Requêtes discriminantes pour l'exploration des données

Julien Cumin*, Jean-Marc Petit*, Fabien Rouge*,
Vasile-Marian Scuturici*, Christian Surace**, Sabina Surdu*

*INSA Lyon, LIRIS (UMR 5205 CNRS) 69621 Villeurbanne
prenom.nom@insa-lyon.fr
**LAM, CNRS, Marseille
prenom.nom@lam.fr

Résumé. À l'ère du Big Data, les profils d'utilisateurs deviennent de plus en plus diversifiés et les données de plus en plus complexes, rendant souvent très difficile l'exploration des données. Dans cet article, nous proposons une technique de réécriture de requêtes pour aider les analystes à formuler leurs interrogations, pour explorer rapidement et intuitivement les données. Nous introduisons les requêtes discriminantes, une restriction syntaxique de SQL, avec une condition de sélection qui dissocie des exemples positifs et négatifs. Nous construisons un ensemble de données d'apprentissage dont les exemples positifs correspondent aux résultats souhaités par l'analyste, et les exemples négatifs à ceux qu'il ne veut pas. En utilisant des techniques d'apprentissage automatique, la requête initiale est reformulée en une nouvelle requête, qui amorce un processus itératif d'exploration des données. Nous avons implémenté cette idée dans un prototype (iSQL) et nous avons mené des expérimentations dans le domaine de l'astrophysique.

1 Introduction

L'un des aspects encore relativement peu étudié 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 communément le cas dans de nombreux domaines, de l'astrophysique à la biologie. Un analyste exploitant ce type de données va passer un temps important à construire la requête de sélection (via SQL) des données qui ont un intérêt pour lui. Nandi et Jagadish (2011) précise que le temps passé pour construire une requête SQL est beaucoup plus important que le temps d'exécution de cette même requête. En partant de ce constat, un nouveau paradigme d'interaction a vu le jour, utilisant les données existantes dans la base de données pour guider le processus de construction de la requête. L'analyste va raffiner successivement la requête en fonction des résultats obtenus pour arriver à une solution convenable.

Sur les données scientifiques plus précisément, il est commun d'avoir des relations avec plusieurs centaines d'attributs, la plupart avec des valeurs numériques issues de mesures physiques. Les critères de sélection des données ne sont pas toujours facilement exprimables avec les attributs existants dans la table. Par exemple, un astrophysicien peut vouloir sélectionner

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	20
	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

FIG. 1 – *Exemple illustratif (Chardin et al. (2014))*

des étoiles autour desquelles orbitent potentiellement des planètes, mais ses données ont des attributs liés principalement à la magnitude et l'amplitude de variations de luminosité. Définir des seuils adéquats sur ces attributs numériques dans une condition de sélection (SQL), sans passer par des outils de fouille de données, peut être une réelle difficulté pour l'analyste.

Afin d'illustrer intuitivement notre approche, nous présentons l'exemple donné en figure 1 avec une seule relation, *EMP*. 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. Les quelques valeurs manquantes sont dénotées par le symbole '-'. Imaginons que nous cherchions les femmes qui ont un bonus supérieur à leur responsable hiérarchique, correspondant à la requête SQL suivante :

```
SELECT Empno, Lastname, Bonus FROM EMP E1 WHERE Sex = 'F'
  AND Bonus > ANY(SELECT Bonus FROM EMP E2 WHERE E2.Empno=E1.Mgrno)
```

Les tuples correspondant à SPEN et KWAN vérifient cette condition. L'idée que nous allons développer dans ce papier est de dire que cette requête aurait pu être formulée comme suit :

```
SELECT Empno, Lastname, Bonus FROM EMP WHERE EducLevel > 17
```

Nous constatons que :

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

Problème abordé Notre souhaitons construire une interaction avec un Système de Gestion de Bases de Données (SGBD) permettant à un analyste d'explorer ses données via des requêtes. 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'interactivité 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 différent, 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 la conception de requêtes SQL et la fouille de données.

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 d'utiliser une méthode de fouille de données 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.

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 possible 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 différent, 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 destinées à l'étude de la structure interne des étoiles et à la recherche d'exoplanètes extra-solaires. Les résultats préliminaires que nous avons obtenus sont très encourageants et démontrent que cette idée pourrait s'avérer très utile pour des analystes de données non-informaticiens de formation.

