# How fuzzy set theory can help make database systems more cooperative

Aurélien Moreau

ANNÉE 2018

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

**THÈSE / UNIVERSITÉ DE RENNES 1**
*sous le sceau de l'Université Bretagne Loire*

pour le grade de

**DOCTEUR DE L'UNIVERSITÉ DE RENNES 1**

*Mention : Informatique*

**École doctorale MathSTIC**

présentée par


# Aurélien MOREAU


préparée à l'unité de recherche IRISA – UMR 6074
Institut de Recherche en Informatique et Systèmes Aléatoires
École Nationale Supérieure des Sciences Appliquées et de Technologie

---

# How Fuzzy Set Theory Can Help Make Database Systems More Cooperative

**Thèse soutenue à Lannion**
**le 26 juin 2018**

devant le jury composé de :

**Arnaud MARTIN**
Professeur, Université de Rennes 1 / *Président*

**Allel HADJALI**
Professeur, ISAE-ENSMA / *Rapporteur*

**Marie-Jeanne LESOT**
Maître de conférences HDR, Sorbonne Université / *Rapporteur*

**Dominique LAURENT**
Professeur, Université de Cergy-Pontoise / *Examinateur*

**Olivier PIVERT**
Professeur, ENSSAT / *Directeur de thèse*

**Grégory SMITS**
Maître de conférences HDR, Université de Rennes 1 / *Co-directeur de thèse*

*À mon père.*

# Résumé de la thèse en français

Les bases de données sont désormais omniprésentes, et nécessaires dans de nombreux cas d'utilisation : la gestion des employés d'une entreprise, le placement des cours d'une université, la gestion des stocks d'un entrepôt, la recherche de liens entre différents achats effectués, etc. Nées dans les années 1970 suite à la genèse du modèle relationnel par Codd, les bases de données relationnelles se sont progressivement imposées comme le standard permettant de stocker et de manipuler des informations. Le modèle relationnel est basé sur la logique booléenne, ce qui se traduit par plusieurs conséquences pour l'utilisateur : pas de classement des résultats (qui sont soit satisfaisants soit non satisfaisants), pas de prise en compte de la sensibilité aux bornes (une modification d'une condition de sélection pouvant transformer un ensemble de résultats vide en un ensemble de résultats pléthorique, et vice-versa). Plusieurs propositions ont été faites pour traiter ces problèmes, telles que les requêtes à préférences, des mécanismes coopératifs permettant de gérer la cardinalité des ensembles de réponses, ou divers paradigmes d'interrogation tels que la recherche par facettes, par mots-clés ou à partir d'exemples.

La théorie des ensembles flous permet de modéliser l'appartenance *partielle* d'un objet à une classe, par opposition à l'appartenance binaire modélisée par la logique booléenne. Elle permet ainsi d'apporter plus de flexibilité dans l'interaction entre les humains et les SGBD. L'utilisateur peut donc exprimer des préférences, et obtenir des réponses ordonnées en fonction de celles-ci. Des variables linguistiques peuvent être utilisées pour modéliser des termes de la langue naturelle tels que *jeune* ou *bien payé*. Cela se fait à l'aide de fonctions d'appartenance déterminant les seuils de valeur à partir desquels un élément aura totalement, partiellement, ou pas du tout la propriété définie. La formulation de requêtes à l'aide de variables linguistiques permet de renforcer l'expressivité des langages de requêtes, et donc de rendre les SGBD plus "intelligents".

Dans ces travaux de thèse nous proposons de tirer parti de la théorie des ensembles flous afin d'améliorer les interactions entre les systèmes de bases de données et les utilisateurs.

Les mécanismes coopératifs visent à aider les utilisateurs à mieux interagir avec les SGBD, notamment par le biais de réponses utiles et ne prêtant pas à confusion. Ces mécanismes doivent faire preuve de *robustesse* : ils doivent toujours pouvoir proposer des réponses à l'utilisateur. `Empty set (0,00 sec)` est un exemple typique de réponse qu'il serait désirable d'éradiquer, en indiquant par exemple à l'utilisateur quelles clauses de sa requête ont le plus contribué à la production d'une réponse vide. Plusieurs approches coopératives basées sur la théorie des ensembles flous ont déjà été proposées, permettant

notamment d'éviter les réponses vides à l'aide de techniques de relaxation par exemple.

Par ailleurs, les réponses à certaines requêtes peuvent parfois laisser les utilisateurs dubitatifs : qu'il s'agisse de réponses vides ou de réponses pléthoriques, il apparaît souhaitable de pouvoir *expliquer* ces réponses. L'origine des réponses (parfois appelée *provenance*) n'est que rarement précisée et beaucoup de systèmes sont vus comme des boîtes noires avec lesquelles les modalités d'interaction sont limitées.

Le caractère informatif des explications de réponses est parfois plus important que les réponses elles-mêmes : ce peut être le cas avec les réponses vides et pléthoriques par exemple, d'où l'intérêt de mécanismes coopératifs *robustes*, capables à la fois de contribuer à l'*explication* ainsi qu'à l'*amélioration* des résultats.

**Exemple 1** *Un utilisateur formule la requête* "restaurants à burgers bons marché à Lannion" *et obtient la réponse* `Empty set (0,00 sec)`*. Comme l'ensemble des résultats est vide, l'utilisateur pourrait souhaiter comprendre quelle(s) condition(s) de sa requête a (ont) contribué à la vacuité de cet ensemble de résultats (telle que* "bon marché" *par exemple), et obtenir des relaxations possibles pouvant mener à des résultats acceptables quoique pas entièrement satisfaisants vis-à-vis de la requête initiale (par exemple en relaxant l'interprétation de* "bon marché"*).*◇

L'exemple ci-dessus illustre une situation où il n'existe pas de réponse pleinement satisfaisante, et où une réponse proche pourrait tout à fait satisfaire un utilisateur. Cependant une explication reste nécessaire pour indiquer à l'utilisateur quelle condition de sa requête empêchait l'obtention de résultats. De plus, l'utilisation de termes de la langue naturelle pour décrire les données permet de garantir l'*interprétabilité* des explications fournies. La possibilité pour l'utilisateur de raffiner un vocabulaire contribue à la personnalisation des explications et améliore l'interprétabilité. Dans l'exemple ci-dessus la modalité "bon marché" est utilisée pour définir les valeurs de prix désirées par l'utilisateur : l'utilisation de modalités définies par les utilisateurs contribue à l'interprétabilité de la requête pour l'utilisateur.

Nous proposons de nous intéresser aux explications dans le contexte des réponses coopératives sous trois angles :

- dans le cas d'un ensemble pléthorique de résultats ;
- dans le contexte des systèmes de recommandation ;
- dans le cas d'une recherche à partir d'exemples.

Ces axes définissent des approches coopératives où l'intérêt des explications est de permettre à l'utilisateur de comprendre comment sont calculés les différents résultats proposés dans un effort de transparence. Le caractère informatif des explications apporte une valeur ajoutée aux résultats bruts, et en ce sens constitue une réponse coopérative.

## Explication de clusters de réponses

Des solutions pour la gestion des réponses pléthoriques ont déjà été proposées : à base de renforcement ou d'augmentation de la requête, ou de résumés des résultats

ou encore de diversification des résultats. Nous proposons ici une approche mêlant les résumés de données avec les explications pour permettre aux utilisateurs de comprendre la structure des réponses à leurs requêtes. Nous utilisons un vocabulaire flou composé de partitions décrivant chacun des attributs de la base. Chaque partition est composée de modalités dont les labels permettent la formulation d'explications avec des termes de la langue naturelle. L'objectif général de l'approche est de permettre à un utilisateur de comprendre l'ensemble des résultats de sa requête, s'il est possible d'y trouver des sous-ensembles intéressants, et le cas échéant quelles sont les propriétés qui caractérisent chacun de ces sous-ensembles et pas les autres.

**Exemple 2** *Considérons un utilisateur recherchant une voiture de seconde main de marque* Peugeot*, modèle 407 sur un site en ligne dédié. L'ensemble de réponses comprend plus de 1000 résultats, ce qui n'est pas évident à parcourir manuellement. Les attributs qui intéressent tout particulièrement l'utilisateur sont le* prix *et le* kilométrage*. Un partitionnement de ces résultats conduit à la formation de deux sous-ensembles distincts :*

- *un sous-ensemble de voitures* chères *avec un* faible *kilométrage ;*
- *un sous-ensemble de voitures* bon marché *avec un* fort *kilométrage.*

*De plus, des informations complémentaires (qui n'ont pas été demandées explicitement par l'utilisateur)* spécifiques à chaque sous-ensemble *sont disponibles : les voitures du premier sous-ensemble sont* récentes*, quand celles du second sont* plus anciennes.◇

Avec l'approche proposée nous visons trois objectifs :

- robustesse de l'approche (fournir des explications pour permettre aux utilisateurs de comprendre les propriétés partagées par des sous-ensembles de réponses) ;
- interprétabilité (les explications produites doivent être facilement interprétables par l'utilisateur, ce que nous nous proposons de faire à l'aide de partitions floues des domaines considérés, et de l'identification des explications les plus informatives) ;
- automatisation de la détection de sous-ensembles de réponses (étape qui ne doit pas requérir des utilisateurs des connaissances sur les algorithmes de clustering et leurs paramètres associés).

L'approche que nous proposons se décompose en trois étapes :

- détection : recherche de sous-ensembles de la réponse à l'aide d'un algorithme de clustering portant sur les attributs projetés de la requête ;
- description : recherche d'explications formées à partir des labels des attributs projetés de la requête ;
- caractérisation : recherche d'explications spécifiques et minimales formées à partir des labels des attributs qui ne sont pas dans la requête.

Une caractérisation candidate prend la forme d'une conjonction (de disjonctions) de couples (attribut, ensemble (flou) de labels). Les caractérisations sont des caractérisations candidates qui vérifient deux propriétés :

- spécificité : la caractérisation considérée est spécifique pour le sous-ensemble de réponses considéré et n'est pas spécifique pour les autres ;
- minimalité : il n'existe pas de sous-caractérisation ayant un degré de spécificité supérieur ou égal à la caractérisation concernée.

Une mesure permettant de déterminer le degré de spécificité d'une caractérisation a été proposée.

Deux variantes de cette approche sont proposées, d'abord une version booléenne puis une version floue. La différence majeure entre ces deux variantes porte sur la prise en compte (dans la version floue) du degré d'appartenance d'un élément à une (ou plusieurs) modalité(s) de la partition du domaine d'un attribut. En effet, la variante floue permet de modéliser l'appartenance graduelle d'un cluster à plusieurs modalités d'une partition, et de préciser l'importance relative de chaque modalité pour un cluster donné. En conséquence la variante floue permet de trouver des caractérisations dans nombre de cas où l'approche booléenne n'y parvient pas. Le caractère robuste de l'approche floue n'est pas garanti (dans la mesure où il n'existe pas nécessairement de caractérisations à trouver pour un sous-ensemble de réponses donné), cependant dans le cas où plusieurs caractérisations existent alors il est aisé de les déterminer puis de les trier à l'aide de leur degré de spécificité.

## Recommandations et explications basées sur les associations

Les systèmes de recommandation proposent aux utilisateurs des éléments pouvant les intéresser parmi la masse de contenus disponibles. Les techniques de recommandation peuvent se baser sur plusieurs critères tels que la similarité entre éléments, la similarité entre les notes données par les utilisateurs, le contexte associé aux éléments, des contraintes spécifiées par l'utilisateur, une combinaison de ces différents critères, etc.

Nous nous intéressons ici aux recommandations basées sur les associations typiques entre éléments, et à leurs explications permettant de les justifier. Nous définissons des critères de similarité basés sur les associations typiques entre éléments (entre un acteur et des réalisateurs par exemple), ou tirant parti de données démographiques (films typiquement appréciés par un public de jeunes étudiants par exemple). Plus exactement, les éléments sont associés au public qui les a appréciés, et nous considérons une similarité basée sur ce public. Les recommandations sont calculées sur la base des propriétés typiques (âge, catégorie d'emploi, genre, etc.) du public associé à chaque élément. Nous proposons deux méthodes pour tirer parti des données démographiques des utilisateurs :

- calculer pour chaque objet les multi-ensembles démographiques représentatifs des utilisateurs qui l'ont aimé, les comparer à l'aide d'une mesure de similarité et

établir pour chaque utilisateur une liste ordonnée de recommandations similaires aux objets qu'il a aimé ;

- calculer les ensembles typiques d'objets aimés par des utilisateurs avec les mêmes caractéristiques démographiques que l'utilisateur courant.

**Exemple 3** *À l'aide de notre approche de recommandation à partir de données démographiques, considérons un utilisateur qui aime les films* Terminator *et* Tron. *Considérons que ces films sont habituellement appréciés par de* jeunes étudiants. *Cette association fréquente entre films et données démographiques peut être mesurée à l'aide d'un degré de typicité :* jeune *et* étudiant *sont deux des propriétés typiques caractérisant le public de ces films. Le processus de recommandation peut tirer parti de ces liens de typicité pour proposer à l'utilisateur (qui n'est pas nécessairement lui-même un jeune étudiant) d'autres films appréciés par des* jeunes étudiants*, tel que* Oblivion.◇

Cette première approche nécessite de disposer d'un certain nombre de notes avant de pouvoir proposer des recommandations. La seconde approche quant à elle ne nécessite pas de notes de la part de l'utilisateur, seulement des informations démographiques (il est cependant nécessaire de disposer de notes d'*autres* utilisateurs). La seconde approche permet donc de traiter le problème du *cold start*, quand la première approche nécessite plus de données.

Dans tous les cas nous nous basons sur les critères de similarité utilisés pour proposer des explications accompagnant chaque recommandation. En d'autres termes, nous pouvons toujours proposer une explication pour chaque recommandation, ce qui contribue à la robustesse de cette approche. Dans le cas de l'exemple précédent, l'explication fournie serait basée sur les liens démographiques *jeune* et *étudiant*. Cela mènerait à une explication telle que : "Oblivion *vous est recommandé car vous avez apprécié les films* Terminator *et* Tron*, et ces trois films ont été appréciés par le même public typique, composé notamment de* jeunes étudiants."

Nous évaluons nos approches avec des données de films issues de la base MovieLens. Nous commençons par démontrer l'intérêt des explications fournies avec nos recommandations à l'aide d'une étude utilisateur, où près de 80% des réponses stipulent que les explications permettent de comprendre l'intérêt des recommandations. Nous avons également comparé nos approches à d'autres approches classiques basées sur le filtrage collaboratif notamment. La *F-mesure* de notre approche est meilleure que celle des autres et notre approche est également capable de correctement deviner des scores que les autres approches testées ne peuvent pas.

## Construction et explication de requêtes à partir d'exemples

La formulation d'une requête n'est pas toujours aisée pour les utilisateurs non-experts, pour lesquels un langage formel tel que SQL n'est pas intuitif. À côté, il existe également des utilisateurs qui ont parfois une idée de ce qu'ils souhaitent mais ne savent pas

comment l'exprimer. Nous proposons Fuzzy Query By Example, une approche permettant aux utilisateurs d'obtenir des résultats sans pour autant connaître ni le schéma de la base ni un langage de requête, en inférant leurs préférences.

**Exemple 4** *Considérons un utilisateur recherchant une voiture de seconde main, et qui n'est pas familier avec les systèmes d'interrogation de base de données. On lui propose d'évaluer différents objets représentatifs d'une base de données. S'il sélectionne plusieurs voitures partageant certaines propriétés (ayant un prix modique par exemple), alors celles-ci seront détectées comme pouvant représenter une préférence de l'utilisateur, puis traduites dans une requête pour obtenir des résultats similaires.*◇

Nous demandons à l'utilisateur d'évaluer (positivement ou négativement) des exemples préalablement choisis de la base de données. L'objectif est de proposer ensuite à l'utilisateur des éléments similaires aux exemples évalués positivement (exemples positifs), en écartant ceux qui sont similaires aux exemples évalués négativement (contre-exemples). Cela soulève deux questions :

- la sélection des exemples à soumettre à l'utilisateur pour évaluation ;
- l'interprétation de ces évaluations pour en déduire une requête retournant des résultats intéressants.

La sélection des exemples à soumettre s'opère par la recherche d'éléments représentatifs de la base de données vis-à-vis des données et vis-à-vis du vocabulaire considéré pour décrire les valeurs des attributs de la base. Plusieurs mesures de représentativité ont été proposées dans ce sens. L'interprétation des évaluations obtenues est traitée à l'aide de la notion de caractérisation vue plus tôt. Les caractérisations obtenues permettent de décrire avec des termes de la langue naturelle les préférences de l'utilisateur déduites des évaluations fournies. Ces mêmes caractérisations sont ensuite traduites en une requête floue permettant de récupérer des éléments satisfaisant ces préférences de l'utilisateur. Enfin, elles permettent d'expliquer les résultats ainsi obtenus. Une éventuelle interaction entre l'utilisateur et les caractérisations obtenues permet ensuite de raffiner les préférences inférées pour améliorer les résultats.

Pour un jeu de requêtes donné nous cherchons à déterminer si la sélection d'exemples de la base que nous opérons est meilleure qu'une sélection aléatoire d'exemples. Nos résultats expérimentaux montrent que c'est le cas et qu'en moyenne l'évaluation d'une quinzaine d'exemples suffit pour inférer correctement les préférences utilisateur cachées derrière chaque requête testée. Des requêtes trop précises menant à des ensembles de réponses (presque) vides ne permettent pas actuellement de retrouver les préférences utilisateurs. Cependant dans le cas de toutes les autres requêtes testées nous avons pu retrouver les conditions implicites correspondant au besoin de l'utilisateur. Les préférences utilisateur inférées sont présentées à l'utilisateur avec des termes de la langue naturelle, ce qui garantit le caractère interprétable de ces préférences. La précision et le rappel des exemples proposés sont également évalués en fonction du paramètre $\lambda$ qui définit le seuil à partir duquel une préférence utilisateur devient pertinente (parmi

l'ensemble des préférences potentielles). Dans l'approche proposée, l'évaluation des exemples est uniquement binaire : des propositions sont avancées pour permettre une évaluation plus fine des exemples par l'utilisateur.

## Synthèse

Au cours de cette thèse nous nous sommes intéressés à l'intérêt des explications pouvant accompagner un ensemble de résultats dans le cadre d'une interaction avec un SGBD. À côté de ces explications ont également été traitées les notions de robustesse et d'interprétabilité, en essayant de garantir un résultat d'une part, et en tirant parti d'un vocabulaire personnalisé d'autre part. Les approches proposées placent l'utilisateur au coeur du processus d'interaction avec le SGBD, en lui fournissant non seulement les résultats demandés, mais en plus, des informations complémentaires permettant de comprendre, donc de mieux appréhender certains de ces résultats.

Les approches proposées ont été implantées dans le cadre du modèle relationnel. Ces approches visent à être étendues au modèle RDF, en tirant parti de la sémantique incluse dans ce modèle. Dans la littérature, des travaux ont déjà été proposés dans ce sens (traitement d'une réponse vide, proposition d'un langage de recommandation, etc.), cependant les notions abordées dans cette thèse telles que la robustesse et l'interprétabilité ne semblent pas avoir été traitées dans ce domaine. De même, les explications qui pouvaient être générées à l'aide des associations clé primaire – clé étrangère dans le modèle relationnel doivent pouvoir être retrouvées voire même améliorées à l'aide du modèle RDF, en tirant parti du mécanisme d'inférence de connaissances.

# Remerciements

Je tiens à remercier Olivier Pivert et Grégory Smits, mes directeurs de thèse qui m'ont guidé et encouragé tout au long de ce chemin. Votre complémentarité et votre rigueur scientifique ont été remarquables dans cette aventure. Vous m'avez appris à clarifier, structurer, et présenter mes idées à coups de « c'est vraiment imbitable » et de « je crois qu'il faut un schéma ».

Je remercie Arnaud Martin qui me fait l'honneur de présider mon jury de thèse. Je remercie Marie-Jeanne Lesot et Allel Hadjali d'avoir accepté de rapporter mes travaux, et Dominique Laurent d'avoir accepté de participer à mon jury.

