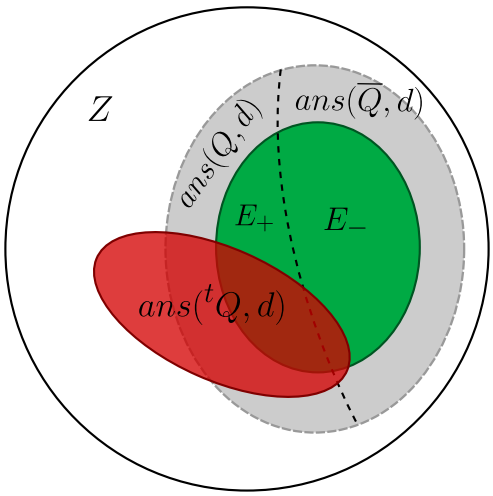


Figure 3: Ensembles de tuples manipulés dans le processus de réécriture

- $\boldsymbol{E_-}$: ensemble des tuples négatifs.
- $\boldsymbol{ans(^tQ, d)}$: Ensemble de tuples résultat de la nouvelle requête ^tQ dans d.

Pour simplifier les notations, on notera $Q, \overline{Q}, ^t Q$ à la place de $ans(Q, d), ans(\overline{Q}, d)$ et $ans(^tQ, d)$. On note aussi la cardinal d'un ensemble par $|.|$.

À l'aide de cette représentation visuelle des données manipulées, il est possible d'expliciter un certain nombre de critères permettant de juger de la qualité de la requête ^tQ obtenue à partir de Q.

Représentativité des données initiales Bien entendu, le processus proposé ne doit pas aboutir à une requête ^tQ dont l'évaluation ne serait pas du tout représentative des résultats de Q. De part le processus d'apprentissage supervisé mis en place, utilisant les exemples et contre-exemples, on peut s'attendre à obtenir des motifs permettant de respecter ce critère. On peut mesurer concrètement ce critère avec les formules suivantes:

$$\frac{|^tQ \cap Q|}{|Q|} \underset{\text{optimal}}{=} 1 \tag{1}$$

$$\frac{|^tQ \cap \pi_X(\overline{Q})|}{|\pi_X(\overline{Q})|} \underset{\text{optimal}}{=} 0 \tag{2}$$

L'équation 1 justifie la représentativité directe des données obtenues par ^tQ vis-à-vis des données obtenues par Q : on devrait optimalement retrouver par ^tQ tous les tuples de Q.

De manière similaire, l'équation 2 permet de mesurer la proportion de tuples obtenus via $\overline{Q}$ retrouvés dans ^tQ, qui se doit d'être la plus faible possible.

Exemple 8 Les critères (1) et (2) sont optimaux pour la requête transmutée de l'exemple. ◊

Critère de diversité L'objectif de ce processus de réécriture est d'aboutir à une requête proche d'une requête initiale de l'utilisateur (mesurable par le critère précédent), mais de répondre également à une attente exploratoire de l'utilisateur et donc de présenter de nouveaux tuples. Il est donc important que cet ensemble de nouveaux tuples soit non-seulement non-vide (équation 3) mais aussi d'une taille pertinente, c'est-à-dire qui ne soit pas très petite par rapport aux données obtenues initialement par l'utilisateur (équation 4), ni comparable à la taille de l'ensemble total des tuples (équation 5), puisque dans ce dernier cas il est probable que ce résultat soit difficile à interpréter par l'utilisateur.

$$^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q}))) \neq \emptyset \tag{3}$$

$$|^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q})))| \ll |Q| \tag{4}$$

$$|^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q})))| \ll |\pi_X(Z)| \tag{5}$$