2 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 Kersten et al. (2011) : les auteurs proposent à l'utilisateur des données supplémentaires via des modifications faibles de la requête initiale. Par rapport à cette approche, 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. D'une manière similaire, Dimitriadou et al. (2014) proposent un système d'exploration des données basé sur l'apprentissage automatique sur des tuples exemples et contre-exemples. Bien que la méthode de génération de nouvelles règles de sélection soit très proche de celle présentée ici (en utilisant les règles apprises par un arbre de décision), la génération des exemples d'apprentissage est fondamen-

talement différente de notre méthode. 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 Bonifati et al. (2014) 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. 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 Shen et al. (2014) 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. DataPlay Abouzied et al. (2012) aide l'utilisateur à formuler une requête quantifiée, exploratoire. L'utilisateur est tenu encore une fois de labelliser les ensembles de tuples exemples/contre-exemples, alors que dans notre approche aucune action de l'utilisateur au niveau des tuples n'est nécessaire.

Query-By-Example Zloof (1977) (QBE) est un langage graphique dans le monde relationnel, permettant la spécification des requêtes via une interface graphique, qui génère automatiquement le code SQL. Bien qu'il facilite la formulation d'une requête, il ne permet pas de construire automatiquement la requête à partir d'un ensemble de données.

Query From Examples Li et al. (2015) (QFE) permet aux utilisateurs d'écrire des requêtes en leur montrant successivement des couples résultats issus de la base de données, légèrement différents d'un ensemble initial, fourni par l'utilisateur. L'intervention de l'utilisateur est nécessaire à chaque itération, pour identifier le résultat correct. La cible de QFE d'après les auteurs sont les utilisateurs dont le niveau d'expertise SQL est faible. En outre, l'utilisateur doit fournir un premier résultat, alors que dans notre approche exploratoire une requête réécrite présentera à l'utilisateur des tuples supplémentaires, diversifiés par rapport au résultat initial.

3 Requêtes discriminantes et leur négation

Nous introduisons brièvement les notations utilisées dans la suite de ce papier, voir Abiteboul et al. (1995) pour les détails. On considère une base de données d définie sur un schéma 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. On considère l'algèbre relationnelle comme langage relationnel ou de façon équivalente sa représentation en SQL sur des exemples. On supposera que les requêtes sont de la forme suivante [1] : $Q = \pi_X(\sigma_F(r_1 \bowtie \ldots \bowtie r_n))$, utilisant les opérateurs de sélection (σ) d'un ensemble d'attributs X, projection (π) via une formule F et de jointure ($\bowtie$). 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 at-

1. En pratique, nous pourrions avoir des requêtes plus générales admettant par exemple des requêtes imbriquées, comme cela est le cas dans l'exemple de l'introduction. Les résultats que nous proposons restent valides, mais leur étude est cependant hors de portée de cet article.

tributs, v une valeur réelle et θ un opérateur binaire usuel ($<, \leq, >, \geq, =$). On note $attr(F)$ l'ensemble des attributs apparaissant dans une formule de sélection F.

La question qui nous intéresse est de définir l'ensemble des tuples qui *ne vérifient pas* une requête Q. Nous proposons de simplifier ce problème en introduisant une nouvelle condition syntaxique sur 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. Cette restriction syntaxique étant très simple, nous gardons une grande expressivité sur la classe de requêtes traitées.

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.

L'existence des valeurs manquantes dans un attribut A ne permet pas de dissocier des tuples en utilisant des conditions sur A, c'est la raison de l'exclusion de ces attributs de la condition de discrimination.

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. Si nous gardons tous les attributs possibles (sans projection sur X) : $E_+(Q) \subseteq \sigma_{F_1 \wedge F_2}(r_1 \bowtie \ldots \bowtie r_n)$. Sur l'exemple de l'introduction, la requête réécrite (pour la rendre compatible avec nos notations) serait :

```
SELECT E1.Empno, E1.Bonus FROM EMP E1, EMP E2
  WHERE (E1.Sex = 'F') AND (E1.Bonus > E2.Bonus
    AND E1.Mgrno=E2.Empno)
```

avec comme condition de discrimination `E1.Sex = 'F'`. Les exemples positifs sont les deux tuples correspondant à SPEN, KWAN.