Je remercie Jean-Christophe Pettier et Vincent Barreaud pour m'avoir permis d'enseigner à l'Enssat tout au long de ma thèse, me permettant de découvrir et de confirmer mon goût pour l'enseignement. Je remercie également les collègues du pôle info : Damien, Daniel, François, Gwénolé, Hélène, Jean-Baptiste, Jonathan, Pierre, et Virginie, pour les échanges et les discussions de kfet.

Une place pour les collègues des autres pôles : Claire, Hervé, Nathalie, Nelly, et Patrice. Most particularly, thank you Sheila for making these past few years far more interesting than they would have otherwise been. Je remercie également les collègues qui font tourner la boutique : Caroline, Catherine, et Nasséra à la scol', Angélique, Joëlle, et Vincent pour la paperasse.

Au-delà de certains collègues qui sont devenus de bons amis, je remercie aussi ceux dont j'ai partagé la route bien avant de m'engager dans la voie de la recherche.

Je remercie mes parents pour avoir toujours eu confiance en moi, et m'avoir permis d'arriver là où j'en suis aujourd'hui.

*Last but not least,* cette grande aventure qu'est la thèse m'aura également permis de trouver, sur ce long chemin, l'âme sœur. Qu'il soit ici pleinement remercié.

# Publications During the Thesis

Moreau, A., Pivert, O., and Smits, G. (2015). A Clustering-Based Approach to the Explanation of Database Query Answers. In *Proc. of the 11th International Conference on Flexible Query Answering Systems (FQAS'15)*, Krakow, Poland

Moreau, A., Pivert, O., and Smits, G. (2016a). A Fuzzy Approach to the Characterization of Database Query Answers. In *Proc. of the 16th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'16)*, Eindhoven, Netherlands

Moreau, A., Pivert, O., and Smits, G. (2016b). Caractérisation floue de clusters de réponses. In *Actes des Rencontres Francophones sur la Logique Floue et ses Applications (LFA'16)*, La Rochelle, France

Moreau, A., Pivert, O., and Smits, G. (2017). A Typicality-Based Recommendation Approach Leveraging Demographic Data. In *Proc. of the 12th International Conference on Flexible Query Answering Systems (FQAS'17)*, pages 71–83, London, UK

Moreau, A., Pivert, O., and Smits, G. (2018). Fuzzy Query By Example. In *Proc. of the 33rd ACM Symposium on Applied Computing (SAC'18)*, Pau, France

# Contents

**Résumé de la thèse en français** **v**

**Remerciements** **xiii**

**Publications During the Thesis** **xv**

**Contents** **xvii**

**1 Introduction** **1**

**2 Background Notions** **9**
2.1 Relational Databases . . . . . 9
2.1.1 The Relational Model . . . . . 9
2.1.2 The SQL Language . . . . . 11
2.2 Fuzzy Set Theory . . . . . 13
2.2.1 Operations on Fuzzy Sets . . . . . 14
2.2.2 Fuzzy Quantifiers . . . . . 15
2.3 SQLf . . . . . 16

**3 Introduction to Cooperative Answering** **19**
3.1 Fundamentals of Cooperative Answering . . . . . 19
3.1.1 Beliefs and Expectations . . . . . 20
3.1.2 Presuppositions . . . . . 21
3.1.3 Misconceptions . . . . . 21
3.1.4 Intensional Answers . . . . . 22
3.1.5 Generalizations . . . . . 22
3.1.6 The Predominant Role of the Users . . . . . 23
3.2 Cooperative Answering in a Relational Database Context . . . . . 23
3.2.1 Browsing assistance . . . . . 24
3.2.2 Dealing with Unsatisfactory Results . . . . . 28
3.2.3 Additional Answers . . . . . 30
3.2.3.1 Association-based and Typicality-based Additional Results . . . . . 31
3.2.3.2 Recommendation . . . . . 32

3.2.4 Providing Explanations to Users . . . 37
3.2.4.1 Explaining Database Results . . . 37
3.2.4.2 Explaining Recommendations . . . 38
3.3 Chapter Conclusion . . . 39
3.3.1 Synthesis . . . 39
3.3.2 Objectives . . . 39

**4 Explaining Query Answers 41**
4.1 General Principle . . . 44
4.2 Detecting Clusters of Answers . . . 47
4.2.1 k-means vs. k-medoids . . . 47
4.2.2 LFCMed-select . . . 49
4.3 Describing Clusters of Answers . . . 50
4.3.1 Fuzzy Vocabulary . . . 50
4.3.2 Crisp Projection of Clusters on Vocabulary Partitions . . . 51
4.3.3 Fuzzy Projection of Clusters on Vocabulary Partitions . . . 52
4.4 Characterizing Clusters of Answers . . . 52
4.4.1 Crisp Characterization . . . 53
4.4.1.1 Crisp Properties . . . 53
4.4.1.2 Crisp Algorithms . . . 54
4.4.2 Fuzzy Characterization . . . 56
4.4.2.1 Fuzzy Properties . . . 56
4.4.2.2 Fuzzy Algorithms . . . 57
4.4.3 Improving the Characterization Format . . . 62
4.5 Experiments . . . 63
4.5.1 Comparing Characterizations . . . 63
4.5.1.1 Crisp Illustrative Examples . . . 63
4.5.1.2 Fuzzy Illustrative Examples . . . 64
4.5.1.3 Discussion . . . 66
4.5.2 Performances . . . 66
4.5.2.1 Crisp Algorithm Performances . . . 66
4.5.2.2 Fuzzy Algorithm Performances . . . 66
4.5.2.3 Discussion . . . 66
4.5.3 Specificity Threshold Values . . . 68
4.6 Discussion . . . 68
4.6.1 Bridges with Formal Concept Analysis and Rough Sets . . . 68
4.6.2 Bridges with Data Mining Techniques . . . 70
4.6.3 Altering the Detection of Clusters . . . 70
4.6.4 Altering the Characterization . . . 71
4.7 Summary . . . 71

**5 Association-Based Recommendations and Explanations 73**
5.1 Typicality in Fuzzy Set Theory . . . 76
5.1.1 Typicality Based on Frequency and Similarity . . . 77
5.1.2 Typicality Based on Strict Equality . . . 78
5.1.3 Comparing Fuzzy Sets of Typical Values . . . 79
5.2 Association-Based Approach . . . 81
5.2.1 Choice of Similarity Criteria . . . 84
5.2.2 Filtering the Sets of Potentially Similar Items . . . 85
5.3 Typicality-Based Approach Leveraging Demographic Data . . . 86
5.3.1 Using the User's Favorite Movies . . . 86
5.3.1.1 Step 1: Computing the Fuzzy Sets of Typical Features . 87
5.3.1.2 Step 2: Computing Multisets . . . 87
5.3.1.3 Step 3: Browsing the Similarity Matrix . . . 88
5.3.2 Using the User's Demographic Data . . . 89
5.3.2.1 Computing Items Typically Liked by People Based on One Characteristic . . . 90
5.3.2.2 Aggregating Typical Sets of Items . . . 90
5.3.3 Using the User's Favorite Movies and Demographic Data . . . 91
5.4 Explaining Recommendations . . . 92
5.4.1 Association-Based Approach . . . 92
5.4.1.1 Use of Foreign Keys . . . 93
5.4.1.2 Use of Atypical Properties . . . 95
5.4.2 Typicality-Based Approach Leveraging Demographic Data . . . 99
5.5 Experiments . . . 100
5.5.1 Computing Actors with Associations . . . 100
5.5.2 Computing Movies Based on the Audience . . . 100
5.6 Summary . . . 104

**6 Building and Explaining Queries with Examples 107**
6.1 Fuzzy Vocabulary . . . 109
6.2 Selecting Examples . . . 110
6.3 Inferring User Preferences from Evaluations . . . 113
6.4 Translating Preferences into Queries . . . 116
6.4.1 Translating Preferences . . . 116
6.4.2 Weighted Disjunction . . . 116
6.5 Explaining the Query and its Results . . . 117
6.5.1 Explaining Inferred Preferences . . . 117
6.5.2 Interacting with the User to Refine Results . . . 118
6.6 Experiments . . . 119
6.6.1 Comparison of Example Selection Methods . . . 119
6.6.2 Impact of the Specificity Threshold $\lambda$ on Precision and Recall . . . 120
6.7 Discussion . . . 122
6.7.1 User-Example-based Approaches . . . 123
6.7.2 Prototypical-Example-based Approaches . . . 123

6.7.3 On Finer Evaluations by the User . . . . . 125
6.8 Summary . . . . . 126

**7 Conclusion** **127**

**Conclusion** **127**

**Bibliography** **145**

**List of Figures** **147**

# Chapter 1

# Introduction

Databases are used everyday for all sorts of purposes: managing companies, mapping out courses in universities, managing stocks in warehouses, discovering links between items from customer purchases, etc.

## Early Stages of Relational Databases

Yet no longer than half a century ago, paper records reigned supreme to store information. While Codd sketched relational databases as early as in 1969 [Codd, 1970], the first commercial implementations were released in 1979 (Oracle v2), thus slowly beginning their deployment in companies and administrations. However, efficiently managing and exploring large amounts of data were not then possible: the architectures and hardware then available were not as sophisticated as those we use today, and the query-answering possibilities were strictly limited to obtaining direct answers to a query over small datasets. Fast-forward a few decades, and relational databases have imposed themselves as the standard to store information.

Obvious obstacles to picking up relational databases include the knowledge of a formal query language, which is not easy to grasp by end-users. Less obvious obstacles are linked to the founding principles of relational databases: the relational algebra enabling the operations (selection, projection, join) to manipulate databases is based on Boolean logic, and is hence restricted by its rigidity. Examples illustrating this rigidity include:

- Absence of any ranking between the answers (a tuple is either satisfactory or not satisfactory), making large sets of answers difficult to process. Yet users generally have some preferences, but Boolean logic does not easily enable the formulation of these subjective and naturally imprecise conditions (works leveraging skyline approaches such as [Borzsony et al., 2001] do use preferences but they do not rank results either);

- Sensitivity to the borders (lack of robustness), enabling small changes to a query to possibly drastically change the answer set (from an empty answer set to a plethoric

answer set and *vice-versa*). Users interested in finding a burgers restaurant close-by may query for those within 1km around the town center, only to find that there are none. By refining their query to search for those within 2km around the town center, they may this time find dozens of them, this time obtaining too many answers. This repetitive querying process forces users to reformulate their queries until the obtention a satisfying answer set. Considering answers that are not strictly in the ranges of the selection conditions but close to these enables: (i) to return more answers, and (ii) to rank these answers with a satisfaction degree.

These issues have all been tackled by researchers from the database community, who responded with new querying paradigms and operators to better manipulate databases. These include:

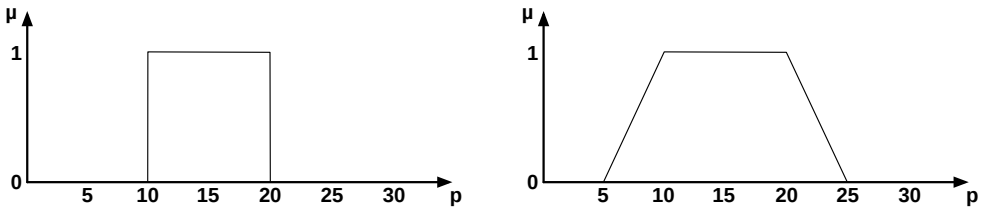
- Preference queries, adding more expressivity to query languages; being able to rank results according to additional user criteria;
- Cooperative mechanisms, helping users to deal with empty answer sets (relaxation mechanisms) and plethoric answer sets (summarization and query strengthening mechanisms), and to understand returned answers;
- User-friendly query paradigms: keyword search, faceted search, similarity search, etc.

## Fuzzy Set Theory

While classical database systems only use Boolean logic to express conditions, the human mind does not naturally see the world merely in black and white. Natural language can describe many vague notions (which can be uncertain as well) that can also be represented with gradual concepts: this led to the need for a formalism representing imprecise concepts.

Independently of databases — which barely existed then — Zadeh introduced fuzzy set theory in 1965 to model the classes of objects from the real world as perceived by the human mind [Zadeh, 1965]. In classical set theory, objects either belong or do not belong to these sets. In fuzzy set theory, objects can *somewhat* belong to a fuzzy set: membership degrees are used to express the extent to which objects belong to fuzzy sets. Classical set operations were extended to fuzzy sets, and with time additional fuzzy notions were introduced. Fuzzy logic has been applied to fields as diverse as robotics, operational decision, artificial intelligence, control theory, and as we will see, databases.

The general objective of fuzzy logic was to model some parts of human reasoning, as well as to capture some human interactions. These include studying the fuzzy frontier between full membership and full mismatch, and using linguistic variables to model terms from natural language. Whereas the vagueness and richness of terms from natural language cannot be captured by Boolean logic, fuzzy logic proved capable of enabling more expressivity to better model reality. In other words, formulating flexible queries




(a) Crisp membership (b) Fuzzy membership

Figure 1.1 – Sensitivity around the borders, explained.

to model human expressivity can be done using fuzzy set theory. This is made possible by using linguistic variables associated with terms of natural language, making up fuzzy vocabularies. Modalities from fuzzy vocabularies model terms that may convey different values and meanings depending on the individuals using them and on the context in which they are used. For instance let us consider the membership functions in Figure 1.1. With a membership function associated with a crisp predicate (Figure 1.1a) the results returned are those strictly between the values 10 and 20. With a membership function associated with a fuzzy predicate (Figure 1.1b) the results returned are those between the values 5 and 25, with those between 10 and 20 being fully satisfactory (which could also represent the condition "around 15"). By returning (more) ranked results, fuzzy membership functions contribute to the robustness of the querying process.

**In this thesis, we are interested in how we can leverage fuzzy logic to improve the interactions between relational database systems and humans.** Fuzzy logic enables modeling flexibility in the interactions with relational database management systems. The study of the relationship between these two domains has led to many scientific contributions, starting with the proposal of a conceptual framework for fuzzy query processing [Tahani, 1977]. Some of the limits of SEQUEL (and other then-used query languages) were already highlighted and Tahani proposed the use of fuzzy predicates to formulate queries that better match reality, with an extension of SEQUEL enabling the fuzzy selection.

The bridge between fuzzy sets and systems was consolidated by Dubois and Prade in their book [Dubois and Prade, 1980] detailing the theoretical notions brought forward with fuzzy set theory and discussing the possible applications. A great many publications followed to broaden the theoretical scope covered by fuzzy set theory and then possibility theory.

Kacprzyk and Zadrozny contributed to expanding functions of the Microsoft ACCESS database software with the FQUERY for ACCESS [Zadrozny and Kacprzyk, 1996] extension. This enables users to formulate queries that leverage some advantages granted by fuzzy logic, including obtaining ranked results and using quantified propositions in selection conditions, *e.g. "find the elements which satisfy* most of *the conditions among* $\{p_1, ..., p_n\}$*."* Then they improved this extension to provide linguistic summaries of databases [Kacprzyk and Zadrozny, 1999].

The use of fuzzy logic for flexible database querying was also put forward by Bosc and Pivert. They proposed to fuzzify the SQL language starting on a fuzzy extension of relational algebra, and presented SQLf [Bosc and Pivert, 1995], a relational database language for fuzzy querying. SQLf provides the possibility to express fuzzy selection conditions, fuzzy joins, fuzzy nesting operators, fuzzy quantified conditions, a quantitative or qualitative thresholding of the result, etc. More details on SQLf are provided in Chapter 2.

These past efforts aimed at making query languages closer to natural language with the use of fuzzy predicates in database queries. They contributed to improving the expressivity in database querying. Cooperative answering, covered below, focuses on improving the informative aspect of database query answers as well as the interaction between users and database systems.

## Cooperative Answering: at the Crossroads of Databases and Fuzzy Set Theory

Cooperative answering techniques aim to help users harness the potential of query-answering systems and databases by enabling these systems to provide helpful and insightful answers. Fuzzy logic aims to provide more *flexibility* and is one of the keys to implementing cooperative mechanisms to enable the user to better interact with systems. Better interactions include helping users to find acceptable answers when there are no "ideal" ones, or filter them down when there are too many. Cooperative answers also include providing hints and explanations as to why that is so in these cases, as well as justifying why some results may be missing when the user expected them.

Cooperative answering techniques are expected to work in most if not all situations where users are brought to interact with query-answering systems. This *robustness* property guarantees the flexibility that users may desire when formulating queries. According to the ISO/IEC/IEEE International Standard [IEC, 2017], robustness is defined as the *degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.* When it comes to database systems this definition can be reformulated as the *degree to which a database can return an adequate answer to any query.* This includes providing *useful* and *non-misleading* answers to all queries, including those initially yielding empty and plethoric answer sets, for instance by resorting to explanations.

The answer sets yielded must be useful (or at least informative) to the users: no empty or plethoric answer sets. Furthermore, the user may want to understand some part or all of the answer set: the user may be curious as to why some results appear in the answer set, or dubious as to why some expected results do not appear in the answer set. Cooperative answers must be informative so as to provide additional explanations to users requesting such information. The different cooperative answering behaviors and state-of-the-art approaches are covered in Chapter 3.

We also focus on the benefits that fuzzy set theory can bring to cooperative answering techniques. Several cooperative mechanisms are based on fuzzy queries (such as

query relaxation or strengthening). We propose to leverage the interpretability of linguistic variables to provide users with explanations related to their database browsing experience.

## A Crucial Challenge in Cooperative Answering: Explaining

Whenever users are provided with answers, their origin — sometimes termed *provenance* — is scarcely justified and systems seldom elaborate on such additional explanations. Many information systems and cooperative approaches are seen as black boxes: interaction with them is limited to some forms, and users do not have access to the entirety of the data stored. In these cases both data browsing and data analysis are made difficult, and the expressivity is close to non-existent as users cannot fully interact with these systems.

Providing explanations guarantees that any query will present some type of answer to the user: an empty result set that is *justified*, along with recommendations of query relaxations yielding results is far better than simply returning an empty set. In the case of very large answer sets, pointing out why these are large and submitting additional selection conditions to strengthen the query can be helpful to enable users to grasp the influence of each selection condition. These examples of cooperative behaviors contribute to the robustness of database systems. The informative nature of explanations is higher than that of actual answers in several cases, *e.g.* explaining the cardinality of a result set (why it is empty or plethoric) as well as proposing solutions on how to attain an acceptable answer.

**Example 1.1** *A user queries for "cheap burgers restaurants in Lannion" and obtains the answer* `Empty set`*. As there are no results, the user may wish to understand which terms from the query contributed to yielding an empty set (*e.g. *"cheap"), and obtain some possible query relaxations that yield some results (*e.g. *extending the query to other cities around Lannion).*◇

The informativeness illustrated above consists in providing *some answer* to users in cases where there are no direct results for instance. Yet explanations also require a clear formulation to be of interest to users. Using terms from natural language to describe data — with labels from fuzzy vocabularies — contributes to the *interpretability* of explanations. This property is especially important in cases where users do not — or cannot — grasp the entirety of the database. Offering to define and refine vocabulary terms increases the personalization experience and improves the interpretability by using the user's own words.

**Example 1.2** *In the above example the modality "cheap" is used to qualify the price values expected by the user: this user-defined modality contributes to the interpretability of the query for the user.*◇

Black boxes govern some aspects of our lives. Be it the general admission process for higher studies in France, or the loan granting process from banks, both public services

and private companies employ decision-based systems that affect everyday lives. These systems follow unknown algorithms that are not — yet — publicly shared and cannot be reviewed. Mostly, explanations are never provided regarding the decisions taken by these systems. Facing the general lack of explanations in these situations, there is a call for transparency and openness in the decisions rendered by public service systems.

In this thesis we propose to investigate the use of explanations in a cooperative answering setting using three research axes:

- In the presence of a plethoric set of answers: users cannot efficiently scroll through sets of answers too large with ordinary means. By default, results that fully satisfy a given query are returned in no particular order, as they all have the same ranking. To provide users facing such sets of results with more comfort and information, we propose to resort to clustering algorithms to identify subsets of data (clustered according to attributes relevant for the user) and then describe them with terms of a fuzzy vocabulary associated with the database considered. Furthermore we propose to characterize the subsets of data identified so as to enable users to select which subset would interest them most based on attributes *not present in the original query* (possibly explaining some of the obtained clusters).