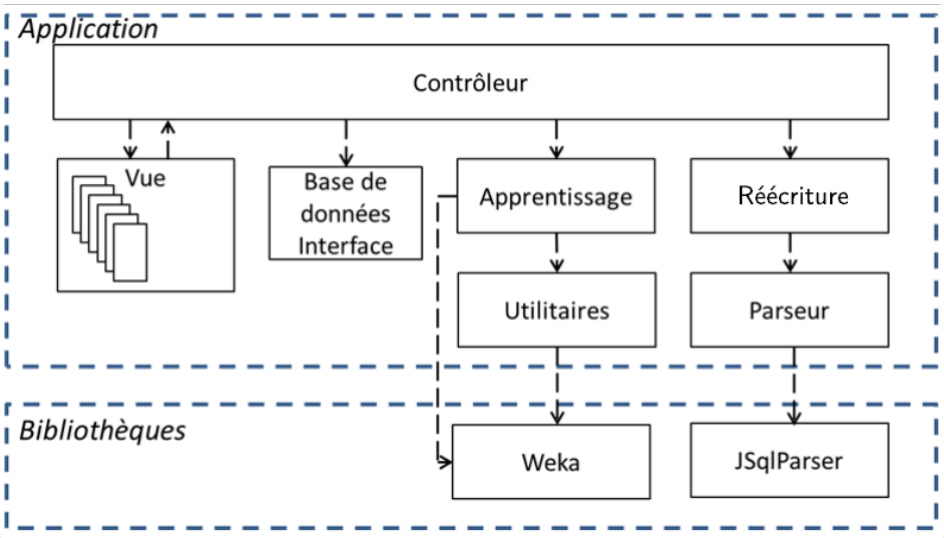


Figure 4: Organisation interne des modules

Exemple 9 Pour le critère (3), on a un nouveau tuple (SPEN) dans le résultat de la requête transmutée. Ce nouveau tuple est comparable au regard des deux tuples initiaux (critère (4)) et plus petit que les 10 tuples possibles (critère (5)). ◊

Ces différents critères peuvent facilement se calculer en SQL, voir section 4.

4 Implémentation de iSQL

Nous détaillons dans cette partie le prototype que nous avons développé. Il s'agit d'une application web implémentée en *Java* (version 1.7), *Scala* (version 2.10.3) et *Javascript* autour du Framework *Play*[3] (version 2.2.3). Les données étaient stockées sur un serveur *Oracle V11*.

Nous utilisons le classifieur *J48*, qui est une implémentation de l'algorithme C4.5 [11] disponible dans la bibliothèque *Open Source Weka* [9] (version 3.6.12). Afin de parser les requêtes SQL, nous utilisons la bibliothèque *Open Source JSqlParser* (version 0.9.1). Elle fournit en particulier une implémentation du *design pattern Visitor* permettant de dynamiquement visiter les différentes clauses d'une requête.

La Figure 4 synthétise l'organisation interne de iSQL ainsi que les bibliothèques spécifiques sur lesquelles elle s'appuie.

Le contrôleur contient l'ensemble des classes nécessaires à la coordination des opérations lancées par une requête de l'utilisateur via une des vues. A chaque requête est attachée une action spécifique dans le contrôleur qui déclenche un ensemble de tâches coordonnées. Le module Vue représente l'interface utilisateur, lisible par un simple navigateur web. *HTML* et *CSS* sont utilisés pour la partie statique et esthétique des vues, alors que le trio *JavaScript*, *JQuery* et *CoffeeScript* est utilisé pour la partie dynamique afin d'extraire de manière asynchrone les informations envoyées par le contrôleur.

Le module "Base de données - Interface" réalise quant à lui l'interconnexion avec les bases de données de l'utilisateur et permet notamment d'exécuter des requêtes sur celles-ci et d'en récupérer les résultats. Seul le SGBD *Oracle* est supporté actuellement.

Le module d'Apprentissage utilise la bibliothèque *Weka* pour réaliser l'apprentissage supervisé et en extraire un arbre de décision. Le *J48* est utilisé avec les paramètres par défaut fournis par la librairie *Weka* c'est-à-dire avec élagage de l'arbre et pas d'utilisation de l'estimation de Laplace dans sa construction.

Le module Utilitaire regroupe l'ensemble des objets nécessaires à l'extraction des conditions d'un arbre de décision. Les prédicats numériques et textuels sont évalués.

Le module Réécriture est le moteur de l'application ; c'est là que sont réalisées en majeure partie les actions qui permettent de réécrire la requête initiale.

Le module Parseur utilise la bibliothèque *JSqlParser*.