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 :

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

Nous notons que : (A) F_1 est considérée comme étant la clause de discrimination des exemples et $\neg F_1$ celle des contre-exemples ; (B) 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 ; (C) 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 : $E_-(Q) \subseteq ans(\overline{Q}, d)$. 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.Mgrno=E2.Empno)
```

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	20	-

FIG. 2 – *Jeu d'apprentissage*

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

Nous pouvons maintenant construire un jeu de données d'apprentissage. L'idée n'est pas de précisément répondre à la requête, mais bien d'aider l'analyste à formuler d'une manière différente une requête qui corresponde mieux à ses attentes. Ces étapes de tâtonnement ne nécessitent pas de prendre en compte toutes les données si celles-ci sont trop volumineuses.

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

Nous ne gardons pas les attributs utilisés dans F_1 pour ne pas apprendre la condition de sélection déjà exprimée dans la requête initiale. En continuant l'exemple, on obtient le jeu décrit dans la Figure 2. L'attribut `Sex` a été supprimé et l'attribut `Class` a été ajouté avec les labels correspondants.

4 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.

Apprentissage supervisé de la condition de sélection Nous utilisons dans notre approche un arbre de décision (Quinlan (1993)) nous permettant de prédire la valeur de l'attribut $Class$ en fonction d'un ensemble d'attributs du jeu d'apprentissage. 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 être 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 pour prédire les valeurs de l'attribut *Class*. 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.

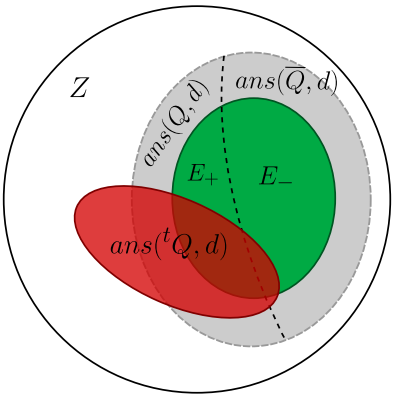

FIG. 3 – *Ensembles de tuples manipulés dans le processus de réécriture*

Sur l'exemple précédent, un arbre de décision va trouver la condition "`Educlevel` $>$ 17", équivalente à la requête :

```
SELECT Empno, Lastname, Bonus FROM EMP WHERE 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 une condition de sélection complètement nouvelle : elle peut porter sur des attributs non identifiés comme utiles dans la requête initiale.

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, voir la figure 3 (où l'ensemble des tuples de $r_1 \bowtie \ldots \bowtie r_n$ est dénoté par $\boldsymbol{Z}$).

Pour simplifier les notations, on notera $Q, \overline{Q}, {}^t Q$ à la place respectivement de $ans(Q, d)$, $ans(\overline{Q}, d)$ et $ans({}^tQ, d)$. On note aussi la cardinal d'un ensemble E par $|E|$. 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 Notre objectif est d'aboutir à une requête ${}^tQ$ dont l'évaluation soit 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 :

$$ \frac{|{}^tQ \cap Q|}{|Q|} \underset{\text{optimal}}{=} 1 \qquad (1) \qquad\qquad \frac{|{}^tQ \cap \pi_X(\overline{Q})|}{|\pi_X(\overline{Q})|} \underset{\text{optimal}}{=} 0 \qquad (2) $$

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. 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.

3. ${}^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q}))) \neq \emptyset$
4. $|{}^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q})))| \not\ll |Q|$
5. $|{}^tQ \cap (\pi_X(Z) - (Q \cup \pi_X(\overline{Q})))| \ll |\pi_X(Z)|$

Pour le critère (3), dans notre exemple nous avons un nouveau tuple (LAUREL) dans le résultat de la requête transmutée. Ces trois tuples sont en nombre comparable au regard des deux tuples initiaux (critère (4)) et sont moins nombreux que les 10 tuples possibles (critère (5)).

5 Implémentation

Nous avons implémenté notre proposition dans un prototype nommé *iSQL*. Il s'agit d'une application web implémentée en *Java*, *Scala* et *Javascript* autour du *framework Play*. Les données sont stockées sur un serveur *Oracle*. Nous utilisons le classifieur *J48*, qui est une implémentation de l'algorithme C4.5 Quinlan (1993) disponible dans la bibliothèque *Weka* Hall et al. (2009).

iSQL propose à l'utilisateur une interface Web lui permettant d'interagir avec une base de données, en spécifiant d'une manière différentiée une condition discriminante (F_1) et une condition de sélection (F_2). 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 précédemment. L'évaluation de cette requête est alors présentée sous forme d'anneaux, voir Figure 6. 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 externes, mais avec un effort beaucoup plus important pour l'analyste, nécessitant le changement d'outils et de systèmes.