  **Example 1.3** *A user wishes to buy a second-hand car that is recent and of European make "Renault", and is interested in its price and mileage. The query yields an overwhelming number of results. A clustering algorithm is applied on these results based on the attributes of interest: price and mileage. The obtained clusters are described according to these attributes with terms from natural language, and characterized with the other attributes present in the database. The user may discover for instance that one cluster is specifically characterized by* recent *cars, and that another is characterized by having* many options. *One of these characterizations may be of interest to the user, who may desire to focus on one of these clusters.*

- In the context of recommendations: when interacting with websites (Amazon, YouTube, etc.) or even with their TV (Netflix) users are now provided with recommendations as to which items they should browse or which series they should binge-watch next. While the purpose of recommendations is obvious (increasing user engagement to keep their business), their formulation and their impact vary greatly depending on how they are delivered. Raw recommendations without any form of explanation may come across as unwanted or puzzling for instance. On the opposite, explanations accompanying recommendations tend to increase the trust of users in recommenders. We propose to focus on the computation of explanations for association-based and demographic-based recommendations that both use the notion of typicality.◇

  **Example 1.4** *Let us consider a user who likes Terminator and Tron, and that these movies are usually liked by young men in college. This frequent association between movies and demographics can be quantified by a typicality degree: young*

*and male are two typical properties of the audience of these two movies. The recommendation process may leverage these typicality links to provide the user (who is not necessarily a young man) with other movies appreciated by young men.*◇

- In the context of a query/answering system: we propose Fuzzy Query by Example, a browsing method based on the evaluation of examples to obtain results. Users are tasked with positively or negatively evaluating examples, which are then characterized so as to bring out the properties shared by *positive examples* and those shared by *counter-examples*. These properties are derived as user preferences from which a query is generated to search for satisfying results. These user preferences are expressed with linguistic variables, and as such constitute an *on-the-fly explanation* of the generated query. The obtained results are then submitted to the user.

  **Example 1.5** *Consider a user browsing accommodations in an unfamiliar country and who is not able to easily formulate a selection query. By asking this user to positively evaluate example houses that possess desired properties and negatively evaluate those deemed uninteresting, we propose to infer preferences from these evaluations. We may discover that this user is interested in houses that have* many bedrooms *and that have a* big garden. *We formulate a fuzzy query based on these preferences and return a set of houses matching the inferred preferences.*

The three research axes mentioned above are all cooperative approaches revolving around explanations whose purpose is to keep the user in the loop. The computation of explanations in addition of results or recommendations aims at providing more transparent results. Explained or justified results are more interesting for curious users likely to question them: informativeness is another key to cooperative answering. We propose to study explanation mechanisms for database results in the three above axes considered.

## Structure of this Document

**Chapter 2** introduces the background notions used throughout this thesis, such as relational databases, fuzzy set theory, and their union: fuzzy databases.

**Chapter 3** covers cooperative answering behaviors and state-of-the-art approaches.

**Chapter 4** presents ClusterXplain, an approach helping users to tackle plethoric answer sets. We define the notion of *characterization* to highlight interesting properties in clusters and we evaluate the performance of our approach. These results were published in [Moreau et al., 2015, Moreau et al., 2016a, Moreau et al., 2016b].

**Chapter 5** presents two recommendation approaches, with their associated explanations. We propose ReSO, an association-based approach that uses the links between entities to find associations, and TyD, a recommendation approach leveraging demographics and user ratings. We evaluate these two approaches w.r.t. classical recommendation approaches. These results were published in [Moreau et al., 2017].

**Chapter 6** presents Fuzzy Query by Example, an approach helping users browse databases with examples. We propose a method to determine examples of interest to evaluate in a dataset, and we determine how to build a fuzzy query retrieving elements of interest based on evaluations. We evaluate our example selection method with regard to precision and recall. These results were published in [Moreau et al., 2018].

**Chapter 7** provides a thesis summary and a discussion on different perspectives of this work and its possible extensions to other contexts.

# Chapter 2

# Background Notions

In this thesis we consider relational databases, and the following chapters will cover cooperative answering approaches over relational databases. Many approaches presented afterwards rely on fuzzy set theory. In order to familiarize the reader with both relational databases and fuzzy set theory, in this chapter we provide the background notions and some notations necessary to understand the rest of this document. Section 2.1 introduces relational databases, Section 2.2 introduces some key notions of fuzzy set theory, and Section 2.3 fuses the two domains to introduce fuzzy databases [Pivert and Bosc, 2012].

## 2.1 Relational Databases

Edgar F. Codd first described the relational model in 1969 in an IBM research report, before publishing it in the ACM Communications [Codd, 1970]. The relational model features a tuple-based data representation into relations: it is the cornerstone of relational databases. Databases refer to both the data as well as the underlying schema structuring the data. Relational databases are managed with relational database management systems (RDBMSs), and are manipulated with the SQL language, based on relational algebra.

### 2.1.1 The Relational Model

**Definition 2.1 (Domain)** *A domain is a set of values (of a given data type such as integer, string, etc.).*

**Definition 2.2 (Attribute)** *An attribute is a variable taking its values in a domain.*

**Definition 2.3 (Tuple)** *A tuple is an element of the Cartesian product of the domains.*

**Definition 2.4 (Relation)** *A relation is a set of tuples that corresponds a subset of the Cartesian product of the domains.*

**Relational Algebra** Codd introduced relational algebra, stating that "relations are sets," and that as a consequence "set operations are applicable to them." Furthermore, tuples are organized into relations: they are structured, enabling the use of relational operators on them. Relational algebra has set operators and relational operators which have relations as inputs and return relations as outputs.

**Set operations** include union, intersection, difference, and Cartesian product. The first three operations (union, intersection, and difference) require that the operands be *compatible*: the relations must contain the same sets of attributes, as these operations are carried over the tuples and produce a relation with the same schema. The union of two relations $r$ and $s$ union$(r, s)$ is the union of the tuples of $r$ and the tuples of $s$ without any duplicates:

$$\text{union}(r, s) = \{t \mid t \in r \textbf{ or } t \in s\}.$$

The intersection of two relations $r$ and $s$ is the set of tuples that are shared by both $r$ and $s$:

$$\text{inter}(r, s) = \{t \mid t \in r \textbf{ and } t \in s\}.$$

The difference of two relations $r$ and $s$ is the set of tuples from $r$ without the tuples form $s$ that are in $r$:

$$\text{differ}(r, s) = \{t \mid t \in r \textbf{ and } t \notin s\}.$$

The Cartesian product of two relations $r$ and $s$ returns the tuples formed by the composition of each tuple from $r$ with each tuple from $s$:

$$\text{cartprod}(r, s) = \{t = \langle u, v\rangle \mid u \in r \textbf{ and } v \in s\}.$$

**Relational operations** include selection, projection, join, and division. The selection operation aims at extracting a subrelation from an input relation by limiting the tuples to those that satisfy a condition. If $r$ is a relation with schema $R(X)$ and $p$ is a predicate referring to attributes of $X$, then the selection operator is defined as follows:

$$\text{select}(r, p) = \{t \mid t \in r \textbf{ and } p(t)\}.$$

The projection operation aims at presenting tuples according to a subset of desired attributes from those initially present in the input relation. If $r$ is a relation with schema $R(X)$ and $Z$ is a proper subset of $X$ ($Z \subset X$), then the projection of $r$ onto $Z$ is defined as:

$$\text{project}(r, Z) = \{z \mid \exists t,\ t \in r \textbf{ and } t.Z = z\},$$

where $t.Z$ denotes the value of $Z$ in tuple $t$. The join operation stems from the Cartesian product, connecting two relations and returning the tuples formed by the tuples from the relations that satisfy a given condition $\theta$:

$$\text{join}(r, s, \theta, A, B) = \{\langle u, v\rangle \mid u \in r \textbf{ and } v \in s \textbf{ and } \theta(u.A, v.B)\},$$

where $R$ (resp. $S$), the schema of $r$ (resp. $s$), is defined over $X$ (resp. $Y$), $A$ and $B$ are two *compatible* respective subsets of $X$ and $Y$, $\theta$ is a comparator, and $\langle u, v \rangle$ denotes the tuple gathering the elements $u$ and $v$. In practice the comparison operator is often the equality, and it is possible to define a variant called equijoin:

$$\text{equijoin}(r, s, A, B) = \{\langle u, v' \rangle \mid u \in r \textbf{ and } v \in s \textbf{ and } (u.A = v.B) \textbf{ and } v' = v.(Y - B)\}.$$

Other join variants include the self-join (linking a relation with itself), the natural join (linking two relations based on all attributes of the same name), the antijoin (linking two relations and returning tuples from the first relation that cannot be matched with the second), and the semi-join (linking two relations, returning all tuples from the first one that have a match in the second).

The division operator consists in partitioning a relation, then checking an inclusion property using a second relation. If $r$ and $s$ are two relations with respective schemas $R(A, X)$ and $S(B, Y)$, where $A$ and $B$ are compatible (sets of) attributes, the division is defined as follows:

$$\text{div}(r, s, A, B) = \{x \mid \forall a, (a \in \text{project}(s, B)) \rightarrow (\langle a, x \rangle \in r)\}.$$

**Normalization** Databases containing redundant data face the risk of encountering anomalies with insertion, update or deletion operations. To avoid such situations, database design follows normalization rules.

**Functional Dependencies** Consider a relation schema $R$, and some sets of attributes of $R$ denoted $X, Y, Z$. The relation $r$ of schema $R$ satisfies the functional dependency $X \rightarrow Y$ if any two given tuples having the same values on the attributes of $X$ also have the same values on the attributes of $Y$:

$$\forall t_1, t_2 \in r, t_1.X = t_2.X \rightarrow t_1.Y = t_2.Y.$$

Functional dependencies are used to determine key attributes in relations, which in turn are used to determine the normalization level of a relation. A database schema $r$ can be modified in order to obtain a new database schema $r'$ that is in third normal form. Typical anomalies caused by insertion, update and deletion operations are usually avoided with the third normal form.

### 2.1.2 The SQL Language

SQL (Structured Query Language) was developed at IBM in the 70s' following the publication of Codd's relational data model. It serves both as a data definition language and as a query language. In SQL, projection-selection-join queries are expressed using the following base block:

**select** ⟨attributes⟩ **from** ⟨relations⟩ **where** ⟨condition⟩;

This base block enables expressing several relational operators.

**Example 2.1** *Consider the relation* ads(id, make, model, color, price, mileage, year, sellerid) *describing second-hand car ads. We desire to query for cars of make* Peugeot *and model 407 with a mileage lower than 50,000. The corresponding SQL query is as follows:*

```
select * from ads where make = 'Peugeot' and model = 407
  and mileage < 50000;
```

*where the first two selection conditions are equality-based and the last one is inequality-based.*◇

Several base blocks may be combined and *nested* to formulate more complex queries with the help of subqueries. Here are some of the possible forms of nested queries:

- *att-1* [**not**] **in** (**select** *att-2* **from**...) where *att-1* and *att-2* are compatible attributes;
- [**not**] **exists** (**select** * **from**...) this predicate is true if the result returned by the inner block is not empty;
- *att-1* $\theta$ {**any**|**all**} (**select** *att-2* **from**...) where *att-1* and *att-2* are compatible attributes.

**Example 2.2** *Consider the relations* ads(id, make, model, color, price, mileage, year, sellerid) *describing second-hand car ads and* sellers(id,name,location) *describing the sellers associated with the ads. We desire to query for cars that cost less than 9,000 and whose seller lives in Paris. The corresponding SQL query is as follows:*

```
select * from ads where price < 9000 and seller in
  (select id from sellers where city = 'Paris');
```

*Let us note that there exists an equivalent join query leading to the same answer set.*◇

Set operations introduced earlier may also be used in SQL:

(**select** *att-1, ... att-n* **from** *relation1* **where** *condition1*)
{**union**|**intersect**|**except**}
(**select** *att-1, ... att-n* **from** *relation2* **where** *condition2*)

Another operator enables the partitioning of a relation to directly obtain results that would otherwise require several subqueries. The partitioning block is:

**select** *att-set-1, ag-set* **from** *relations* **where** *ind-cond*
**group by** *att-set-2* **having** *set-cond*

where:

- *ind-cond* is a selection condition;

- *att-set-2* is the set of attributes onto which the partitioning is done;
- *set-cond* is a condition for the subsets obtained with the partitioning;
- *att-set-1* is a subset of attributes from *att-set-2*;
- *ag-set* is a list of aggregate functions applying to attributes of the partitioned relation.

**Example 2.3** *Consider the relation* ads(id, make, model, color, price, mileage, year, sellerid) *and assume we desire to obtain statistics over the ads in the database, such as the average price of each pair "make, model." This is obtained with the following query:*

```
select make, model, avg(price) from ads
group by make, model;
```

*In order for these statistics to be representative, we may wish to enforce an additional condition stating that there must exist at least 10 ads concerning the same pair "make, model" before returning its average price. This is done by adding the condition* ***having*** *count*$(*) >= 10.$\diamond$

## 2.2 Fuzzy Set Theory

Fuzzy set theory was introduced by Zadeh [Zadeh, 1965] for modeling classes or sets whose boundaries are not clear-cut. For such objects, the transition between full membership and full mismatch is gradual rather than crisp. Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as *young, cheap, fast*, etc. Formally, a fuzzy set $F$ on a referential $U$ is characterized by a membership function $\mu_F : U \to [0, 1]$ where $\mu_F(u)$ denotes the grade of membership of $u$ in $F$. In particular, $\mu_F(u) = 1$ reflects full membership of $u$ in $F$, while $\mu_F(u) = 0$ expresses absolute non-membership. When $0 < \mu_F(u) < 1$, one speaks of partial membership.

**Definition 2.5 (Support)** *The support of a fuzzy set $F$ denoted by $F_{supp}$ is the set of elements that somewhat belong to $F$,* i.e. *that have a strictly positive membership degree:*

$$F_{supp} = \{u \mid u \in U \textbf{ and } \mu_F(u) > 0\}.$$

**Definition 2.6 (Core)** *The core of a fuzzy set $F$ denoted by $F_{core}$ is the set of elements that fully belong to $F$,* i.e. *that have a membership degree equal to 1:*

$$F_{core} = \{u \mid u \in U \textbf{ and } \mu_F(u) = 1\}.$$

**Definition 2.7 ($\alpha$-cut)** *The $\alpha$-cut of a fuzzy set $F$ denoted by $F_\alpha$ is the set of elements that belong to $F$ that have a membership degree greater than or equal to $\alpha$:*

$$F_\alpha = \{u \mid u \in U \textbf{ and } \mu_F(u) \geqslant \alpha\}.$$

**Definition 2.8 (Normalized fuzzy set)** *A fuzzy set is said to be normalized when at least one of its elements attains the maximum possible membership value.*

Table 2.1 – Examples of t-norms and t-conorms

| t-norm | t-conorm | name |
|---|---|---|
| $\min(u, v)$ | $\max(u, v)$ | Zadeh |
| $xy$ | $x + y - xy$ | probabilistic |
| $\max(u + v - 1, 0)$ | $\min(u + v, 1)$ | Łukasiewicz |
| $u$ if $v = 1$<br>$v$ if $u = 1$<br>0 otherwise | $u$ if $v = 0$<br>$v$ if $u = 0$<br>1 otherwise | Weber |

### 2.2.1 Operations on Fuzzy Sets

**Intersection and Union** Intersection and union, respectively associated with the Boolean conjunction and disjunction, are interpreted with a triangular norm and conorm.

A t-norm $\top(x, y)$ is a binary operator with the properties of associativity, commutativity, monotonicity, and has 1 as its neutral element. Each t-norm is associated with a t-conorm by the following duality relationships:

$$\top(x, y) = 1 - \bot(1 - x, 1 - y) \text{ and } \bot(x, y) = 1 - \top(1 - x, 1 - y),$$

which correspond to the De Morgan laws in the usual set-theoretic framework, *i.e.*:

$$(a \text{ and } b) = (\overline{\overline{a} \text{ or } \overline{b}}) \text{ and } (a \text{ or } b) = (\overline{\overline{a} \text{ and } \overline{b}}).$$

A t-conorm $\bot(x, y)$ is associative, commutative, monotonic, and has 0 as its neutral element. The following inequalities hold for t-norms and t-conorms:

$$\forall y, \top(x, y) \leqslant x \leqslant \bot(x, y) \text{ and } \forall x, \top(x, y) \leqslant y \leqslant \bot(x, y).$$

The intersection and union of fuzzy sets are defined as follows:

$$\forall u \in U, \mu_{E \cap F}(u) = \top(\mu_E(u), \mu_F(u)),$$

$$\forall u \in U, \mu_{E \cup F}(u) = \bot(\mu_E(u), \mu_F(u)),$$

where $\top(u, v)$ (resp. $\bot(u, v)$) is a t-norm (resp. t-conorm).

Several t-norms and their corresponding t-conorms have been proposed, see for instance Table 2.1.

**Compromise Operators** The other Boolean operators, *e.g.* the set difference or the Cartesian product, also have fuzzy counterparts. There also exist fuzzy operators with no Boolean equivalents, capable of expressing a compromise between their arguments. The classical arithmetic or geometric mean can be computed, as well as the weighted

average. Let $W$ be a normalized fuzzy set containing weights (the sum of the weights being equal to 1), the weighted average of the fuzzy sets $E_1, ..., E_n$ is given by:

$$\mu_{(E_1,...,E_n)}(u) = \sum_{i=1}^{n} (\mu_{E_i}(u) * W[i]).$$

Yager proposed a dynamic alternative named OWA (Ordered Weighted Averaging) [Yager, 1988], in which the fuzzy set values $E_1(u), ..., E_n(u)$ are ranked in descending order. As such, the first weight $W[1]$ is applied to the highest $E_i(u)$ value:

$$\mu_{\mathrm{OWA}(E_1,...,E_n,W)}(u) = \sum_{i=1}^{n} (\mu_{E'_i}(u) * W[i])$$

where $\mu_{E'_i}(u)$ is the $i^{th}$ highest value among $\{\mu_{E_1}(u), ..., \mu_{E_n}(u)\}$.

**Inclusion** There are several interpretations of the inclusion between two fuzzy sets. One is Boolean, also termed "inclusion in the sense of Zadeh" with which in order to have $E \subseteq F$ all values of $E$ must be lower or equal to those of $F$:

$$(E \subseteq F) \Leftrightarrow (\forall u \in U, \mu_E(u) \leqslant \mu_F(u)).$$

A more gradual interpretation of the inclusion $E \subseteq F$ is based on cardinalities, such that we can define a degree quantifying the extent to which we have $E \subseteq F$:

$$deg(E \subseteq F) = \frac{card(E \cap F)}{card(E)} = \frac{\sum_{u \in U} \top(\mu_E(u), \mu_F(u))}{\sum_{u \in U} \mu_E(u)}$$

where $\top$ is a t-norm.

There also exists another interpretation of the inclusion that is based on fuzzy implications [Dubois and Prade, 1980]:

$$deg(E \subseteq F) = \min_{x \in X} (\mu_E(x) \Rightarrow_f \mu_F(x)).$$

### 2.2.2 Fuzzy Quantifiers

Beyond the universal ($\forall$) and existential ($\exists$) quantifiers, Zadeh introduced linguistic fuzzy quantifiers [Zadeh, 1983]. Some of them are absolute and express a quantity, such as "*a dozen*" or "*about 20*" and others are relative and express a proportion, such as "*most of them*" or "*just a few.*"

An absolute quantifier $QA$ is modeled as:

$$\mu_{QA} : \mathbb{N} \text{ (or } \mathbb{R}) \rightarrow [0,\ 1],$$

whereas a relative quantifier $QR$ is modeled as:

$$\mu_{QR} : [0,\ 1] \rightarrow [0,\ 1].$$

The degree $\mu_{QA}(x)$ (resp. $\mu_{QR}(x)$) expresses the adequation between the number (resp. proportion) $x$ and the considered quantifier.

Fuzzy quantified statements are statements of the form "*Q X are A*," where $A$ is a fuzzy predicate and $Q$ is a fuzzy quantifier (*e.g.* "most of the elements of $X$ satisfy $A$"). These aim at evaluating the extent to which the number of elements in $X$ satisfying the fuzzy predicate $A$ is in adequacy with the quantifier $Q$. There also exist quantified statements of the form "*Q B X are A*" ("most of the elements of set $X$ which satisfy $B$ also satisfy $A$").