[3] `https://www.playframework.com/`

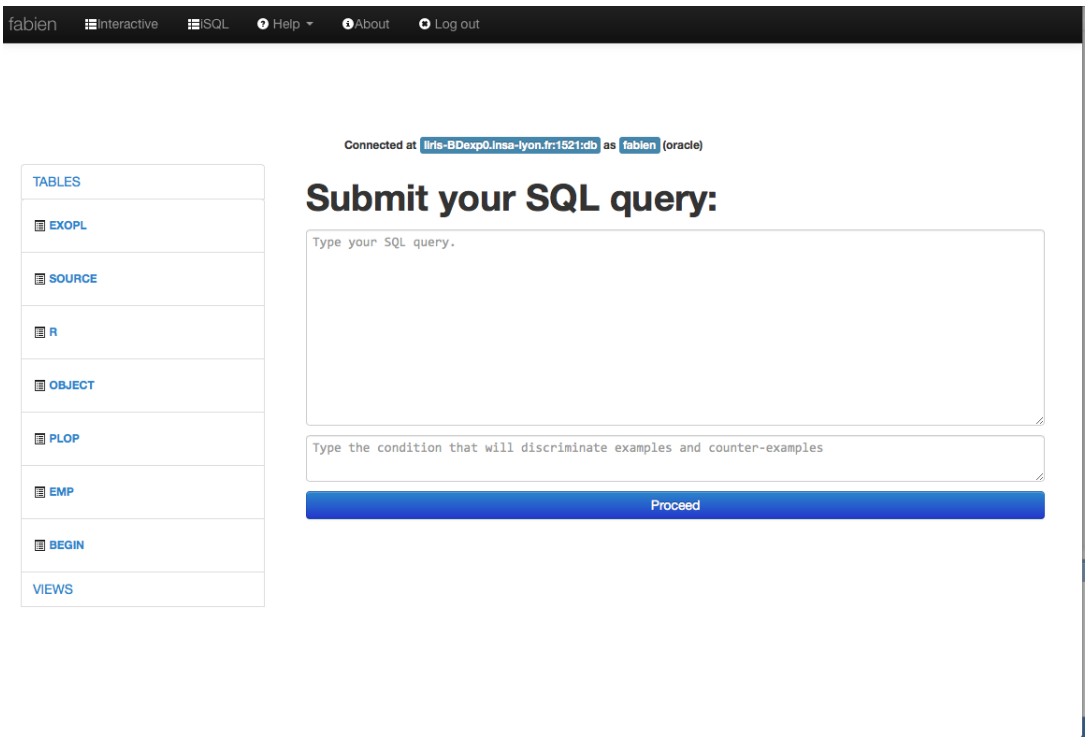


Figure 5: Interface de requêtage du prototype iSQL

Présentation de l'outil Après s'être connecté à une instance de bases de données, l'interface Web de iSQL est donnée en Figure 5.

La condition discriminante F_1 doit être bien différentiée de F_2 aux yeux de l'utilisateur, avec une interface de saisie différenciée.

Pour des questions de temps et d'accès, les exemples et les contre-exemples sont extraits en une seule requête afin d'éviter l'ouverture de deux flux sur le SGBD. Ces précisions établies, la requête exécutée par le système a la forme suivante :

```
SELECT t.*, (CASE
    WHEN F1 THEN 'YES'
    ELSE 'NO' END) AS CLASS
FROM R1 ...  RN t
WHERE F2
AND J;
```

La clause `CASE WHEN` permet de différencier les exemples qui vérifient F_1 des contre-exemples qui ne la vérifient pas. Ceci est enregistré dans un nouvel attribut, noté `CLASS` dans cet exemple, en référence à la notion de classe utilisée pour l'apprentissage.

Il est facile d'effectuer un échantillonnage ici et de ne pas prendre en compte l'ensemble des tuples.

Une phase de pré-traitement avant l'apprentissage a été implémentée pour supprimer les attributs ayant trop de valeurs manquantes.

A cette étape, l'analyste doit sélectionner les attributs qu'il souhaite utiliser pour la phase d'apprentissage. Ne lui sont proposés que les attributs qui n'apparaissent pas dans la condition discriminante. Nous prévoyons de trier ces attributs suivant différents critères pour aider l'analyste dans ce choix difficile si le nombre d'attributs est grand.