6 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`[2] a été extrait afin de créer une base de tests homogène. Il est constitué de 97717 tuples ayant 62 attributs. Les attributs représentent, entre autre, la position de l'étoile, ses magnitudes à différentes longueur d'onde, le degré de son activité, etc. 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'. Nous obtenons 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 qui n'ont pas encore été étudiées, à partir des étoiles pour lesquelles la présence ou l'absence de planètes a été confirmée. 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 (Figure 4) génère la requête permettant d'extraire les exemples et les contre-exemples. 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 5).

2. `http://cesam.lam.fr/exodat`

FIG. 4 – *Interface de requêtage du prototype iSQL*

```
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)
```

FIG. 5 – *Arbre de décision appris et requête transmutée générée après avoir sélectionné les attributs MAG_B, AMP11, AMP12, AMP13 et AMP14*

Évaluation des résultats Afin d'évaluer la pertinence de la requête trouvée par rapport à la requête initiale, nous avons présenté nos résultats aux experts astrophysiciens du LAM. Les différentes proportions obtenues en Figure 6 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 pour les instruments actuelles. 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). 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. L'approche proposée permet de guider les recherches vers des étoiles qui n'auraient pas été faciles d'identifier à priori. 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. Elles 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 favorisant les valeurs connues.

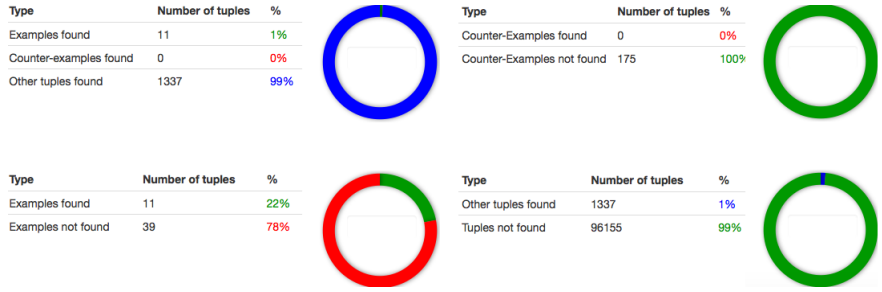

FIG. 6 – *Évaluation des proportions pour la requête transmutée de la figure 5*

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.

7 Conclusion et discussion

Nous avons présenté une approche 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.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 obtenues, ou le nombre de tuples de la requête initiale retrouvés. 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, sans séparer l'analyse entre des systèmes différents (SGBD puis système d'analyse de données) est aussi une rupture importante dans leur façon de travailler.

Comme perspective, il est possible de ne plus solliciter l'analyste pour choisir une condition discriminante et d'étudier les différentes classes connues de requêtes. On pourra aussi considérer d'autres types de motifs : pour une hypothèse (ou motif ou requête) donnée, la notion d'exemples et de contre-exemples semble relativement universelle.

Remerciement : Ce travail a été financé par le projet `Petasky` [3] du programme `Mastodon` de la mission interdisciplinarité du CNRS.

3. http ://com.isima.fr/Petasky

Références

Abiteboul, S., R. Hull, et V. Vianu (1995). *Foundations of Databases*. Addison Wesley.

Abouzied, A., J. M. Hellerstein, et A. Silberschatz (2012). Playful query specification with dataplay. *Proc. VLDB Endow. 5*(12), 1938–1941.

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

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

Dimitriadou, K., O. Papaemmanouil, et Y. Diao (2014). Explore-by-example : an automatic query steering framework for interactive data exploration. In *SIGMOD*.

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

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

Li, H., C. Chan, et D. Maier (2015). Query from examples : An iterative, data-driven approach to query construction. *PVLDB 8*(13), 2158–2169.

Nandi, A. et H. V. Jagadish (2011). Guided interaction : Rethinking the query-result paradigm. *PVLDB 4*(12), 1466–1469.

Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann.

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

Zloof, M. M. (1977). Query-by-example : A data base language. *IBM Syst. J. 16*(4), 324–343.

Summary

In the Big Data era, it is essential to explore data to unearth new knowledge. As user profiles become increasingly diverse and data ever more complex, it has become progressively hard to explore data. Analysts can access gigantic scientific data through SQL. In this paper, we propose a rewriting technique to help them formulate queries, to rapidly and intuitively explore big data. We introduce discriminatory queries, a syntactic restriction of SQL, with a selection condition dissociating positive and negative examples. We construct a learning dataset whose positive examples correspond to the results desired by analysts, and negative examples to those they do not want. We reformulate the initial query using machine learning techniques, and obtain a new query, more efficient and diverse. We propose measures to evaluate the rewriting quality. To support our approach, we developed the iSQL prototype on top of a commercial DBMS and conducted experiments with astrophysicists.