Zadeh [Zadeh, 1983] defined the cardinality of the set of elements $X = \{x_1, x_2, ..., x_n\}$ which satisfy $A$ as:

$$\Sigma count(A) = \sum_{i=1}^{n} \mu_A(x_i).$$

The quantified statement "*Q X are A*" may then be evaluated as follows:

$$\mu(\text{Q X are A}) = \mu_Q(\Sigma count(A)),$$

while the quantified statement "*Q B X are A*" may be evaluated by:

$$\mu(\text{Q B X are A}) = \mu_Q\left(\frac{\Sigma count(A \cap B)}{\Sigma count(B)}\right) = \mu_Q\left(\frac{\Sigma_{x \in X}(\top(\mu_A(x), \mu_B(x))}{\Sigma_{x \in X}\mu_B(x)}\right).$$

More interpretations of fuzzy quantified statements were put forward by Yager, including the competitive-type aggregation [Yager, 1984] and the OWA-based interpretation [Yager, 1988], which respectively correspond to a Sugeno and a Choquet fuzzy integral [Bosc et al., 1995].

## 2.3 SQLf

Traditional SQL queries are sometimes limited and may fail to provide users with an adequate answer. Consider a user looking for a house with a conjunction of selection conditions such as:

$$\text{price} \leqslant 200000 \text{ and city} = \text{'Paris' and surface} \geqslant 100.$$

Such a query may fail and not provide any results, or return a great many results without any form of ranking. In both cases, values very close but outside the borders are not considered. Fuzzy set theory provides tools to enable flexible querying.

SQLf, a fuzzy set-based extension of SQL, was proposed in [Bosc and Pivert, 1995] with the objective to make SQLf as close to SQL as possible, by trying to maintain query equivalences. In SQL, projection-selection-join queries are expressed using the following block:

**select** ⟨attributes⟩ **from** ⟨relations⟩ **where** ⟨condition⟩.

In SQLf the structure of the projection-selection-join block somewhat remains the same:

**select** $[n \mid t \mid n, t]$ ⟨attributes⟩ **from** ⟨relations⟩ **where** ⟨fuzzy condition⟩.

The main differences w.r.t. the classical SQL base block concern:

- optional parameters to filters results ($n$ to specify the number of desired results, $t$ to specify a quantitative threshold, with the same principle as an $\alpha$-cut),
- the possibility to resort to fuzzy conditions.

More complex queries such as *nested queries* also have an SQLf counterpart: *e.g.* queries based on an imbrication based on **in** may be interpreted with the membership to a fuzzy relation and/or the fuzzy membership (to any sort of relation).

**Example 2.4** *Consider we desire to find cars with a price lower than 9,000 with a seller* close to *Paris. In the case of the membership to a fuzzy relation, the query may be written as follows:*

```
select * from ads where price < 9000 and seller in
  (select id from sellers where city is 'close to Paris');
```

*The attribute* city *is evaluated with the predicate* close to*, which may be represented by a decreasing fuzzy membership function.*

*Consider now that we desire to find cars with a price that is about the same as the average price of a* Renault Clio*. In the case of a fuzzy membership to a relation:*

```
select * from ads where price in_f
  (select avg(price) from ads where make = 'Renault' and model = 'Clio');
```

*where* `in_f` *expresses the fuzzy membership to a set.*◇

The partitioning of a relation is also possible with SQLf, by using a fuzzy-set-oriented condition in the *having* clause aimed at the selection of groups.

**Example 2.5** *Consider we desire to obtain statistics over the ads in the database, such as the number of ads of each pair "make, model" whose average price is close to 10,000. This is obtained with the following query:*

```
select make, model, count(*) from ads
group by make, model having avg(price) around 10,000;
```

*Aggregate functions in SQLf include those in SQL (*min, sum*, etc.), as well as more complex conditions involving fuzzy quantifiers.*◇

# Chapter 3

# Introduction to Cooperative Answering

Following the background notions this thesis is based on, in this chapter we propose to draw an overview of cooperative answering principles as well as their declination over relational databases. This chapter aims at positioning the contributions presented in the subsequent chapters w.r.t. the state of the art.

Providing users with helpful and detailed answers is called cooperative answering [Gaasterland et al., 1992]. Instead of simply fulfilling a user query, the system tries to answer the query better, possibly interpreting it differently from how it was specified. The objective is to understand the motives of users in order to provide them with the answers they expect, which can be quite different from the ones they asked for. Even better, to offer the user additional information that was not asked for, but nevertheless could be of interest to them. As such, cooperative answering may be seen as a communication process between users and systems.

This chapter covers the fundamental principles of cooperative answering in Section 3.1. Then the applications of cooperative answering to relational databases are covered in Section 3.2. A chapter conclusion preparing the ground for the other chapters is drawn in Section 3.3.

## 3.1 Fundamentals of Cooperative Answering

Cooperative answering techniques can be divided into five categories, depending on whether they concern:

1. Users' beliefs and expectations;
2. Presuppositions;
3. Misconceptions;
4. Intensional answers;

5. Generalization of queries and answers.

Each of these categories is presented in further detail. According to Grice's maxims [Grice, 1975], useful and non-misguiding answers are the key. These maxims are:

- Quality: avoid false assumptions;
- Quantity: avoid unnecessary and redundant information;
- Relation: find users' goals and intentions to determine the relevance of answers;
- Manner: avoid ambiguity and overly long answers.

We will firstly classify and review some cases where cooperative answering techniques would be welcome.

### 3.1.1 Beliefs and Expectations

In order to establish an efficient dialogue between users and systems, both need to understand the query and the answer. Since natural language is ambiguous and sentences may carry many meanings, systems have to figure out the intentions of the users. When several interpretations of the user's intentions exist, taking into account the reaction of the user should guide the system towards the intended option. The goals of users tend to change over time, as a dialogue is established between users and systems. Upon learning that some queries have no answer, users reformulate and try to adapt to the data. Systems should get ahead of them — pre-empt, forestall — and have a better insight on the intentions of the user as more queries are posed. For instance, users may have *beliefs* that are in opposition with reality:

**Example 3.1** *Q: "Is Sam an associate professor?"* – A common belief could lead the user to believe that most associate professors have tenure. However Sam is not tenured: he is an associate professor.
*A: "Yes, but he doesn't have tenure."*◇

In this example the system explicitly corrects the user's belief by highlighting the fact that Sam is not tenured. On the other hand, users may have different *intents* when posing queries:

**Example 3.2** *Q: "How did John take the exam?"* – Intent: ?
*A1: "He crammed the night before."* – enablement
*A2: "He took it with a pen."* – instrumental/procedural
*A3: "He took it badly."* – emotional.◇

As illustrated above, some questions may have very different answers based on their context. The context of the interaction between the user and the system should be taken into account in order to understand the meaning of the question.

### 3.1.2 Presuppositions

Users can pose queries while presupposing the existence of elements in the database, leading to empty sets of answers. Users seeking the names of participants of a course that never took place over a specific period of time should be informed of this fact, instead of simply returning an empty set due to the false presuppositions of the user.

**Example 3.3** *Q: "Which students took Introduction to Databases this Spring?"* – Introduction to Databases is a Fall class, not a Spring class.
*A:* $\emptyset$ *– "There are no Introduction to Databases classes in Spring."*$\diamond$

False presuppositions may render a query meaningless: in the example above, the user presupposes that there *was* an Introduction to Databases class in spring. Indeed, seeking the results of a specific course presupposes that this given course exists to begin with. However the example below presumes that there is at least one Introduction to Databases class in spring, but it does not presuppose so. The answer "no" is correct and meaningful in both cases (whether or not there was such a class in spring).

**Example 3.4** *Q: "Did any students pass an Introduction to Databases class this spring?"* – No student passed the class.
*A: "No." – "All students failed."*
*Q': "Did any students pass an Introduction to Databases class this spring?"* – Introduction to Databases is a fall class, not a spring class.
*A': "No." – "There are no Introduction to Databases classes in spring."*$\diamond$

Although they are quite similar, presumptions are not the same as presuppositions [Kaplan, 1982]. Presumptions are more complicated to handle as they need more reasoning: in the example above a false presumption leads to two identical answers in two different cases. A solution to help remove these – and find more than an empty set – may be to generalize the query, if it is possible to remove the failing part of the query. The answers returned will still be related to the original query formulated by the user. To follow the maxim of quantity, information should be given in correct amounts. Indeed, there is no need to specify that "No employees own red cars" if in fact no employees own a car at all. If a subquery does not have any answers, then the original query will not have any either. This implication is called a scalar implicature [Gaasterland et al., 1992].

### 3.1.3 Misconceptions

False presuppositions can stem from the schema of the database. However, misconceptions have more to do with the domain and the semantics. Considering academia, people can have several kinds of relationships with a course. Students take courses, and teachers give courses. A false presupposition would be to ask who failed a class that has not taken place (see Example 3.3). A misconception would be to ask "which courses does one teacher take," or "which classes does one student teach."

**Example 3.5** *Q: "Which students teach Advanced Databases?"*
*A: "None." – "Students do not give classes. Professors do."*◇

An exception would occur in the case of Ph.D. students who can both take and teach courses (although usually on different levels). Such misconceptions are then said to be object-related as the relationships between objects usually prevent such mistakes from occurring. In any case, explaining the cause of the failing query would be a cooperative answer.

### 3.1.4 Intensional Answers

A distinction must be made between data and knowledge. Direct answers to queries come from the extension of the database, namely its content, which is the data populating it. The structure of the database — such as the schema, the views, and the integrity constraints — form the intension of the database. Intensional answers could be viewed as summaries of results, better informing the user. When asking "which first year students take Introduction to Databases?", if all the first years take this class, then simply stating "All first year students take Introduction to Databases." is more informative than listing the names of all first year students at once.

Intensional answers may vary in quality based on relevance, optimality, completeness, nonredundency, and efficiency. A survey of intensional answers was published in [Motro, 1994], covering work on relational, logic-based, semantically-rich and object-oriented frameworks.

### 3.1.5 Generalizations

Generalizing a query consists in extending its scope in order to give more information to the user. This can be done through query relaxation or attribute addition for instance. Query relaxation consists in returning answers that are not in the direct answer but *close enough* to be of interest to the user. Attribute addition consists in adding information carried by attributes not originally present in the query. On a similar level, follow-up questions to yes/no questions may also be anticipated with domain knowledge, thus adding information not *explicitly* requested:

**Example 3.6** *Q: "Has a turquoise car gone by?"* – The system assumes the user also wants to know where it went by.
*Q′: "Has a turquoise car gone by? If so, where?"*
*A′: "Yes, one went by on Oxford Street."*◇

Query abstraction and refinement in RDBMSs [Chu and Chen, 1994] are based on hierarchies of relations and domains. These are called "abstraction hierarchies." To obtain broader answers, such systems replace query terms with other ones that stand higher in the abstraction hierarchy. A query browsing for "the average marks of undergraduate students majoring in computer science" may be generalized into a query browsing for "the average marks of all undergraduate students," or again "the average marks of all students altogether."

### 3.1.6 The Predominant Role of the Users

The key to cooperative answering resides in the interaction between the user and the system. While users should try to make their query as plain and clear as possible, the system should also be able to infer as much as possible the expectations of the user in the broadest possible way. The system could use the following about users:

- interests and preferences;
- needs;
- goals;
- context.

Interests could be taken into consideration to filter information based on the user's fields of interest. Preferences could be taken into account so long as they do not come into conflict with the conditions of the query. They may also be used to rank the returned answers when ranking is possible. Needs differ depending on the user's knowledge of the domain under consideration. A new user may not require very technical information, while a specialist will not be interested in generalities. Children and adults may not expect the same answer when asking the same question.

**Example 3.7** *Q: "What is a circle?"* – The system takes into account the age of the user in its answer.
*A1: "A circle is a shape similar to a DVD."* – Possible answer for children.
*A2: "A circle is a plane figure bounded by one line, and such that all right lines drawn from a certain point within it to the bounding line, are equal."* – Possible answer for adults, from [Dodgson, 1883].◇

User constraints [Gaasterland et al., 1992] can represent needs, and dictate whether an answer is acceptable or not to the user. While they are syntactically similar to integrity constraints, the latter have to hold wrt. the data in the database, while the former does not. Goals and intent reflect the use of "sessions" to separate the different uses a user may have with the system. Past interactions between the user and the system may be exploited if the user has already had similar goals in a previous "session." Misinterpreting the current intent of the user may lead to undesirable results. Conversing with the user to elucidate his/her goals may help resolve misconceptions, by explaining how the beliefs of the system and those of the user may be in conflict [McCoy, 1988, Quilici et al., 1988].

## 3.2 Cooperative Answering in a Relational Database Context

An overview of cooperative answering in relational databases is given in [Minker, 1998]. The cooperative behaviors mentioned in this paper are covered above in Section 3.1. In this section we present some of the more recent efforts towards cooperative answering in relational databases.

Subsection 3.2.1 covers some of the existing tools to facilitate the navigation of users in relational databases, as well as taking into account their preferences. Then Subsection 3.2.2 details the current state of research on unsatisfactory results such as the many answer problem (too many results), the empty answer problem (no results at all), and the missing answer problem (absence of some expected results). Subsection 3.2.3 gives an overview on the computation of additional answers in relational databases. Finally Section 3.2.4 focuses on one interaction between users and systems: explanations.

### 3.2.1 Browsing assistance

Beyond formal query languages, there are other query paradigms: for instance keyword search [Simitsis et al., 2007], faceted search [Stoica et al., 2007], graph browsing, image browsing, query by example, natural language querying [Li and Jagadish, 2014], query editing [Koutrika and Simitsis, 2013], etc. All of these query paradigms have been introduced to help users browse databases in the best possible way, and as intuitively as possible. However, there is a noticeable trade-off between easily formulating queries — *e.g.* with keyword search — and getting precise and correct answers — *e.g.* with a dedicated query language such as SQL. In other words, each of these paradigms has its advantages and drawbacks. Below we review some cooperative paradigms dedicated to helping users handle databases, such as summaries and Query By Example.

**Data and Schema Summaries** Querying a new database for the first time is simple when the schema is small and easy to understand. But in cases where there are dozens or hundreds of tables, querying becomes much more difficult. Schema summarization [Yu and Jagadish, 2006] helps users discover new schemata. A schema summary should be magnitudes smaller, easy to grasp, highlighting important elements, and enabling the user to perceive the bigger picture from *abstract elements*. As a schema is made up of tables and links between these tables, a schema summary consists of *abstract elements* representing aggregates of these tables as well as *abstract links* representing the links between these aggregated tables. *Important elements* possess properties such as high connectivity and high cardinality. Schema summaries should feature *important elements* while ensuring that the overall schema is appropriately *covered*.

**Example 3.8** *The authors of [Yu and Jagadish, 2006] illustrate their approach on XML schemata with the XMark benchmark (about auctions, represented in Figure 3.1). Figure 3.2 illustrates full schema summaries (A and B) of the XMark schema as well as an expanded schema summary (C) based on A. Elements in summaries A and B are all abstract elements (except the root element site). Dashed boxes in C represent abstract elements. Dashed arrows indicate abstract links. Both schemata A and B contain the same number of elements, however A appears to be a better summary as it conveys the notion of "bidder" which is central in the context of auctions, while B features the notion of "region." The summarization process enables expanding a summary, such as a user wishing to focus on "open auctions" on the schema A will obtain the expanded schema C.*◇

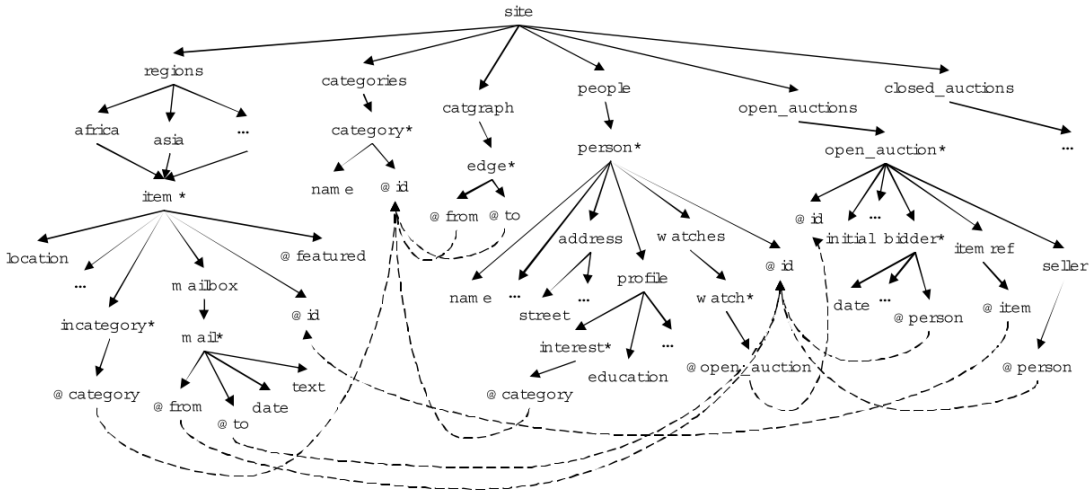



Figure 3.1 – Example schema based on XMark, from [Yu and Jagadish, 2006]

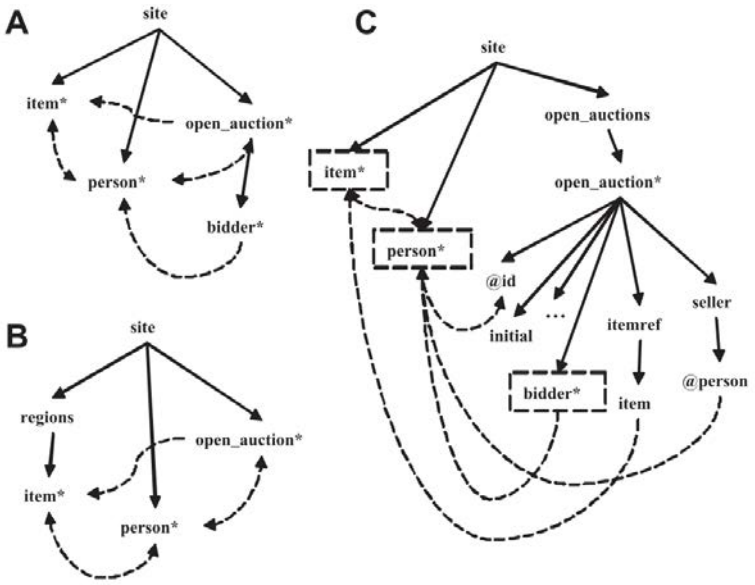



Figure 3.2 – Schema summaries from [Yu and Jagadish, 2006]

Content and schema summarization is investigated in [Yang et al., 2009]. In their paper, the authors define the *importance* of tables in the database in order to rank them. This *importance* takes into account the content of the table (its size, its attributes) and how it is related to other tables in the database. The main difference with the *importance* in [Yu and Jagadish, 2006] is that here the authors take into consideration the entropy of table attributes. The authors also define a distance measure between tables, so as to be able to: (i) compare them by formally considering the different join types, and (ii) apply a clustering algorithm on them to regroup them. The authors of [Yang et al., 2009] compare their approach to that of [Yu and Jagadish, 2006] on relational database schemata, using the TPCE benchmark.

SaintEtiQ [Ughetto et al., 2008] is a data summarization system which also offers a personalized querying process. Users can select words from their own vocabulary to query the database with the help of summaries. Vocabularies are composed of linguistic variables that describe a domain of numerical values, and may be represented with fuzzy sets by trapezoidal membership functions, see for instance the representation of the domain of the attribute *hardness* in the context of materials in Figure 3.3. These summaries have several levels of granularity, and are built hierarchically so that users can explore a tree-like structure until the adequate level of granularity is found. This enables querying a database through a summary, while also being able to rewrite terms from the vocabulary (thus rewriting the query). One example of such rewriting is given in Figure 3.4 where the term *medium* is rewritten with the terms *soft or hard*.