En parcourant en profondeur l'arbre de décision appris, nous générons la requête réécrite comme présentée en Section 3.2. L'évaluation de cette requête est alors présentée sous forme d'anneaux, voir Figure 8. Ces anneaux représentent les cinq critères présentés dans la section 3.3 : le premier anneau permet de visualiser les proportions globales de tuples exemples, contre-exemples et nouveaux tuples obtenus par l'évaluation de ${}^tQ$. Les deuxième et troisième anneaux représentent directement les critères de représentativité donnés respectivement par les formules 1 et 2, permettant de se faire une idée visuelle de la qualité de la réécriture vis-à-vis de la requête initiale. Le dernier anneau permet à l'utilisateur de

visualiser les critères de diversité donnés par les formules 3, 4 et 5, permettant de juger de la qualité exploratoire de la réécriture.

Il faut noter que l'utilisateur de iSQL n'a à aucun moment eu besoin de changer son environnement de travail. Il continue à manipuler les données en SQL de façon complètement transparente. Il est naturellement possible de faire ce processus en utilisant des outils d'apprentissage ou de fouille de données, mais de façon beaucoup plus douloureuse pour l'analyste, avec le changement d'outils et de systèmes.

5 Application sur une base de données scientifiques

Notre application porte sur des données issues du projet européen `CoRoT` qui a observé pendant plusieurs années les étoiles de notre galaxie avec pour but, l'étude des étoiles et la découverte de planètes extra-solaires. Un échantillon de la base `EXODAT`[4] a été extrait afin de créer une base de tests homogène et ciblé sur l'observation d'un champs `CoRoT`. L'échantillon étudié dans cette application est constitué de 97717 tuples de 62 attributs. Les attributs représentent, entre autre, la position de l'étoile, ses magnitudes à différentes longueur d'onde, le degré de son activité. Un attribut particulier permet de qualifier la présence ou non de planètes autour de l'étoile. Cet attribut sera nommé `Object` et peut prendre trois valeurs distinctes :

- p : il y a présence de planètes autour de l'étoile considérée.
- E : il n'y a pas de planètes autour de l'étoile considérée.
- null : la présence ou non de planètes autour de l'étoile n'a pas été étudiée.

L'objectif de cette expérimentation est d'obtenir une liste d'étoiles qui peuvent potentiellement abriter des planètes. La requête initiale est donc très simple : il suffit de sélectionner les tuples dont la valeur pour l'attribut `object` est 'p'. Il aura ainsi les étoiles pour lesquelles la présence de planètes est déjà confirmée.

Initialisation Dans cette application nous souhaitons donc identifier certaines conditions qui permettent de discriminer la présence de planètes pour les étoiles dont il manque des informations, à partir des étoiles dont on sait dire s'il y a ou non des planètes autour d'elles. La requête d'un utilisateur ayant cet objectif pourrait donc initialement être :

```
SELECT DEC, FLAG, MAG_V, MAG_B, MAG_U
FROM EXOPL
WHERE OBJECT IS NOT NULL
AND OBJECT = 'p'
```

La condition de sélection `OBJECT = 'p'` représente ici notre condition de discrimination F_1 et `OBJECT IS NOT NULL` la condition F_2.

Construction de E_+ et E_- A partir de cette requête initiale, le prototype génère la requête permettant d'extraire les exemples et les contre-exemples (voir Figure 6).

Il y a donc 50 exemples et 175 contre-exemples sur les 97717 tuples que compte la base de données. La plupart des étoiles n'ont donc pas été classifiées et ont une valeur nulle sur l'attribut `Object`.

Apprentissage L'utilisateur doit à ce moment sélectionner les attributs qu'il souhaite voir pris en compte pour l'apprentissage de règles sur les exemples et contre-exemples. Dans notre expérimentation, un échange rapide avec des physiciens travaillant sur ces mêmes données a fait ressortir que les différents attributs de *magnitudes* et d'*amplitudes* sont des données pertinentes à étudier, relatives à la lumière observée sous différents filtres de longueurs d'ondes. À partir de cette information experte mais facilement exploitable, nous avons en l'espace de quelques minutes essayé quelques ensembles d'attributs sur lesquels faire l'apprentissage. Dans le cas présent, l'expert a sélectionné les attributs MAG_B, AMP11, AMP12, AMP12, AMP13 et AMP14.

L'apprentissage est alors lancé et fait ressortir l'arbre de décision généré dont la règle suivante peut être extraite :

- `MAG_B > 13.425 AND AMP11 <= 0.001717`

grâce à laquelle le système propose la nouvelle requête (voir Figure 7).