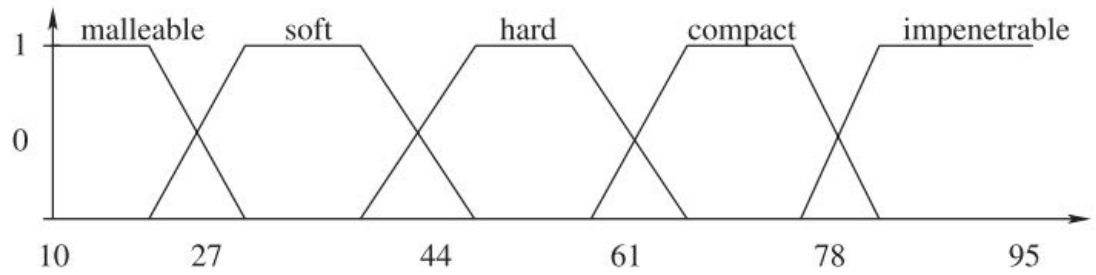



Figure 3.3 – An example of partition associated with the domain of the attribute hardness, from [Ughetto et al., 2008]

In [Smits et al., 2017b] the authors propose an interactive data exploration approach that relies on two steps. The first part consists in building a personalized linguistic summary of the dataset, before displaying it as a tag cloud. The second part focuses on enabling users to use exploration functionalities on top of the summary so they may discover properties of interest in the data, such as frequent, atypical, or diversified associations between properties.

**Query By Example (QBE)** Query by Example was the first graphical query language, developed at approximately the same period as SQL at IBM Research [Zloof,

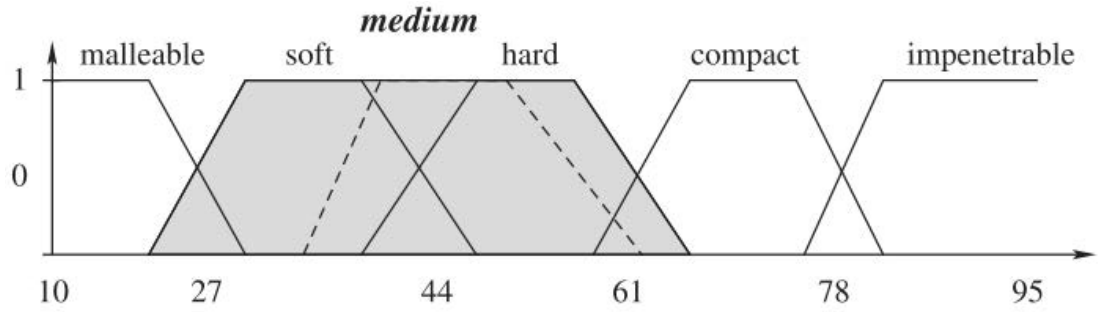



Figure 3.4 – A (hierarchical) rewriting of a term based on two adjacent terms in a partition, from [Ughetto et al., 2008]

1975, Zloof, 1977]. It uses a graphical representation of the tables and provides simplistic means of formulating queries by filling boxes or using scroll menus. It is possible to select "constant elements" (which correspond to the selection part of the query) and "example elements" which vary.

**Example 3.9** *For example, a user looking for "red items" may input as a "constant element" the attribute color "red," and as an "example element" the attribute item type "pen." Results will include other items in the database available in the color "red."*◇

Beyond the query language, Query by Example is first a paradigm in information retrieval to acquire results based either on:

- an (or several) input tuple(s);
- or the (positive, negative, on a predefined scale, ...) evaluation of prototypical examples reflecting the content of the database.

The resulting output returns elements that are similar to the input tuple provided as an example, or that reflect the choices of the user if prototypical examples were evaluated.

**Example 3.10** *For example, if a user browses houses and positively evaluates houses with 3 bedrooms and negatively evaluates houses with a small garden, then the results will include houses with 3 bedrooms and a "not small" garden.*◇

A variant of QBE using uncertainty is presented in [Tatemura et al., 2008], to reflect the user's ignorance of the combined schema of multiple sources. The user provides tuples of interest to them and chooses the components of the output schema. The system has to match the different data sources and provide queries to the user that satisfy their requirements. The user then helps the system to refine the query by evaluating the presented results.

In [De Calmès et al., 2003] the authors present a fuzzy logic-based QBE approach. They ask the user to evaluate (on a scale, positively or negatively) prototypical examples from the database — which may be fictitious, as long as they reflect the contrast between

the attribute values of the items in the database. Based on these evaluations, a query is generated to look up items that are "similar to at least one example (w.r.t. all the attributes) and which are dissimilar to all counter-examples (each time w.r.t. at least one attribute)." Each example and counter-example can be weighted and so can the different attributes of the database, in order to obtain answers as representative as possible of the user's choices. The authors point out that the counter-examples have a limited influence on the result: in general they intervene only when the user has not selected any positive examples, or has made incoherent choices when reviewing examples.

The authors of [Zadrozny et al., 2010] present an approach inspired from QBE. Assuming that some users do not necessarily know how to browse a database system, they are presented with items from the database and asked to sort them positively or negatively. The evaluation can be global — on the whole item — and local — attribute by attribute. They use the $k$-nearest neighbors algorithm to find the next examples to evaluate, based on those freshly evaluated by the user. A model of the user's profile is derived from their choices, so the user can check whether the system correctly interprets the user's choices. This profile is obtained by determining which linguistic labels are relevant to the user — computing how many times each label was among the positively evaluated examples. The computation of this user profile is carried out as an entirely different step from the computation of the next examples to evaluate, although the authors believe that this profile computation could also provide the next examples to evaluate.

### 3.2.2 Dealing with Unsatisfactory Results

Even when using an adequate and dedicated query language, users do not always obtain the results they expected. Should their queries be too specific, query relaxation methods can help them resolve the empty-answer problem. On the contrary, if they were too large, then summaries of the answer set or additional selection conditions could help them tackle the plethoric-answer problem. Moreover when the answer does not contain some expected results, why-not queries are able to shed some light on their absence.

**Empty Answer Set** By reconsidering the crisp border between true and false in Boolean logic, fuzzy logic increases the number of query results by also returning answers that *somewhat* satisfy the query conditions, in addition to the results fully satisfying the query conditions (if there are any).

**Example 3.11** *Let us consider that the user is interested in cars, and that given a high number of selection conditions there are no results to display. Let us consider that the condition "year is very recent" was specified. With a crisp query, this condition may be interpreted as "year $\geqslant$ 2015." However, the user may also accept results with a year close enough to 2015, such as 2014 or 2013. Fuzzy queries will also return these results — albeit with a lower degree — and order them according to how much the selection conditions are satisfied.*$\diamond$

These results are ordered by decreasing membership degree so as to review the best results first. SQLf [Bosc and Pivert, 1995] is a fuzzy extension of SQL proposed to handle such queries. Let us note that fuzzy queries may also yield an empty answer set, and [Bosc et al., 2008] proposed a relaxation method to obtain results for these queries. Indeed, while fuzzy queries increase (and rank) the number of results returned, they do not eliminate the risk of returning an empty answer set. Another approach using a summary of the database leveraging fuzzy cardinalities proposes to explain failing queries [Smits et al., 2014a], and to extend answer sets containing only elements with a low degree. *Minimal failing subqueries* help users understand the issues in their queries, so they may revise their original queries.

To help users avoid empty answers to their queries, the authors of [Koudas et al., 2006] proposed a framework to automatically relax selection and join conditions based on numeric values. They use a lattice-based approach to determine which conditions should be relaxed to acquire results, while conserving the query as close as possible to the original. The different possible relaxations are compared with a partial order inspired from the concept of "skyline," namely a relaxation skyline. The relaxation skylines for all selection conditions are computed separately, so there is at least one tuple returned each time.

Many relaxation methods which only take into consideration selection conditions are referred to as "top-k" [Agrawal et al., 2003, Ilyas et al., 2008]. They use a scoring function to sort which tuples would best correspond to relaxed queries based on to what extent each condition would need to be relaxed. These approaches consider basic selection conditions of the form *"attribute = value"*, as well as more general conditions such as *"attribute IN [...]"*.

Other methods are based on "abstract hierarchies" [Chu and Chen, 1994] as detailed in Subsection 3.1.5.

**Plethoric Answer Set** Fuzzy queries return ordered results, which are very helpful when dealing with *somewhat* satisfactory answers. However even fuzzy queries may lead to a plethoric answer set, with all answers somewhat (or fully) satisfying the selection conditions. A solution is to ask users to strengthen their queries so as to increasingly limit the number of acceptable results. These results can also be reduced by providing an $\alpha$-cut that will discard results that are not satisfactory enough. Other fuzzy approaches to handle plethoric answer sets include [Bosc et al., 2010], which is based on query expansion. This approach may be applied in the case of *under-specified queries*, where some additional selection conditions could be added to the query, with the objective to get to a reduction of the answer set. The candidate selection conditions to add to the query are determined by their degrees of correlation with the query.

Aside from query strengthening, other approaches focus on summarizing the answer set, and consider clustering to do so whenever a distance metric is available to compare the results, *e.g.* [Liu and Jagadish, 2009]. In this case the authors offer the users the possibility to refine their results by presenting the most representative answers, *i.e.* the medoids obtained with the clustering algorithm.

In addition to being confronted with too many answers to queries, users sometimes

have difficulties browsing datasets for other reasons. These include attribute values that may not be easy to apprehend, or that all attributes may not be available for querying. Some systems offer specific views [Singh et al., 2016] to filter the results with a clustering algorithm, as in the following example.

**Example 3.12** *Let us consider a user interested in second-hand cars, represented by the relation* cars(price, make, model, bodyType, drivetrain, mileage, engineSize, year, *etc.*). *The user wishes for a relatively recent SUV car. The submitted query is "select * from cars where mileage between 10000 and 30000 and bodyType = SUV" which yields thousands of results. Using a clustering algorithm, it is possible to provide the user with representative elements of the results. Based on these representative elements, the user may choose to focus on one cluster of elements in particular, refine his/her query in the process and reduce the number of results to browse. This process may be reiterated until the number of results is small enough to browse manually. One such representative element could be (make = Chevrolet, price = [25K-30K], year = [2011-2012],* etc.*).*◇

Interactive query refinement is applicable to both the "too many" and the "too few answer" problems. The framework "Stretch 'n' Shrink" [Mishra and Koudas, 2009] helps users get an answer set of the desired cardinality, regardless of the many/few answer problems. To do so, the approach estimates the cardinality of all possible refinements of a query with a "Superset Sampling Estimator."

**Why-Not Answers** The missing answer problem has been addressed in [Herschel, 2013], to provide explanations to users as to why the items they expected are not part of the answer set. This study summarizes previous work with the same objective but different ways of explaining the absence of expected results. Three kinds of explanations have been used separately to deal with the missing answer problem: instance-based [Herschel and Hernández, 2010], query-based [Chapman and Jagadish, 2009], or modification-based [Tran and Chan, 2010]. Instance-based explanations consist in updating the data source so that running the query again will yield the missing answer. Query-based explanations consist in finding which query operator(s) removed the expected tuple from the result. Modification-based explanations first verify whether or not the expected answer can be computed from the data sources, and then modify the original query so that it includes the missing answers. In [Herschel, 2013], Herschel introduces hybrid explanations mixing all of the above with the Conseil algorithm.

### 3.2.3 Additional Answers

In order to browse database elements, users usually submit queries. In some cases, such as keyword-based search engines, users may receive additional results that are not in the direct answer to their queries. These additional answers aim at providing content of interest to users, by taking into account the previous interactions between the user and the database such as the query history.

In other cases, users do not submit queries but provide systems with information of their own, such as item ratings, or personal information. Recommender systems (RS)

then do the rest by providing users with content of interest, by comparing items, their ratings, or even users.

#### 3.2.3.1 Association-based and Typicality-based Additional Results

Additional answers are computed on top of a query returning direct answers. They are computed by looking for results similar to those in the direct answer. Such a similarity can be computed by taking into account the domains of the items [Kato et al., 2012]. The authors of [Stefanidis et al., 2009] coined the term of "You May Also Like" answers to refer to recommendations from relational databases. They listed several types of recommendations based on the origin of their computation:

1. Current-state approaches, that use the content of the query result (local analysis) and/or the schema of the database (global analysis);
2. History-based approaches, that use previously submitted queries or obtained query results;
3. External sources approaches, that use data from other sources such as DBpedia to establish associations between results.

In heterogeneous networks, PathSim [Sun et al., 2011] provides a distance measure between items — a target item and a candidate similar item. It takes into account the number of links between items, as well as the number of links self-leading to each item. The authors manually define a path to be followed to find similar items, giving a semantic meaning to the relation between them. Then they use PathSim between the target item and the candidate similar items to find which ones are the top-k most similar items.

**Example 3.13** *Let us consider a cinematographic database with movies, directors and actors. Co-actors are linked by a path "actor–movie–actor." Movies directed by the same director are linked by a path "movie–director–movie." More complex paths include "actor–movie–director–movie–actor" — actors starring in movies of the same director — or "actor–movie–actor–movie–director–movie–actor–movie–actor" — actors whose co-actors have starred in movies directed by the same director.*◇

A few works in recommendation have considered the use of typicality to compute similarities and associations. Typicality can be based on frequency for instance.

**Example 3.14** *Let us consider authors publishing in journals. Authors who often publish in* Fuzzy Sets and Systems *are typically associated with this journal. As such, a similarity measure can be defined based on this criterion: similar authors have similar sets of typical journals in which they publish.*◇

In [Pivert et al., 2013] the authors introduce an approach based on fuzzy associations to provide users with additional answers to their database queries. The objective is to give users — in addition to the direct answer to their query — items that would be of

interest to them. To do so, the authors look for items similar to those directly returned, by leveraging the associations between the items in the direct answer and the other items in the database. The use of fuzzy logic along with foreign keys makes it possible to discover and rank items similar to those from the direct answer, notably with the definition of a typicality-based similarity measure. For instance, in a cinematographic database, actors would be considered similar to a given actor should they have:

- Played mainly for the same directors as the given actor;
- Played mainly in movies of the same genres as the given actor;
- A set of most frequent co-actors which is similar to the given actor's.

From then on, the principle is to compute the multisets of the comparison items (directors, genres, or co-actors) of the given actor, and those of other actors in the database. Then to compute the fuzzy sets of *typical* values for the multisets, before finally computing a degree of matching between the above fuzzy sets. This degree is then used to rank the results. This approach is categorized as "current-state" in the sense of [Stefanidis et al., 2009].

The use of typicality in collaborative filtering RS was first introduced in [Cai et al., 2010] and then extended in [Cai et al., 2014]. The authors propose to start by creating item groups, in which items are fuzzily affected to by a clustering method — meaning that items may belong to several groups to different degrees. Then for each item group a corresponding user group is created, and populated with users who liked the movies in these item groups. Users are more or less typical in user groups, depending on their appreciation of the movies in the associated item groups. The affectation of users to user groups is done by a fuzzy clustering algorithm, meaning that users belong to a group to a certain degree $\in [0, 1]$. Recommendations for a given user are computed based on the ratings of other users in the given user's neighborhood. A neighborhood selects users with close typicality degrees in the different user groups — aggregated and compared with classical distance measures.

#### 3.2.3.2 Recommendation

Recommender systems (RSs) help users by providing them with suggestions. Based on some knowledge about users — such as ratings, web histories, or social network data — these systems recommend items that may be of interest to them.

Recommender systems can be classified into several categories [Ricci et al., 2015], depending on the recommendation algorithms, the data type, or the origin of the data for instance.

**Content-Based (CB) Recommender Systems** CB RSs provide users with predictions based on the similarity between items. To do this, they create a profile and a vector of features to describe each item. The recommended items are those that are similar to the items liked by the user based on this feature vector. For example, it could be movies of the same genre, movies with the same actors, or books written by the same

author. These features may be numerical or categorical attributes, so different similarity measures are required, especially when dealing with keywords. An alternative to keywords is to use concepts to facilitate the comparison of attributes [Middleton et al., 2009]. Another is to extract data elsewhere from the Web [Semeraro et al., 2009]. CB RS have a few typical shortcomings, including the cold start (new user problem), as well as overspecialization (lack of diverse recommendations). However, once a new item has had its feature vector constructed, it is ready to be recommended, eliminating the new item problem.

**Collaborative Filtering (CF) Recommender Systems** CF RSs provide predictions based on the ratings of users. No additional data — whether about items or users — are required, which makes these approaches very popular. The utility matrix – number of items × number of users – describes the rating of each user for each item. It goes without saying that a given user cannot rate every item in a large-scale system so there are many unknown ratings to figure out to provide recommendations. The first challenge is to acquire ratings, either explicitly with a feedback form, or implicitly through actions such as browsing items. Some of these approaches consider neighborhoods: sets of users or items similar to the one for which a prediction is made. There are two kinds of similarity here: similarity between items and similarity between users (both based on ratings).

This leads to two subclasses of neighborhood-based CF RSs: user-based CF and item-based CF.

**User-Based CF RSs** User-based CF RSs consider the ratings of similar users (similar in the sense of their ratings) on the same item to predict scores. Considering the ratings (rating vector $r_x$) of a given user $x$, we look for the most similar users and are able to estimate the given user's ratings based on these similar users. There are several similarity measures that may be used in this case, which must be carefully selected depending on the context. The Jaccard index (Equation (3.1)) will not take into account the value of the rating, only the fact that there is or not a rating for this item by this user. So if two users $x$ and $y$ have rated the exact same movies, they will be completely similar (on the basis of their rating profiles), even if they have rated these movies in completely different ways. That is because their rating vectors $r_x$ and $r_y$ are compared based on the presence or absence of ratings for movies. This is interesting in cases where Boolean actions (clicking on an item or not for instance) are considered. The cosine similarity measure (Equation (3.2)) will consider that the missing ratings are somewhat bad and not exactly absent: zeroes will be used for the missing ratings to compute the similarity, inducing the idea that the user does not like the movie. A solution to this problem is to subtract the mean from the values to all ratings, so that the zeroes will then be considered more "neutral." The Pearson correlation coefficient (Equation (3.3)) takes into account this average rating $\overline{r_x}$ of the user $x$, so as to factor his/her behavior: some users give generally high marks and others generally low marks. This coefficient computes the similarity between two users based on the set of items $S_{xy}$ they have both rated.

$$s_{\text{Jaccard}}(x,y) = \frac{|r_x \bigcap r_y|}{|r_x \bigcup r_y|} \tag{3.1}$$

$$s_{\text{Cosine}}(x,y) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{\|r_x\| \cdot \|r_y\|} \tag{3.2}$$

$$s_{\text{Pearson}}(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}} \tag{3.3}$$

**Example 3.15** *Let us consider the two vectors* $r_x = [1,0,0,1,3]$ *and* $r_y = [1,0,2,2,0]$. *The rating value itself has no consequence on the Jaccard index and so we rewrite and obtain* $r'_x = [1,0,0,1,1]$ *and* $r'_y = [1,0,1,1,0]$. *By applying the Jaccard index to these rewritten vectors we get* $s_{Jaccard}(x,y) = 2/4 = 1/2$. *By using the Cosine similarity on the original vectors we get* $s_{Cosine}(x,y) = 3/(\sqrt{9} * \sqrt{11}) = 1/\sqrt{11}$.$\diamond$

Once the set of the $k$ most similar users has been computed, it is possible to recommend an item $i$ for the original user $x$ based on the ratings of the $N$ other users who rated it (among $k$), with Equations (3.4) or (3.5) for instance:

$$r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi} \tag{3.4}$$

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}} \tag{3.5}$$

where $r_{xi}$ is the prediction for user $x$ for item $i$, based on the similarity degree $s_{xy}$ between users $x$ and $y$, and the rating $r_{yi}$ of user $y$ for item $i$.

**Item-Based CF RSs** User-based CF RSs consider the ratings of similar items (similar in the sense of their obtained ratings) by the same user to predict scores. Considering the ratings (rating vector $r_i$) of a given item $i$ not yet rated by user $x$, we look for the most similar items (in terms of similar ratings) among those already rated by $x$ and are able to estimate the rating for item $i$ based on these similar items. It is possible to use the same similarity measures (between item rating vectors) and predictions which were applied in user-based CF.

The choice between user-based and item-based collaborative filtering should be made according to the cardinality of the utility matrix. A system with a low number of users and a high number of items will favor the user-based model. More generally, it would appear according to [Sarwar et al., 2001, Deshpande and Karypis, 2004] that the item-based model is more reliable than the user-based model mainly because item similarity is "easier" to compute as users tend to have multiple tastes that may vary with time.

To improve CF RSs, the rating behavior of the user is taken into account — whether or not the user usually rates items higher or lower than other people. CF RSs also have issues with cold start (both new user and new item problems).

**Demographic Recommender Systems** These systems use user-related data such as age, occupation, gender, or localization. Approaches based on demographic data are popular in marketing papers, but there has not been much research from the RS community on "pure" demographic RSs according to [Ricci et al., 2015]. They are often used to improve collaborative filtering methods by restricting the neighborhood based on the user's characteristics [Vozalis and Margaritis, 2007]. Demographics can also help with the cold start problem, by using stereotypes to recommend items to new users who have not rated any item yet.

**Knowledge-Based Recommender Systems** These systems consider the user's needs and preferences. They need user requirements as input, obtained by a fill-out form or through a dialog. These requirements are domain-dependent: an expert must enable the input of requirements for each system of a different domain. This differs from collaborative filtering in which only ratings are needed, regardless of the type of item considered (movies, books). Knowledge-based systems are divided into two groups: case-based systems and constraint-based systems.

**Case-Based Recommender Systems** These systems rely on similarity functions that estimate the adequacy between the user's needs and preferences (input data) and the recommendations (output data) [Lorenzi and Ricci, 2005]. Let us consider a system recommending restaurants and a user whose favorite restaurant $R$ is in Lyon. He/she is on a trip to Paris and would like to go to a restaurant similar to $R$. The objective is to compare Parisian restaurants to $R$ with similarity measures on the domain attributes (considering their average prices, their cooking styles, their number of stars in cook guides, and other such parameters from the domain describing restaurants). These recommenders use similarity measures in the same manner as some Query by Example systems do, such as [De Calmès et al., 2003].

**Constraint-Based Recommender Systems** These systems deal with constraints formulated by the user, and leverage those available from the item domain [Felfernig and Burke, 2008]. By specifying constraints on some attributes, the values on some other attributes may be restrained. If we consider again a restaurants recommender, in a constraint-based system our traveling user will provide explicit constraints such as "cuisine is French" and "location is Paris 6e or Paris 7e" and "number of stars is 2." The system will alert the user that under these constraints, the recommended restaurants are all very expensive (because there is a domain constraint which infers the price "very expensive" from the location and the number of stars), while providing the expected results (if there are any).

**Context-Aware Recommender Systems** With the Internet of Things, context elements such as time and localization are now also taken into account. Mapping applications may provide some information depending on the time of the day: traffic-related data in the morning, or close-by restaurants around lunchtime. A tourist application

[Ruotsalo et al., 2013] proposes to provide additional information on places the user visits based on his/her location.

**Hybrid Recommender Systems** Any combination of several of the above systems. Basic combinations simply aggregate scores from different approaches, while more complex systems fuse intermediary steps from different approaches.

**Recommendation in Databases** Recommenders are closely tied to (relational) databases to store items, users, ratings, profiles, along with other statistics. However these are dedicated systems that solely focus on the task of recommending, which appears as a black box to most users, considering the impossibility to efficiently interact with recommenders. Indeed, classical recommenders simply provide recommendations (*e.g.* to a given user) and are not designed to answer specific user-inputs. In order to make recommenders more accessible, several frameworks manipulating recommenders on top of databases have been put forward. They follow some general principles: recommendations can be either pre-computed or computed on-the-fly. Most systems use pre-computed results stored in a matrix: the similarity between items with content-based systems; the rating-based similarity between users in user-based collaborative systems, and the rating-based similarity between items in item-based collaborative systems. One of their general objectives is to enable users to actively *query* for recommendations (with a query language or through more or less user-friendly interfaces). Some of these frameworks are detailed below.

In relational databases, the framework FlexRecs [Koutrika et al., 2009] laid bases for recommendations on top of any database using several recommendation strategies, leading to the CourseRank system to pick classes in college. Separating the database and the RS allows users to customize recommendations according to their database, as opposed to many RS which function like a black box. However the recommendation methods proposed are limited to content-based and collaborative filtering.

To facilitate the use of recommender systems in databases, Adomocivius *et al.* introduced RQL [Adomavicius and Tuzhilin, 2001], proposing ideas for a recommendation query language for relational databases. They later extended RQL into REQUEST [Adomavicius et al., 2011]. For both of these systems, the objective is to enable users to compute customized recommendations as easily as submitting SQL queries. The authors propose a Recommendation Algebra to describe REQUEST. It aims at enabling recommendations that are more complex than typical collaborative filtering ones. It proposes to use other dimensions in addition to users and items, including contextual data (such as time, place, and company). The main issue with both systems is that the recommendation computation is planned for on-the-fly execution, which would lead to important processing times and impracticality of use.

In [Sarwat et al., 2013] the authors present RecDB, a recommender system to use on top of PostgreSQL. They introduce the SQL operator `CREATE RECOMMENDER` to configure which algorithm to use on which data (ratings, users, items). Content-based and collaborative filtering algorithms can be configured with RecDB. Once a recommender has

been configured, it is ready to provide recommendations with the `RECOMMEND` operator. Although this step is carried out offline, there is no indication on how long the recommender creation takes. The initialization step creates the data structures necessary to generate recommendations based on all the users, items, and ratings chosen.

### 3.2.4 Providing Explanations to Users

Ever since their commercial introduction in 1979, users have had trouble interacting with relational database management systems. The twofold querying process can easily rebuke users.

1. Formulating a query can become an issue in itself, whether because of the query language, or of the user's (limited) knowledge of the database.
2. The answer set obtained can be unsatisfactory if it is empty or plethoric.

To help the user understand the querying process, and the results obtained, some systems provide explanations. We propose to distinguish:

- Explaining database results, how a (set of) answer(s) is computed;
- Explaining recommendations, based on which algorithms and which elements they are computed;

#### 3.2.4.1 Explaining Database Results

In relational database systems, explaining the origin of a given tuple in a result relies on *provenance*. Several forms of provenance exist, that differ according to their computation and goal: why, how, and where [Cheney et al., 2007]. Why-provenance consists in finding all the source tuples that led to the production of a given tuple in a query result. How-provenance consists in explaining how source tuples have been associated (with join conditions for instance) to generate a given tuple in the query result. Where-provenance focuses on finding the locations in the database from which the query answers come. In this context locations refer to attributes: for each attribute value of a tuple from the query result, the objective is to find columns which have cells with the same values. The where-provenance of a tuple can be obtained with its why-provenance if its values come from unaltered columns (the attributes from the resulting relation are original attributes directly projected from the schema and not obtained by a combination of columns).

A notion of causality in databases is introduced in [Meliou et al., 2010], with similitudes with why-provenance. A cause is defined as a source tuple $t$ that generated another tuple $r$ in the result set of a query $q$, if the removal of $t$ from the database removes $r$ from the result set of $q$. $t$ is also called a counterfactual cause. One of the main differences is that causes have a degree of responsibility measuring to what extent a tuple $t$ is considered to be a cause. This degree takes into account the minimum number of tuples that need to be removed from the database so that the tuple $t$ becomes counterfactual (*i.e.* is removed from the result set of the query considered).

A formal approach based on *intervention* is presented in [Roy and Suciu, 2014]. It aims at explaining phenomena on trends in query results. In order to discover whether a tuple is essential in leading to the answer set, they remove it and process the query against the database once more to assess whether the output differs significantly. To illustrate their approach, the authors consider the number of academic-authored and industrial-authored papers at SIGMOD each year between 1991 and 2011. The first number increased steadily over the period observed. So did the second number until years 2000-2007, before significantly decreasing. An hypothesis is that one industrial group such as *Bell-Labs* significantly participated in this conference until the mid-2000, before declining. By removing publications from this industrial, the number of industrial-authored publications lowers to the point that the trend of industrial-authored publications follows that of academic-authored publications. The authors model *causal paths* between tables by using their primary key-foreign key relations. The explanations considered in this case take the form of conjunctions of predicates of attributes values. A similar mechanism — called *influence* — is used in SCORPION [Wu and Madden, 2013] to understand the origin of outliers in aggregate queries.

In case-based reasoning, finding explanations is a task close to feature selection in both statistics and machine learning. For instance, in [De Calmès et al., 2003] the authors look for explanations for the distributions of attribute values in a query result. To declare that a given attribute $A$ "explains" a peak, the $A$-values of the items in this peak must be different from the $A$-values of the items in the other peaks. In this case, the items of this peak do not have to have similar values, considering that they share the fact that they are all different from the other peaks. However the authors also propose to enforce that the attribute values in the peak one tries to explain must have similar values. However, the authors point out that the found explanations may not always be meaningful with sets containing values that are too different.

#### 3.2.4.2 Explaining Recommendations

Many recommender systems do not provide any information on how exactly each prediction is computed. The users know that they give away some information of theirs — ratings, demographics, tastes — but cannot fathom with which piece of information each prediction is returned. With recommender systems, explanations are often limited to *justifications*, such as a limited description of the technique that computed the recommendation, e.g. "*you were recommended this item because other users similar to you bought it.*" However it is possible to provide more than simple justifications by using the item descriptions in conjunction with the recommendation technique. For instance, content-based approaches compute similarity based on item profiles, thus enabling recommendations such as "*you were recommended this item because you seem to have an interest in this property.*" Item-based collaborative filtering techniques can produce recommendations along with some of the items that helped compute it. Providing explanations to users to enlighten them on why they get some of their recommendations would: 1) increase the trust of users in the system, and 2) improve the transparency of the system. More reasons to provide explanations include efficiency, effectiveness,

persuasiveness, satisfaction, and scrutability [Gedikli et al., 2014]. There have been several surveys on the computation of explanations and their effectiveness [Tintarev and Masthoff, 2012, Gedikli et al., 2014]. In association-based systems (Subsubsection 3.2.3.1), computing the associations between entities also leads to explanations as to why they are related.

## 3.3 Chapter Conclusion

As mentioned at the beginning of this chapter, cooperative answering aims at helping users express their intent and receive an adequate answer. Difficulties in interacting with the systems are manifold: they may come from not knowing the system itself and not be acquainted with it (or querying systems in general in the case of beginners), or from not understanding the schema of the database. Be it presuppositions or misconceptions, users cannot always grasp the entirety of the database content and its schema. Likewise, systems should not settle for direct answers when intensional answers may be preferred.

Cooperative systems call for adaptability, clarity, and transparency, and resonate with the now crucial issue of explainable and interpretable AI. To ensure and implement such characteristics, explanations are needed: explaining the "behaviors" of the system is the key to human-computer understanding.

### 3.3.1 Synthesis

Database querying may seem natural to experts, but it remains something few people are capable of doing. Diverse search paradigms have been proposed to simplify the navigation for users such as keyword search or faceted search. However these paradigms do not help uncertain or unknowing users to browse a database. The Query By Example paradigm offers users the possibility to simply evaluate examples and to provide results on the basis of these evaluations. Beyond the results themselves, the inferred user preferences are stated and explained to let the user know about them.

### 3.3.2 Objectives

We propose to guide users and their interactions with systems in the following situations:

- Browsing a set of answers too large to manually check all of them;
- Receiving recommendations;
- Evaluating examples to obtain similar results.

Systems only too rarely help the user understand why the answer set is empty or why there are too many results. In Chapter 4 we present an approach to help users understand their query results with terms from the natural language, as well as discover patterns characterizing the data elements retrieved by their query. We do not aim at

explaining individual results, but groups (clusters) of results. The explanations we propose (descriptions and characterizations) take the form of conjunctions of (disjunction of) linguistic labels.

In Chapter 5 we present recommendation techniques as well as their associated explanations enabling users to understand how these recommendations are formulated. These recommendations techniques use typicality to leverage the associations between entities and the demographics from users. Typicality also enables the formulation of explanations more detailed than justifications to let the user understand how the proposed recommendations are computed. Explanations include the typical associations that generated the recommendation, as well as the atypical associations that make the recommendation stand out among others. We distinguish our approaches from the one in [Cai et al., 2014] (Subsubsection 3.2.3.1) with our use of typicality. Our approach also uses typicality to find similar users, however we look for items typically associated with groups of people based on their demographic characteristics, and resort to associations between entities.

In Chapter 6 we present Fuzzy Query by Example, an approach simply asking users to evaluate examples so as to: elicit their preferences, return results derived from these preferences, and explain which user preferences were inferred. To infer user preferences, we look for common properties between positive examples on the one hand, and common properties between counter-examples on the other hand. These preferences are then used to generate a fuzzy query matching those, before returning its results to the user with the inferred preferences. We distinguish our approach from the other two evaluation-based Query By Example approaches mentioned in Subsection 3.2.1. Unlike the approach described in [De Calmès et al., 2003] we do not use similarity relations on the attribute domains to compute results. Also, we use the inferred preferences to compute a fuzzy query and present its results, unlike the approach in [Zadrozny et al., 2010] which resorts to an iterative evaluation of examples until the user is satisfied. In the associated chapter discussion, we draw the differences in greater detail on the following criteria: example selection, example evaluation and result computation.

# Chapter 4

# Explaining Query Answers

The general issue of providing answers with additional information is one of the aspects of the domain known as *cooperative query answering* [Gaasterland et al., 1992], a challenging research direction in the database domain. Several types of approaches have recently been proposed that share that general objective. Helping users explore databases is also a form of cooperative answering, along with handling failing queries [Koudas et al., 2006] — which can be dealt with by relaxing the selection conditions — or queries yielding a plethoric answer set — which on the contrary may need more conditions to filter the answers — or ranking the answers to return only the *top-k* ones.

In this chapter we focus on the issue of *plethoric answer sets*. The many-answer problem can be tackled in several ways, with some focusing on the query (strengthening or augmenting the query), and others on the results (by summarizing them, or ranking them with preferences, or diversifying them).

Query-oriented approaches regroup two directions: strengthening the query (furthermore restraining selection conditions), and augmenting the query (adding new selection conditions). Both directions have been tackled in the case of fuzzy queries. In [Bosc et al., 2008] the authors propose to strengthen queries by making selection conditions more drastic to reduce the size of the answer set. To this end they propose to alter the fuzzy sets associated with the selection conditions by reducing their cores (and thus lowering the space of values fully satisfying the fuzzy set membership functions). Solutions related to query augmentation have been proposed in [Bosc et al., 2010] and [Pivert and Smits, 2014]. In both approaches the authors propose to use additional predicates to augment the original query. The main difference resides in the selection of these predicates. In [Bosc et al., 2010] the predicates are selected by semantic correlation to the original query so as to remain as close as possible to the original scope of the query. In [Pivert and Smits, 2014] the predicates are selected based on correlations inferred from a repository of previously executed queries. These approaches have a twofold objective: the initial answer set size must be reduced and the scope of the original query must not be modified too much. In order to evaluate and determine the best predicates to add in both approaches, the authors resort to fuzzy cardinalities to predict the extent to which the answer set will be reduced.

Approaches focusing on the results include summarizing the results (*e.g.* with clustering or rewritings) and ranking the answers according to user preferences. Summarizing the results throught a rewriting based on a fuzzy vocabulary enables the user to control the scope of the query by selecting the desired level of granularity in the vocabulary [Ughetto et al., 2008]. The authors of [Liu and Jagadish, 2009] offer users the possibility to refine their results by presenting them with the most representative answers to their queries. However they do not provide any additional information regarding the formed clusters beyond the attributes used by the user. In addition to being faced with too many answers to queries, users sometimes have difficulties browsing datasets for other reasons. These include attribute values that may not be easy to apprehend, or that all attributes may not be available for querying. Some systems offer specific views such as [Singh et al., 2016] to filter the results with a clustering algorithm as illustrated by the following example.

**Example 4.1** *Let us consider a user interested in second-hand cars, represented by the relation* cars(price, make, model, bodyType, drivetrain, mileage, engineSize, year, *etc.*). *The user wishes for a relatively recent SUV car. The submitted query is "select * from cars where mileage between 10000 and 30000 and bodyType = 'SUV' which yields thousands of results. Using a clustering algorithm, it is possible to provide the user with representative elements of the results. Based on these representative elements, the user may choose to focus on one cluster of elements in particular, refine his/her query in the process and reduce the number of results to browse. This process may be reiterated until the number of results is small enough to browse manually. One such representative element could be (make = Chevrolet, price = [25K-30K], year = [2011-2012],* etc.*).*◇

The approach described in [Singh et al., 2016] enables users to grasp the underlying structure of some datasets and offers some expressivity by displaying the attribute values ranges for each cluster of answers. However the authors do not use terms from the natural language to describe the answers, and do not provide discriminating characterizations for each cluster to explain them. Also, they require the user to know exactly how many clusters should be obtained to apply the *k-means* algorithm.

Unike summaries — which aim to be concise and to reduce the quantity of results presented to the user — approaches based on diversifying the results focus on selecting which items should be returned first to the user. The basic top-$k$ approaches do not take into account the possible similarity between the top-$k$ results, only the score w.r.t. the user query. In order to avoid redundant results, top-$k$ approaches taking into account diversity have been put forward [Qin et al., 2012]. More generally, the diversification of results has gained a lot of attention in the information retrieval community. Several approaches have been proposed, tackling this issue in the presence of a taxonomy of information [Agrawal et al., 2009], or in the case of keyword search over structured databases [Demidova et al., 2010]. Some of these approaches such as [Wang et al., 2013] consider tools used by summary-based approaches mentioned above such as clustering.

Beyond the many-answer problem, another cooperative mechanism of interest is the *explaining of answers.*

In the approach presented in [Pivert and Prade, 2012], for instance, the suspect nature of some answers (involved in the violation of one or several functional dependencies) to a request is identified through auxiliary queries. This may be viewed as a form of cooperative answering where additional information (here, the suspect nature of an answer, possibly with a degree) is given to the user.

In [Meliou et al., 2010], the authors take advantage of the lineage of answers for finding causes for a query result and computing a degree of responsibility of a tuple with respect to an answer, as a basis for explaining unexpected answers to a query. The idea is that "tuples with high responsibility tend to be interesting explanations to query answers." Another example of explanation needs is when the set of answers obtained is clustered in clearly distinct subsets of similar or close answers. Then, it may be interesting for the user to know what meaningful differences exist between the tuples leading to the answers which could explain the discrepancy in the result.

**Example 4.2** *For instance, if one looks for possible prices for houses to let obeying some (possibly fuzzy) specifications, and that two clusters of prices are found (*e.g. *by looking at the distribution of prices), one may discover,* e.g.*, that this is due to two categories of houses having, or not, some additional valuable equipment such as a swimming pool.*

A case-based reasoning approach explored this direction [De Calmès et al., 2003], however the authors pointed out that the explanations computed may not be meaningful when the values describing the different sets differ too much. One of our objectives is to provide end users with a mechanism to understand the answer set and its structure. We desire to be able to describe the answer set according to the attributes specified by the user in his/her query, and at the same time be able to formulate explanations and possibly narrow it down according to unexpected criteria, such as attributes not present in the original query.

**In this chapter we propose to combine summaries of answers and explanations to help users understand their results as well as the underlying structure of these results.**

**Example 4.3** *Consider a user querying for second-hand cars of make "Peugeot" and model 407 on a dedicated website. The answer set yielded contains over 1000 results, which is far too much for the user to browse exhaustively. The attributes that interest this user most are price and mileage. A partitioning of the results based on these two attributes yields two subsets of results:*

- *one subset of* expensive *cars with a* low mileage*;*
- *one subset of* cheap *cars with a* high mileage.

*Furthermore, additional information — not explicitly requested by the user —* specific to each subset *is available: the first subset contains* recent *cars, while the second contains* older *cars.*◇

In the following we propose *ClusterXplain*: an approach that first uses a clustering algorithm to detect groups of answers (a group corresponds to elements that have similar values on the attributes from the projection clause of the query) — this is the description step, that makes use of a fuzzy vocabulary. Then we look for common properties between the elements of each cluster (which elements from other clusters do no possess) for the other attributes — this is the characterization step.

Our objectives include:

1. Robustness (providing explanations to most user queries to enable users to understand the characteristics shared by groups of answers);
2. Interpretability (the explanations produced must be easily understandable by an end-user, which we will achieve through the use of fuzzy partitions of the domains involved and the identification of the most informative explanations);
3. Automatization of the cluster detection process (which must not require users to understand clustering algorithms and their parameters).