[4] `http://cesam.lam.fr/exodat`

```
SELECT t.*, (CASE WHEN
        t.OBJECT = 'p'THEN 'YES'
       ELSE 'NO' END) AS CLASSTYPE
FROM (EXOPL) t
WHERE  t.OBJECT IS NOT NULL
```

Proportions

Number of examples: 50 (22.2 %)
Number of counter-examples: 175 (77.8 %)

Figure 6: Construction des échantillons d'exemples et de contre-exemples

```
J48 pruned tree
------------------

MAG_B <= 13.425: NO (194.0/36.0)
MAG_B > 13.425
|   AMP11 <= 0.001717: YES (11.0)
|   AMP11 > 0.001717: NO (20.0/3.0)

Number of Leaves  :     3

Size of the tree :      5
```

```
SELECT DEC,FLAG,MAG_V,MAG_B,MAG_U
FROM EXOPL
WHERE (MAG_B > 13.425 AND AMP11 <= 0.001717)
```

Figure 7: Arbre de décision appris et requête transmutée générée après avoir sélectionné les attributs MAG_B, AMP11, AMP12, AMP13 et AMP14

Évaluation des proportions Afin d'évaluer la pertinence de la requête trouvée par rapport à la requête initiale, nous avons présenté la Figure 8 aux experts.

Les différentes proportions obtenues en Figure 8 montrent que cette requête semble bien discriminer un sous-ensemble des exemples par rapport aux contre-exemples, tout en faisant ressortir 1% de nouveaux tuples de la table. Clairement, ces exemples étaient tout simplement hors de portée de l'analyste, la condition `AMP11 <= 0.001717 AND MAG_B > 13.425` avait peu de chance de germer à cette étape là de l'exploration des données. En fait le système trouvé montre les limites de détectabilité. En effet, pour des magnitudes supérieurs à 13,425, c'est à dire pour des étoiles plus faibles, il est indispensable d'avoir des amplitudes de variabilité de l'étoile inférieurs à 0.001717 (c'est à dire que la lumière émise par l'étoile doit avoir une variabilité faible). Cette double condition montre que l'on a une limitation en detection. Ces tuples, représentant des étoiles autour desquelles la présence ou non d'une planète n'a pas été étudiée, peuvent donc être des cibles prioritaires d'étude de part leur proximité, dans l'espace d'exploration des données, à un sous-ensemble des étoiles autour desquelles la présence d'une planète a été confirmée.

Intérêt pour l'Astrophysique Cette approche est importante pour plusieurs points. D'abord parce que l'échantillon d'étoiles comportant des planètes est très petit par rapport au nombre d'étoiles existant dans notre propre galaxie. Il devient alors difficile d'employer des méthodes classiques d'apprentissage pour déterminer le type de population. L'approche proposé dans ce papier permet de guider les recherches dans des domaines qui n'auraient pas été évident de cerner. Cette approche permet itérativement de définir des paramètres dans les formules de sélection qui ne semblaient que peu pertinents de prime abord.

D'autre part l'Astrophysique, comme toute science d'observation, comporte beaucoup de données qui, si elle ne sont pas erronées, sont entachées d'une incertitude. L'approche décrite permet de déterminer les échantillons les plus intéressants pour répondre à un problème spécifique. Enfin les bases de données astrophysiques possèdent beaucoup de valeurs NULL et une couverture assez disparate des attributs d'un projet d'observation à un autre. Cette approche permet de limiter l'importance de ces valeurs nulles en optimisant les valeurs connues.