In this chapter, we first outline our approach in Section 4.1, then we detail the three steps of *ClusterXplain*, namely detection (Section 4.2), description (Section 4.3), and characterization (Section 4.4). We present experimental results in Section 4.5, and discuss them as well as some close research directions in Section 4.6. Finally, Section 4.7 points out some limitations of the approach and outlines perspectives for future work.

## 4.1 General Principle

Let $R$ denote the relation concerned by the selection-projection query $Q$ (note that $R$ may be the result of a join operation on multiple relations). $\mathcal{A}$ being the set of attributes of $R$, let us denote by $\mathcal{A}_\pi$ the subset of $\mathcal{A}$ made of the attributes onto which $R$ is projected (i.e., the attributes of the resulting relation), by $\mathcal{A}_\sigma$ the subset of $\mathcal{A}$ concerned by the selection condition, and let us denote $\mathcal{A}_\omega = \mathcal{A} \backslash (\mathcal{A}_\pi \cup \mathcal{A}_\sigma)$.

**Example 4.4** *Consider the simple following query:* "select mileage, year from cars where price is low;" *over the schema car(price, mileage, year, consumption, color, model). There is only one selection condition — on the attribute price — thus we have* $\mathcal{A}_\sigma = \{price\}$. *Two attributes are projected, thus we have* $\mathcal{A}_\pi = \{mileage, year\}$. *The remaining set,* $\mathcal{A}_\omega = \{consumption, color, model\}$, *contains all the attributes of* $\mathcal{A}$ *with the exception of* price, mileage, *and* year.

Let us consider a set of clusters of answers, formed based on the attributes from $\mathcal{A}_\pi$ (with a clustering algorithm). The three main steps of the approach are:

1. **Detection** of the clusters: applying a clustering algorithm on the data projected (attributes from $\mathcal{A}_\pi$) from the query (Section 4.2);
2. **Description** of the clusters: projecting them on the vocabulary defined on the domains of the attributes from $\mathcal{A}_\pi$ (Section 4.3);

3. **Characterization** of each cluster in terms of the vocabulary defined on the domains of the attributes from $\mathcal{A}_\omega$ (Section 4.4).

Step 1 groups data elements into clusters so as to exhibit the inner structure of the relation $R$ on the projected attributes. We cluster these data elements as the projected attributes are those of interest in a query. Step 2 is about using a fuzzy vocabulary to describe each one of these clusters. Step 3 aims at providing one or several characterizations for each of these clusters. A characterization is considered as additional information as it concerns attributes that do not appear in the query. Descriptions and characterizations both appear in the form of a conjunction of modalities (*i.e.* fuzzy labels) from the vocabulary, the only difference being the origin of the attributes under consideration. The objective is to find properties that will permit to describe the clusters with the attributes used to produce them (from $\mathcal{A}_\pi$) and then characterize them with attributes not involved in the query (from $\mathcal{A}_\omega$). This corresponds to identifying patterns (which may or may not be a *cause*) between clusters, *e.g.* the presence of a swimming pool or not in Example 4.2, or the age of the car in Example 4.3. Beyond simply identifying patterns, we wish to identify properties shared by elements of a given cluster *and* that are not shared by the elements of other clusters: the identified properties must *characterize* the given cluster. Figure 4.1 graphically presents the three steps of *ClusterXplain*.

A characterization is related to the set of clusters built in step 1. It is made of a set of linguistic descriptions, one for each cluster. Let us denote by $\mathcal{C} = \{C_1, \ldots, C_n\}$ the set of clusters obtained.

**Definition 4.1** *A* (fuzzy) ***description*** $E_{C_i}$ *attached to a cluster* $C_i$ *is a conjunction of couples (attribute,* (fuzzy) *set of labels) of the form*

$$E_{C_i} = \{(A_j,\ F_{i,j}) \mid A_j \in \mathcal{A}_\pi \text{ and } F_{i,j} \text{ is a (fuzzy) set of linguistic labels from the partition of the domain of } A_j\}.$$

**Definition 4.2** *A* (fuzzy) ***candidate characterization*** $E_{C_i}$ *attached to a cluster* $C_i$ *is a conjunction of couples (attribute,* (fuzzy) *set of labels) of the form*

$$E_{C_i} = \{(A_j,\ F_{i,j}) \mid A_j \in \mathcal{A}_\omega \text{ and } F_{i,j} \text{ is a (fuzzy) set of linguistic labels from the partition of the domain of } A_j\}.$$

**Example 4.5 (Crisp Example)** *Let us consider a set of second-hand cars, which could be classified into two sets based on a clustering on the attributes* price *and* mileage *so that:*

- *Cluster 1 is described by: "price is expensive and mileage is small";*
  *One possible candidate characterization is: "year is recent".*
- *Cluster 2 is described by: "price is affordable and mileage is high";*
  *One possible candidate characterization is: "year is old".*$\diamond$

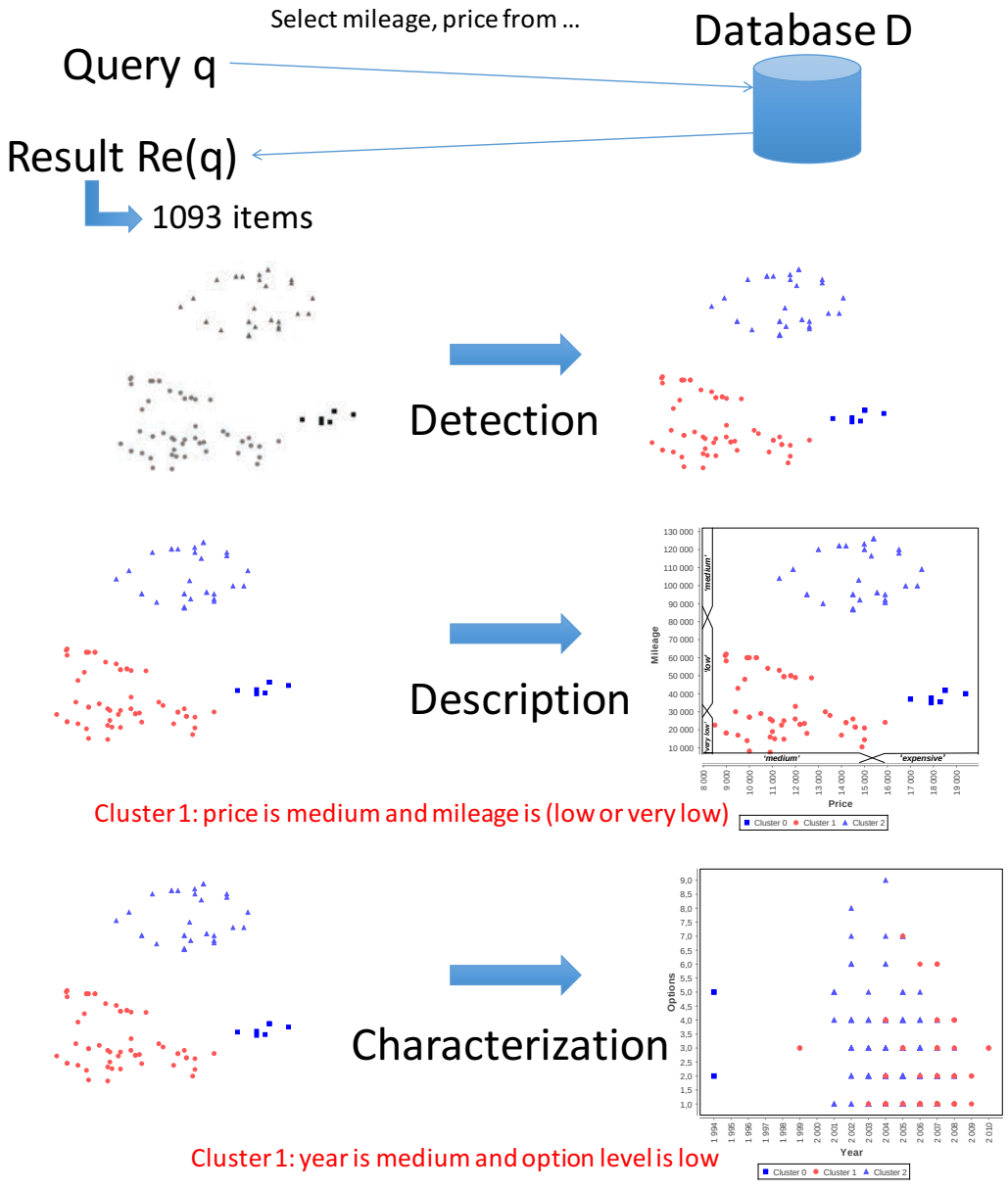



Figure 4.1 – ClusterXplain workflow

Table 4.1 – Correspondance between modalities and clusters: Example 4.6

| | *Price* | *Mileage* | *Year* | *Consumption* | ... |
|---|---|---|---|---|---|
| $C_1$ | {*0.7/expensive, 0.3/medium*} | *small* | {*0.8/recent, 0.2/medium*} | *medium* | ... |
| $C_2$ | {*0.7/low, 0.3/very low*} | *high* | {*0.6/old, 0.4/very old*} | *0.8/high, 0.2/medium* | ... |

**Example 4.6 (Fuzzy Example)** *Let us consider a query looking for the year and mileage of second-hand cars. Thus* $\mathcal{A}_\pi = \{price, mileage\}$ *and* $\mathcal{A}_\omega = \{year, consumption, make, ...\}$.

*The following fuzzy descriptions and characterizations may be obtained:*

- *Cluster 1 is described by:*
  *"(price is expensive (0.7) or medium (0.3)) and (mileage is small (1))";*
  *One possible candidate characterization is:*
  *"(consumption is medium (1)) and (year is recent (0.8) or medium (0.2))".*

- *Cluster 2 is described by:*
  *"(price is low (0.7) or very low (0.3)) and (mileage is high (1))";*
  *One possible candidate characterization is:*
  *"(consumption is high (0.8) or medium (0.2)) and (year is old (0.6) or very old (0.4))".*$\diamond$

**Remark 4.1** *In Example 4.6 only one candidate characterization is given for each cluster for the sake of simplicity. However let us note that every combination of attributes from* $\mathcal{A}_\omega$ *forms a candidate characterization, from single-attribute characterizations to the characterization containing every attribute from* $\mathcal{A}_\omega$.

## 4.2 Detecting Clusters of Answers

The first step is the detection of clusters. We briefly present and compare two families of clustering algorithms: *k-means* and *k-medoids*, before orienting and justifying our choice w.r.t. our objectives.

### 4.2.1 k-means vs. k-medoids

*k-means* is perhaps the most famous and overused clustering algorithm. *k-means* uses *centroids*, imaginary points to represent the centers of the clusters (computed with means, hence the name). *k-medoids* differentiates itself from *k-means* with the center of the clusters considered. *k-medoids* uses *medoids*, true points of the clusters designated as their "center." One direct consequence is that categorical attributes may be used with *k-medoids*. While is it possible to compute a mean value over numerical values, it is not possible to do so using categorical values. Both of these algorithms require a

$k$ parameter to partition a dataset into $k$ subsets. Finding the "right" $k$ is often the main problem with these clustering algorithms. The *k-medoids* algorithm (also known as Partitioning Around Medoids, or *PAM*) is presented in Algorithm 1.

```
Input: number of clusters k ; metric dist ;
Output: med list of medoids ; partition set of clusters
Global Variable(s): MAX_ITER
1  begin
2      Initialize the k medoids med;
3      medoidsChanged ← true;
4      while medoidsChanged and iter < MAX_ITER do
5          foreach element p do
6              find the medoid to which p is closest;
7              add p to this medoid's cluster;
8          end
9          medoidsChanged ← false;
10         foreach cluster do
11             check if there is a better medoid among its elements;
12             if there is a better one then
13                 update med;
14                 medoidsChanged ← true;
15             end
16         end
17     end
18 end
```

**Algorithm 1:** k-medoids (KMed)

Two variants, CLARA and CLARANS were designed for *larger* datasets. A fuzzy variant (and some of its optimizations) was introduced, the *Fuzzy c-medoids* (*FCMdd* or *fcmed* for short) in [Krishnapuram et al., 2001], considering that a point could now "more or less" belong to a cluster. Membership functions are used to compute whether a point belongs to one cluster or to another. Several parameters such as a fuzzification coefficient as well as a minimum membership degree are also required.

**Remark 4.2** *$k$ is the number of clusters in crisp algorithms whereas $C$ is the number of clusters in fuzzy algorithms.*

The cost of this algorithm is quite high (as high as *k-medoids* theoretically, but the computation of distance measures is far more expensive in a fuzzy context than in a Boolean one), so the authors [Krishnapuram et al., 2001] proposed some optimizations such as Linear FCMed, or *lfcmed*. The objective of this optimization is to reduce the number of medoid candidates during the medoid updating step, so instead of browsing every cluster element, this algorithm only browses the $c$ elements with the highest membership degree. While keeping track of these elements with the highest membership

degree increases the cost of the first part of the algorithm, not having to check every element of every cluster in the second part outweighs this drawback.

### 4.2.2 LFCMed-select

To optimize the clustering process, and offer more options, Lesot *et al.* proposed LFCMed-Select [Lesot and Revault d'Allonnes, 2012], with two major differences: (i) the possibility to over-estimate the number of clusters, no longer having to exactly specify the number of needed clusters, and (ii) the cluster selection step. After applying the LFCMed algorithm, selection criteria such as minimal cluster size (number of elements in a cluster) and cluster compactness (maximum cluster radius) are used to cut down the inadequate clusters. Of course, this leads to a partial clustering of the data, as the discarded data is not returned and no longer considered. The unassigned data can be added to the selected clusters, provided that they are close enough to one of the medoids.

The original algorithm stipulates that it should be applied to random subsets of data, to obtain several sets of clusters to be joined together with a hierarchical clustering process. This usually expensive procedure should be cheaper than usual as the many applications of the LFCMed-Select "heavily reduce the volume of data" [Lesot and Revault d'Allonnes, 2012].

All the above algorithms, except for the first one, are based on the fuzzy paradigm. In our case we decided to return to the Boolean paradigm and thus adapted LFCMed-Select into the LCMed-Select algorithm. Indeed, we believe that we do not need a fuzzy clustering algorithm because our objective is to describe and characterize crisp clusters of elements, to which data points fully belong. Adding membership degrees for the data points to the clusters would alter the obtained descriptions and characterizations. With crisp clusters, a data point contributes to the rewriting of a cluster w.r.t. the vocabulary once for each attribute. With fuzzy clusters, a data point will contribute to the rewriting of each of the clusters it belongs to w.r.t. the vocabulary for each attribute. The obtained descriptions and characterizations may not be as sensible with fuzzy clusters as with crisp clusters: our objectives include formulating clear and understandable explanations, which is hardly compatible with data points belonging to several clusters. From a computational point of view, the computation of the membership functions with fuzzy clustering algorithms results in a higher cost compared to crisp clustering algorithms.

The main advantages of this algorithm are:

- use of heterogeneous data — as with all algorithms of the *k-medoids* family;
- use of large data sets;
- overestimation of the number of clusters — no need for the exact number of clusters to obtain;
- the random cluster initialization effects are limited with the cluster selection step;

- no need for complex fuzzy membership functions — only needing crisp clusters reduces computation times.

In our experiments (Section 4.5), we used the distance measure (4.1) to compare numerical attributes, and identity for categorical ones.

$$dist(x,\, y) = \frac{|x - y|}{\max(x,\, y)} \tag{4.1}$$

Following the detection of clusters, the next step is their description.

## 4.3 Describing Clusters of Answers

The second step is the description of clusters. The clustering was applied on the attributes projected by the query, *i.e.* on $\mathcal{A}_\pi$. This resulting description uses terms from the natural language that are associated with the domain partitions of the attributes of $\mathcal{A}_\pi$. We first provide definitions of the fuzzy vocabulary, and then we present a crisp and a fuzzy version of the description step.

### 4.3.1 Fuzzy Vocabulary

In the approach we propose, it is assumed that the user specifies a vocabulary defined by means of fuzzy partitions, or that there exists a pre-defined vocabulary relevant to the applicative context considered. Let $R$ be a relation defined on a set $\mathcal{A}$ of $q$ categorical or numerical attributes $\{A_1, A_2, \ldots, A_q\}$. A fuzzy vocabulary on $R$ is defined by means of fuzzy partitions of the $q$ domains. A fuzzy partition $\mathcal{P}_i$ associated with the domain $\mathcal{D}_i$ of attribute $A_i$ is composed of $m_i$ fuzzy predicates $\{P_{i,1}, P_{i,2}, \ldots, P_{i,m_i}\}$, such that for all $x \in \mathcal{D}_i$:

$$\sum_{j=1}^{m_i} \mu_{P_{ij}}(x) = 1$$

where $\mu_{P_{ij}}(x)$ denotes the degree of membership of $x$ to the fuzzy set $P_{ij}$.

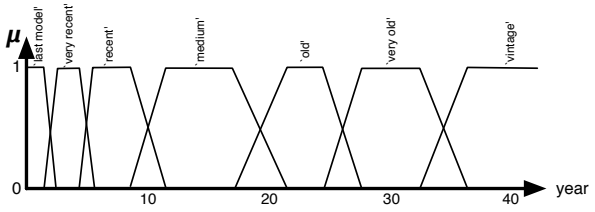



Figure 4.2 – A partition over the domain of the attribute *year*

More precisely, we consider Ruspini partitions [Ruspini, 1969] for numerical attributes (Fig. 4.2) composed of fuzzy sets, where a set, say $P_i$, can only overlap with its predecessor $P_{i-1}$ or/and its successor $P_{i+1}$ (when they exist). For categorical attributes,

we simply impose that for each value of the domain the sum of the satisfaction degrees on all elements of a partition is equal to 1. Each $\mathcal{P}_i$ is associated with a set of linguistic labels $\{L_1^i, L_2^i, \ldots, L_{m_i}^i\}$.

**Example 4.7** *As an example, let us consider a database containing ads about second-hand cars and a view named secondHandCars of schema (id, model, description, year, mileage, price, make, length, height, nbseats, consumption, acceleration, co2emission) as the result of a join-query over the database. A common sense partition and labelling of the domain of the attribute year is illustrated in Fig. 4.2. Table 4.2 shows a possible common sense partition and labelling of the domain of the categorical attribute make.*$\diamond$

Table 4.2 – A partition over the domain of the attribute *make*

| | *make* | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Dodge | Jeep | ... | Honda | ... | Nissan | Renault | Peugeot | Dacia | ... | ARO | Oltcit | ... | VW | Lamborghini | Skoda | ... |
| American | 1 | 1 | ... | 0 | ... | 0 | 0 | 0 | 0 | ... | 0 | 0 | ... | 0 | 0 | 0 | ... |
| Asian | 0 | 0 | ... | 1 | ... | 0.6 | 0 | 0 | 0 | ... | 0 | 0 | ... | 0 | 0 | 0 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| French | 0 | 0 | ... | 0 | ... | 0.4 | 1 | 1 | 0.4 | ... | 0 | 0 | ... | 0 | 0 | 0 | ... |
| East-European | 0 | 0 | ... | 0 | ... | 0 | 0 | 0 | 0.6 | ... | 1 | 1 | ... | 0 | 0 | 0 | ... |
| German | 0 | 0 | ... | 0 | ... | 0 | 0 | 0 | 0 | ... | 0 | 0 | ... | 1 | 0.5 | 0.6 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

### 4.3.2 Crisp Projection of Clusters on Vocabulary Partitions

Once the clusters are formed, they are projected onto the vocabulary in order to provide the user with a description of the answers in natural language. When a cluster satisfies several modalities for a given attribute, a simple way to project it is to return the disjunction of the associated labels. A cluster $c_i$ can be "boxed up" with $2 * p$ points $(x_{j,min}, x_{j,max})$, (one pair for each of the $p$ dimensions of the clustering) so that these points indicate which fuzzy labels the cluster satisfies (to a degree $> 0.5$, others are not considered significant) for the attribute $A_j$. For instance in Figure 4.3, cluster 2 satisfies labels 2 and 3 of attribute 1, so the disjunction of these two labels should be considered. As to cluster 1, it only satisfies label 1. Regarding attribute 2, cluster 2 satisfies label $b$ only and cluster 1 satisfies label $a$. Label $b$ is not satisfied by cluster 1 because its degree is below 0.5.