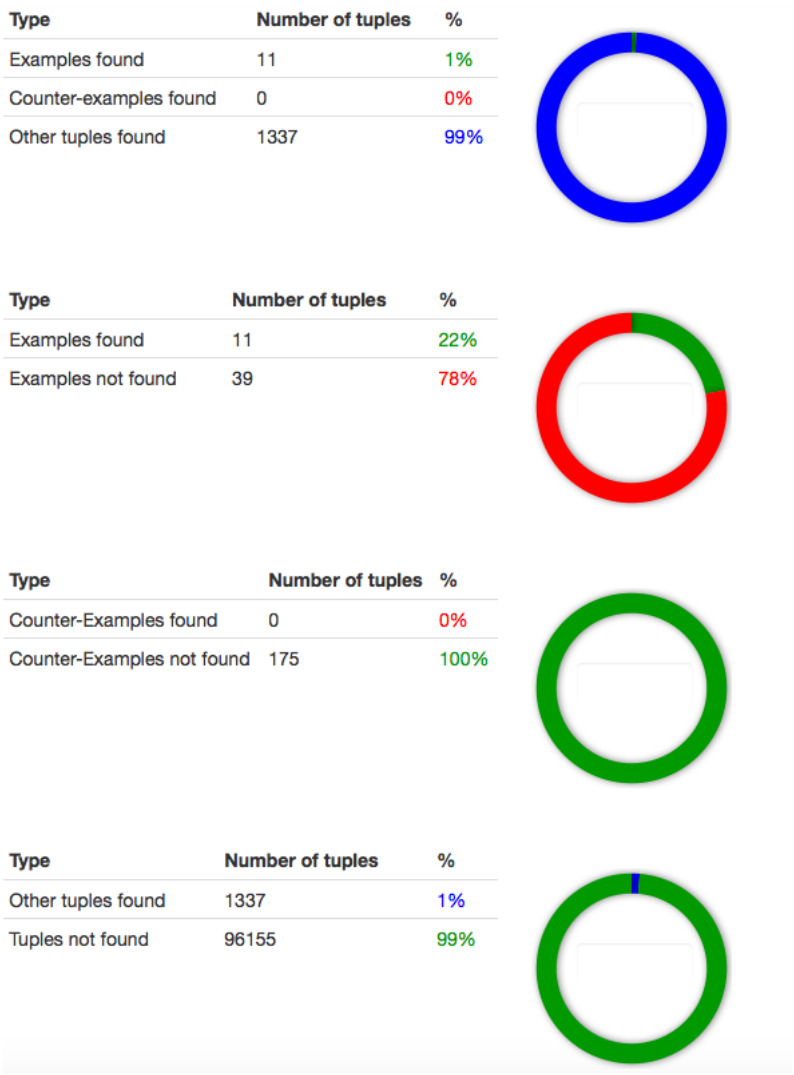
Classification des nouveaux tuples par d'autres classifieurs Sur cette validation, il est frappant de voir à quel point le processus que nous avons proposé ressemble à ce qu'il est possible de faire avec des outils de fouille de données ou d'apprentissage automatique. Son intérêt principal est de rester dans le cadre de SQL, de façon transparente pour l'analyste. Cela dit, afin de s'assurer de la capacité de l'algorithme d'arbre de décision utilisé à aboutir à des règles pertinentes sur les tuples, nous avons étudié le comportement d'autres méthodes de classification disponibles dans la librairie Weka sur les nouveaux tuples obtenus. Chacun des classifieurs[5] a été entrainé sur les mêmes exemples et contre-exemples issus de la requête initiale de l'utilisateur. Nous leur faisons ensuite classifier les 1337 nouveaux tuples que nous avons obtenus :

- Perceptron multicouche (*MLP*) avec 2 couches cachées et 100 neurones par couche : 1315 sont classés comme étant des étoiles hébergeant des planètes et 22 comme des étoiles sans planètes.
- Classifieur à règles floues : 626 sont classés comme hébergeant des planètes, 127 comme sans planètes, et 584 tuples sont inclassables.
- Machine à vecteurs de support (*SVM*) à noyau (*RBF*) : 725 sont classés comme hébergeant des planètes et 615 comme sans planètes.

De ces résultats, nous pouvons dire que la pertinence des tuples extraits à partir des conditions issues de l'arbre de décision produit par l'algorithme *J48* est en partie vérifiée par d'autres classifieurs, voire même en grande partie retrouvée par le *MLP* ce qui est rassurant quant à l'utilisation d'arbres de décision dans le processus présenté. En revanche, la génération de condition de sélection relationnelle à partir des modèles appris par ces autres classifieurs est souvent impossible à construire.

Rapidité et simplicité d'utilisation L'un des objectifs majeur visé par ce système est de proposer un système simple, interactif et rapide d'utilisation pour obtenir des tuples qui peuvent intéresser l'utilisateur. L'expérimentation présentée justifie l'atteinte de cet objectif en évitant notamment à l'utilisateur de devoir jongler constamment entre son système de gestion de bases de données et son outil de fouille de données. Le prototype masque de plus toute la complexité experte de la fouille de données à laquelle il n'est pas

[5]Les paramétrages de ces classifieurs sont les paramétrages par défaut de Weka, en dehors des paramètres précisés.



Type	Number of tuples	%
Examples found	11	1%
Counter-examples found	0	0%
Other tuples found	1337	99%

Type	Number of tuples	%
Examples found	11	22%
Examples not found	39	78%

Type	Number of tuples	%
Counter-Examples found	0	0%
Counter-Examples not found	175	100%

Type	Number of tuples	%
Other tuples found	1337	1%
Tuples not found	96155	99%

Figure 8: Évaluation des proportions pour la requête transmutée de la figure 7

forcément familier, complexité qu'il se devrait de surmonter s'il utilisait des outils de fouille de données comme `KNIME` ou `Weka`.

6 État de l'art

L'exploration interactive des données est, en raison de l'émergence récente des problématiques liées au Big Data, un domaine de recherche en plein essor. Un certain nombre d'articles de la littérature présentent en particulier des approches d'exploration basées sur la réécriture de requête et/ou sur l'apprentissage automatique, dans l'esprit de ce qui est proposé dans cet article.

L'idée de *Query Morphing* a été proposée par [10] : les auteurs proposent à l'utilisateur des données supplémentaires ne répondant pas à leur requête mais restant potentiellement proches des tuples du résultat exact, et lui donnent accès à des données qui sont probablement difficiles à atteindre avec des conditions de sélection simples, et contrecarrent donc partiellement l'exactitude des résultats en SQL, qui peut être trop limitante dans un cadre exploratoire [7]. Cependant, au lieu de générer des requêtes par modifications faibles de la requête initiale de l'utilisateur comme le font [10], nous proposons ici d'exploiter des ensembles d'exemples et de contre-exemples de la requête initiale de l'utilisateur, pour obtenir une nouvelle requête via les règles apprises sur ces ensembles. Il nous semble en effet trop difficile de définir une méthode de modification *faible* d'une requête SQL qui soit suffisamment générale pour convenir à toutes les conditions de sélection, et surtout à toutes les données.

Dimitriadou *et al.* proposent dans l'article [8] un système d'exploration des données basé, comme

la méthode du présent article, sur la proposition de nouveaux tuples via l'apprentissage automatique sur des tuples exemples et contre-exemples. Bien que la méthode de génération de nouvelles règles de sélection est très proche de celle présentée ici (en utilisant les règles apprises par un arbre de décision), la génération des exemples d'apprentissage est fondamentalement différente de notre méthode. En effet, le système proposé par Dimitriadou *et al.* demande à l'utilisateur de labelliser manuellement des tuples pour constituer ces ensembles d'apprentissage, à chaque étape de réécriture.

Le système *JIM* proposé dans [5] utilise également la notion de tuples exemples et de tuples contre-exemples pour aider l'utilisateur à formuler des requêtes de jointure. Là aussi, ces ensembles d'exemples et de contre-exemples sont issus de la labellisation des tuples faite manuellement par l'utilisateur.

Bien que les auteurs de [8] présentent des méthodes pour réduire le nombre de tuples à labelliser, nous pensons qu'il est simplement préférable de ne jamais avoir à demander à l'utilisateur de devoir labelliser des tuples pour explorer ses données. En effet, cette tâche est d'une part rapidement usante pour l'utilisateur (et réduit donc l'interactivité, la simplicité et l'attractivité du système), et d'autre part est loin d'être triviale lorsque l'utilisateur n'est pas expert sur les données (ce qui est très souvent le cas lorsque l'on souhaite justement les "explorer") ou si les critères de labellisation sont intrinsèquement complexes (ce qui est également très souvent le cas, sinon l'utilisateur pourrait probablement obtenir les données avec une requête simple). Nous proposons dans cet article d'obtenir ces ensembles d'exemples et de contre-exemples automatiquement à partir de la requête initiale, sans demander l'intervention de l'utilisateur.

La proposition de [12] vise à découvrir les requêtes possibles à partir d'un ensemble de tuples identifiés par l'utilisateur, libérant totalement celui-ci du travail de formulation de requêtes. Le nombre de requêtes générées peut néanmoins être très grand et contrairement à ce que nous proposons, ils ne prennent en compte que les exemples positifs donnés par l'utilisateur.

Pour finir, notre approche s'apparente au raisonnement suivant, classique en intelligence artificielle (voir par exemple [2]) : On dispose d'une *hypothèse* initiale H, formulée par des experts. En présence de données, les exemples positifs correspondent aux données qui vérifient H et les exemples négatifs les données apportant des contre-exemples à H. Clairement, les hypothèses peuvent être beaucoup plus générales que les requêtes SQL, allant des dépendances fonctionnelles (DF) à des motifs beaucoup plus complexes. L'approche que nous proposons à donc un caractère générique et se généralise à de nombreux problèmes d'intelligence artificielle où une hypothèse est formulée puis raffinée au fur et à mesure des interactions. Cette notion d'exemple et de contre-exemple a aussi abondamment été étudiée dans le contexte des bases de données avec les relations d'Armstrong [3, 4] pour les DF.

7 Conclusion et discussion

Nous avons présenté une approche que nous pensons très prometteuse pour répondre à un des enjeux du Big Data : formuler de façon interactive une requête SQL qui correspond à ce que l'analyste recherche et qui soit efficace à exécuter sur des données gigantesques. À partir d'une requête initiale d'un utilisateur non-expert, nous avons montré qu'il était possible d'obtenir un ensemble d'exemples et de contre-exemples permettant d'alimenter un processus d'apprentissage à arbre de décision, dont le modèle appris permet une réécriture directe de la requête initiale en utilisant les règles obtenues. Cette nouvelle requête, de part le processus d'apprentissage, renvoie des résultats proches de la requête initiale tout en donnant de nouveaux tuples proches de la requête initiale. Cette diversité aurait été quasiment impossible à atteindre sans l'aide de l'apprentissage pour formuler la requête.

L'utilisateur peut également évaluer la qualité globale de la réécriture de sa requête, à l'aide des métriques en comparant par exemple le taux de nouvelles données atteintes, ou le nombre de tuples de la requête initiale retrouvés. Ainsi, il peut rapidement juger de la direction de son exploration, sans avoir à interpréter en premier lieu les données qu'il obtient à chacune de ses requêtes.

Une application Web a été développée et a permis de tester l'approche sur des données scientifiques en astrophysique. Les retours des astrophysiciens sont prometteurs tant l'approche correspond bien à leur façon de travailler : formulation d'une hypothèse puis raffinement successif en fonction des données à disposition. Le fait d'intégrer de façon transparente des techniques d'apprentissage avec SQL est aussi une rupture importante dans leur façon de travailler qui nécessite de séparer l'analyse entre des systèmes différents (SGBD puis système d'analyse de données), ce qui ne permet pas le tâtonnement nécessaire dans une phase d'exploration.

De nombreuses possibilités sont ouvertes par cette approche : nous pouvons facilement imaginer de l'étendre à d'autres types de requêtes SQL, notamment les requêtes skylines. Nous pouvons aussi étendre ce travail à la fouille de motifs à partir de langages déclaratifs comme `RQL`[6], ou plus généralement à tout

type de motifs. Pour une hypothèse (ou motif ou requête) donnée, la notion d'exemples et de contre-exemples semble relativement universelle. Ce type d'approche pourrait donc jouer un role intéressant dans l'exploration du Big Data dans les années à venir.

8 Remerciement

Ce travail a été financé par le projet `Petasky`[6] du programme `Mastodon` de la mission interdisciplinarité du CNRS.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[3] W. W. Armstrong. Dependency structures of data base relationships. In *IFIP Congress, Stockholm, Sweden*, pages 580–583, 1974.

[4] C. Beeri, M. Dowd, R. Fagin, and R. Statman. On the structure of Armstrong relations for functional dependencies. *J. ACM*, 31(1):30–46, 1984.

[5] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with JIM. *PVLDB*, 7(13):1541–1544, 2014.

[6] B. Chardin, E. Coquery, M. Pailloux, and J. Petit. RQL: A sql-like query language for discovering meaningful rules. In *ICDM (demo)*, pages 1203–1206, 2014.

[7] J. Cumin and F. Rouge. Exploration interactive des données dans les bases de données. *Synthèse Bibliographique INSA de Lyon*, 2015.

[8] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *SIGMOD*, 2014.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[10] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12):1474–1477, 2011.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 493–504, New York, NY, USA, 2014. ACM.

[6] http://com.isima.fr/Petasky