**Remark 4.3** *This projection does not reflect the representativity of each modality in clusters: if the borders $(x_{j,min}, x_{j,max})$ for attribute $A_j$ each fully satisfy two different labels, then the number of cluster points satisfying each of these two labels is not taken into account in the description.*

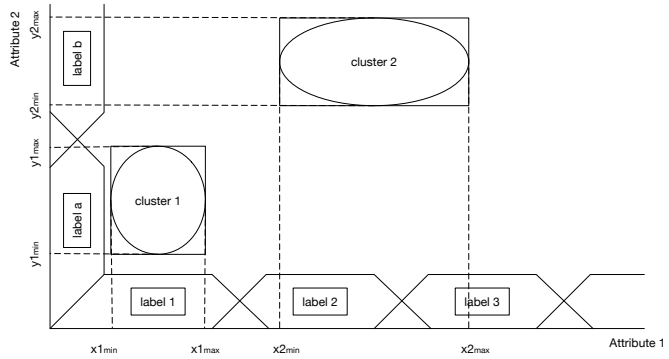



Figure 4.3 – Projection onto $\mathcal{A}_\pi$

### 4.3.3 Fuzzy Projection of Clusters on Vocabulary Partitions

The projection of $C_i$ onto the partition of an attribute $A_j \in \mathcal{A}_\pi$ is represented by a fuzzy set of labels $F_{i,j} = \{\mu_{L_k^j}(C_i)/L_k^j \mid L_k^j \in \mathcal{P}_j\}$ where

$$\mu_{L_k^j}(C_i) = \frac{\sum_{x \in C_i} \mu_{L_k^j}(x)}{|C_i|} \tag{4.2}$$

and $\mu_{L_k^j}(x)$ is the degree of membership of $x$ to $L_k^j$. It is assumed that the only labels that appear in $F_{i,j}$ are such that $\mu_{L_k^j}(C_i) > 0$. Note that the fuzzy set $F_{i,j}$ is not normalized in general, but this does not matter here. The degree associated with each label is related to the number of points verifying it and to their membership degrees, hence making descriptions representative of each cluster.

## 4.4 Characterizing Clusters of Answers

The final step is the characterization of clusters. We present two versions of the characterization process: a crisp one and a fuzzy one. Both types of characterizations have similar properties, namely specificity and minimality. Intuitively:

- Specificity aims at providing characterizations with attribute labels that characterize one cluster only;
- Minimality aims at providing characterizations as small as possible to avoid overwhelming the user with attribute labels.

We use these properties to rank candidate characterizations and determine which ones are actual characterizations.

**Definition 4.3** *A **characterization** is a candidate characterization that is both specific and minimal.*

In both cases, we present algorithms to determine which candidate characterizations are actual characterizations. We enumerate the possible candidate characterizations by increasing size: we start with one-attribute candidate characterizations, then with two-attribute candidate characterizations, etc. Testing candidate characterizations by increasing size enables checking the minimality property w.r.t. the characterizations already found immediately. The combinatorics regarding the number of possible characterizations is exponential, so we propose algorithms that leverage the minimality property to reduce the space of candidate characterizations.

### 4.4.1 Crisp Characterization

When considering attributes that were not involved in the clustering process, i.e., attributes from $\mathcal{A}_\omega$, the value distributions will in general be closer to the one in Figure 4.4, since the values of a given cluster are not linked by any similarity relationship anymore. In this case it is not very wise to "box up" one cluster altogether as we did above, because all adjacent labels may not be satisfied by a given cluster. For instance in Figure 4.4, cluster 2 satisfies labels 1 and 3, but not label 2. One must then check which labels each individual cluster element satisfies.

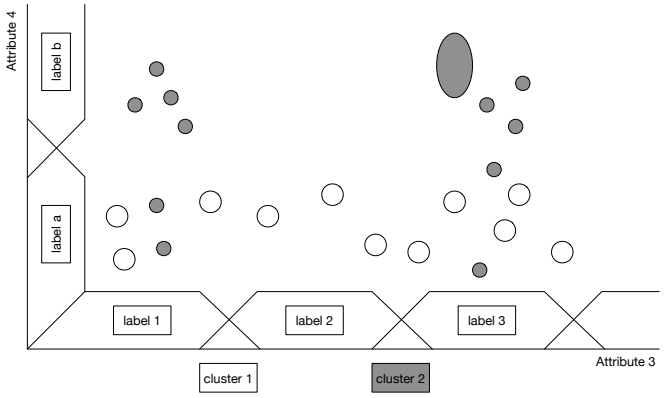



Figure 4.4 – Projection onto $\mathcal{A}_\omega$

The first step to discovering characterizations (in the sense of Definition 4.2) lies in filling a table associating each cluster with its projection on the attributes of $\mathcal{A}_\omega$. For every attribute $A_i$, $i \in [p+1,\, q]$ in $\mathcal{A}_\omega$, we indicate which modality $L_j^i$, $j \in [1,\, m_i]$ (or disjunction of modalities) is satisfied by each cluster. This results in Table 4.3, and an example of one such table was already given in Table 4.1 in Example 4.6.

#### 4.4.1.1 Crisp Properties

To be informative, a characterization should satisfy two properties: specificity and minimality, first defined in a crisp context.

**Property 4.1** *Specificity: a characterization must identify and characterize a single cluster,* i.e. *only the elements belonging to the concerned cluster.*

Table 4.3 – Correspondence between modalities and clusters

| | $A_{p+1}$ | $A_{p+2}$ | $\cdots$ | $A_q$ |
|---|---|---|---|---|
| $C_1$ | $(L_2^{p+1} \vee L_3^{p+1})$ | $L_3^{p+2}$ | $\cdots$ | $L_8^q$ |
| $C_2$ | $L_4^{p+1}$ | $L_9^{p+2}$ | $\cdots$ | $L_6^q$ |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| $C_k$ | $L_5^{p+1}$ | $(L_3^{p+2} \vee L_4^{p+2})$ | $\cdots$ | $L_8^q$ |

In other words, a characterization $E(c_i)$ should satisfy the following equivalence:

$$\forall x \in R, Q(x) \wedge E(c_i)(x) \Leftrightarrow x \in c_i, \tag{4.3}$$

*i.e.* every element $x$ belongs to a strengthening of $Q$ formed by the conjunction $Q \wedge E(c_i)$ iff $x$ belongs to the cluster $c_i$. This equivalence guarantees the specificity of $E(c_i)$. Therefore:

$$\nexists x' \in c_j, (j \neq i) \text{ such that } E(c_i)(x') \tag{4.4}$$

**Property 4.2** *Minimality: viewing a characterization as a conjunction of predicates (or disjunctions of predicates), then one says that $E(c_i)$ is a minimal characterization of the cluster $c_i$ iff $\nexists E'(c_i) \subset E(c_i)$ so that $E'(c_i)$ characterizes $c_i$.*

**Example 4.8** *For instance, consider houses to rent, and suppose we have identified a subset of answers whose characterization is $E$ = (price is expensive) $\wedge$ (swimming pool = yes) $\wedge$ (garden is big), there should not exist a characterization* e.g. *$E'$ = (price is expensive) $\wedge$ (swimming pool = yes) also characterizing this cluster only.*$\diamond$

#### 4.4.1.2 Crisp Algorithms

Based on the elements from Table 4.3, we use Algorithm 2 to find characterizations for the clusters. This algorithm both uses the number of clusters as well as the data from Table 4.3 as input. For each cluster $c_i$ (line 2) we look for characterizations of every "size" (line 4) — starting with a single couple (attribute, label), then with two of them, then three, ... — while checking that they are all indeed specific in the sense of Eq. (4.3) (the candidate characterizations are all minimal, as narrowed down in line 5) with Algorithm 3 (line 6). Those which are specific are then added to the set of characterizations of the cluster considered (line 7).

**Example 4.9** *Consider the data in Table 4.3. Starting with cluster $C_1$, the first couple (attribute, label) considered is $(A_{p+1}, (L_2^{p+1} \vee L_3^{p+1}))$ — the first pair from the table. The second one is $(A_{p+2}, L_3^{p+2})$, and the last candidate characterization of size 1 for $C_1$ is $(A_q, L_8^n)$. Then characterizations of size 2 are considered, such as $((A_{p+1}, (L_2^{p+1} \vee L_3^{p+1})) \wedge (A_{p+2}, L_3^{p+2}))$.*

**Remark 4.4** *In this crisp version, all unordered combinations of couples (attribute, pair of labels) are considered, leading to an exponential number of candidate characterizations. It is only by finding characterizations that the number of candidate characterizations decreases, since candidate characterizations must not be a superset of an actual characterization (such candidate characterizations are not minimal, thus their specificity is never checked).*

With Algorithm 3, we check whether a characterization (conjunction of conditions) is specific for a given cluster. To do this, the characterization is confronted to every other cluster (lines 8 and 9). For each conjunct of the characterization, the corresponding labels associated with the clusters and attributes from Table 4.3 are compared to it in order to check whether the two overlap or not (line 12) — two conjuncts are said to overlap if they are the same or if they have a disjunct in common. If, at the end of the loops, no overlapping has been found, then the characterization is in fact specific to the cluster.

**Input:** $n$ clusters $c$; $|\mathcal{A}_\omega|$ attributes/values for each cluster;
**Output:** a set of characterizations for each cluster

```
1  begin
2      foreach cluster C_i do
3          Charact(C_i) ← ∅;
4          for j ← 1 to |A_ω| do
5              for every possible characterization E of size j that is not a superset of
                any element of Charact(C_i) do
6                  if E is_specific(i,E) then
7                      Charact(C_i) ← Charact(C_i) ⋃ E
8                  end
9              end
10         end
11     end
12 end
```

**Algorithm 2:** Characterizations Finder

**Remark 4.5** *In the crisp approach described above, the definition of specificity (Formula 4.3) is very drastic. Indeed, for a characterization $E_(c)$ to be specific, its properties must be satisfied by c and c only: there must not exist a single data point from another cluster also satisfying this characterization. In case of real-world datasets where clusters are not clear-cut, the risk is high that no characterization may be found with this definition of specificity.*

To increase the robustness of our approach, we introduce more flexibility by considering a fuzzy version of the characterization.

**Input:** int $i$, condition $cc$
**Output:** bool *spec*
Let $cc$ be a conjunction of conditions, $cc_g$ the condition of $cc$ on attribute $A_g$
Let $T[i, j]$ be the condition describing cluster $C_i$ on attribute $j$ (in Table 4.3)
$n$ clusters $C$ ; $|\mathcal{A}_\omega|$ attributes/values for each cluster ;
function is_specific() returns *result*
**begin**
    $spec \leftarrow true$;
    $l \leftarrow 1$;
    **while** $l \leqslant n$ & *spec* **do**
        **if** $l \neq i$ **then**
            $h \leftarrow 1$;
            **while** $h \leqslant |\mathcal{A}_\omega|$ & *spec* **do**
                **if** $cc_h$ *overlaps with T[l,h]* **then**
                    $spec \leftarrow false$
                **end**
                $h \leftarrow h + 1$
            **end**
        **end**
        $l \leftarrow l + 1$
    **end**
    $result \leftarrow spec$
**end**

**Algorithm 3:** Specificity Checker

### 4.4.2 Fuzzy Characterization

As with the crisp characterization, the first step to discovering fuzzy characterizations (in the sense of Definition 1) consists of filling a table associating each cluster with its projection on the attributes of $\mathcal{A}_\omega$ (cf. Formula 4.2, considering this time that $A_j \in \mathcal{A}_\omega$). For every $A_j \in \mathcal{A}_\omega$, $j \in [1, |\mathcal{A}_\omega|]$, we indicate which modality $L_k^j$, $k \in [1, |\mathcal{P}_j|]$ (or fuzzy set of modalities) is satisfied by each cluster and to which degree $\mu_{L_k^j}(C_i)$.

#### 4.4.2.1 Fuzzy Properties

The properties defined for the crisp characterization in Subsubsection 4.4.1.1 are extended for the fuzzy characterization, so as to provide more gradual answers, *e.g.* the specificity property is no longer Boolean, and is represented by a degree instead, making the approach more robust.

**Property 4.3** *Specificity: the specificity degree $\mu_{spec}(E_C)$ determines how representative a characterization $E_C$ is for a given cluster $C$, and not so for the other clusters.*

Since the cluster projections are fuzzy sets of labels, the notion of specificity must itself be viewed as a gradual concept. Being specific for a cluster characterization $E$

means that there does not exist any other cluster with the same characterization, *i.e.* with fuzzy sets that are not disjoint from those of $E$ for every attribute. It is then necessary to define the extent to which two such fuzzy sets are disjoint. Let us first consider a characterization involving a single attribute. Let $E_1$ and $E_2$ be the respective projections of the clusters $C_1$ and $C_2$ onto an attribute $A_j$ of $\mathcal{A}_\omega$, whose associated fuzzy partition is denoted by $\mathcal{P}_j$. One may define:

$$\mu_{disjoint}(E_1,\, E_2) = 1 - \max_{L_k^j \in \mathcal{P}_j} \min(\mu_{L_k^j}(C_1),\, \mu_{L_k^j}(C_2)), \tag{4.5}$$

which corresponds to the fuzzy interpretation of the constraint $\nexists L \in \mathcal{P}_j$ such that both $C_1$ and $C_2$ are $L$. When several attributes – let us denote by $\mathcal{A}$ this set of attributes – are involved, two characterizations are globally disjoint if they are so on at least one attribute and we get:

$$\mu_{disjoint}(E_1,\, E_2) = \max_{A_j \in \mathcal{A}} (1 - \max_{L_k^j \in \mathcal{P}_j} \min(\mu_{L_k^j}(C_1),\, \mu_{L_k^j}(C_2))). \tag{4.6}$$

Finally, the specificity degree attached to a candidate characterization associated with a given cluster $C$ may be defined as:

$$\mu_{spec}(E_C) = \min_{C' \neq C} \mu_{disjoint}(E_C,\, E_{C'}), \tag{4.7}$$

where $E_{C'}$ denotes the projection of $C'$ onto the attributes present in $E_C$.

**Property 4.4** *Minimality: viewing a characterization as a conjunction of fuzzy sets of predicates, one says that $E_C$ is a minimal characterization of the cluster $C$ iff $\nexists E'_C \subset E_C$ so that $E'_C$ characterizes $C$ with a specificity degree equal or greater than that of $E_C$.*

Formally, we use the inclusion in the sense of Zadeh ($F_1 \subseteq F_2$ iff $\forall x \in U, \mu_{F_1}(x) \leq \mu_{F_2}(x)$ where $U$ denotes the universe on which fuzzy sets $F_1$ and $F_2$ are defined) and we get:

$$\begin{array}{c} E_C \text{ is minimal iff } \nexists E'_C \text{ such that } \forall A_j \in \mathcal{A}_\omega,\, E'_C[A_j] \subseteq E_C[A_j] \\ \text{and } \mu_{spec}(E'_C) \geq \mu_{spec}(E_C) \end{array} \tag{4.8}$$

where $E_C[A_j]$ denotes the fuzzy set related to attribute $A_j$ in $E_C$.

**Example 4.10** *If we consider houses to let, and suppose we have identified a subset of answers whose characterization is $E$ = (price is expensive (0.7) or very expensive (0.3)) $\wedge$ (garden is big (0.4) or very big (0.6)), there should not exist a characterization* e.g. *$E'$ = (price is expensive (0.7) or very expensive (0.3)) also characterizing this cluster only i.e. so that $\mu_{spec}(E') \geqslant \mu_{spec}(E)$.*$\diamond$

#### 4.4.2.2 Fuzzy Algorithms

Given the definition of specificity, a characterization involving every attribute from $\mathcal{A}_\omega$ will have the highest specificity degree possible, denoted *maxSpec*. (*Elements of proof:*

adding attributes to characterizations will add more terms to the aggregate $\max_{A_j \in \mathcal{A}}$, in Equation (4.6), thus potentially raising the specificity degree).

The first step of the characterization process is to determine for each cluster the maximal specificity degree *maxSpec* one may expect for its characterizations. Clusters whose maximal specificity degree is greater than a predefined threshold $\lambda$ are said to be fully characterizable. For the others, two strategies may be envisaged: to accept a less demanding specificity threshold, or to try to find specific characterizations on *subsets* (of points) of the clusters concerned. Hereafter, we investigate the second option and propose a solution based on the notion of cluster focusing. With this method, one expects to be able to generate specific enough characterizations of an interesting subset of a non fully characterizable cluster. Our goal being to characterize a set of items gathered particularly according to their closeness to each other, it appears obvious to focus on the most central points of the cluster concerned. It is nevertheless worth noticing that the central points of a cluster built on the attributes from $\mathcal{A}_\pi$ do not necessarily form a compact and characterizable set on the attributes from $\mathcal{A}_\omega$.

Thus, Algorithm 4 is applied on each cluster to determine its maximal specificity degree, and if necessary to determine the largest subset of central points for which a characterization of a high enough specificity degree may be found.

This focusing step is done with the *clusterFocus* function, which requires three parameters: the cluster $originalC_i$, a focusing step $\alpha$ and the number of focusing steps *focus-factor*. It returns a limited part of the cluster, $(100 - \alpha)\%$ of $originalC_i$. The new *maxSpec* value for this cluster is then computed (line 9). For this calculation, all clusters are considered in their entirety (whether some have already been focused or not) except for the current one.

**Remark 4.6** *The* clusterFocus *function may be altered so as to focus on the most* typical *elements of a cluster, instead of the most central ones. Considering typicality means taking into account the relation of an element with the other clusters — increasing the computation cost in the process — as opposed to only considering the medoid distance.*

If it is still not characterizable, this step can be repeated until the cluster is reduced to its medoid/centroid (line 6), always computing the new size of the cluster focusing based on the original cluster $C_i$ (line 8). In other words, clusters are automatically truncated to provide users with the best characterizations possible *i.e.* with a specificity degree higher than $\lambda$. When displaying characterizations, users will be informed whether or not said characterizations concern a full or a focused cluster.

Once the maximal specificity degree has been computed for each cluster, (either complete or truncated), Algorithm 5 is applied to determine for each cluster all the possible characterizations of a minimal size and a maximal specificity.

This algorithm takes as input the number of clusters, the *maxSpec* value for each of them computed with Algorithm 4 as well as the data from Table 4.1. For each cluster $C_i$ (line 2), we look for characterizations (line 5) composed first of a single fuzzy set of labels (for one attribute only), then with two of them, then three, etc., and check whether candidate characterizations are specific and minimal. If so, they are added to the set of characterizations (line 9).

**Input:** $n$ clusters $C$ ; $|\mathcal{A}_\omega|$ attributes/values for each cluster ; specificity threshold $\lambda$ ; focusing step $\alpha$ ;
**Output:** one *maxSpec* for each cluster ;

```
1  begin
2      foreach cluster C_i do
3          compute maxSpec;
4          focus-factor ← 0;
5          originalC_i ← C_i;
6          while maxSpec < λ ∧ |C_i| > 1 do
7              focus-factor ← focus-factor + 1;
8              C'_i ← clusterFocus(originalC_i, focus-factor, α);
9              compute maxSpec for C'_i;
10             C_i ← C'_i;
11         end
12     end
13     characterize each cluster (focusing) with Algorithm 5;
14 end
```

**Algorithm 4:** Cluster Characterizer

**Remark 4.7** *Some attributes from $\mathcal{A}_\sigma$ may also be added to $\mathcal{A}_\omega$: those concerned by inequality conditions ($<, \leqslant, \geqslant, >, \neq$) as results may have several satisfying values for these attributes, and participate in the characterization process.*

**Remark 4.8** *In Algorithms 4 and 5, specificity degrees are compared to two values: maxSpec and $\lambda$. On the one hand maxSpec is the maximum specificity degree that a characterization may have for a given cluster, and it may be computed with the characterization containing all attributes. One of its purposes is to prune candidate characterizations in Algorithm 5 once a characterization with a specificity degree maxSpec has been found. On the other hand $\lambda$ is a specificity threshold set to determine whether a candidate characterization is "acceptable" or not in terms of specificity. Its value is discussed in Subsection 4.5.3. Both maxSpec and $\lambda$ are used together to determine whether a given cluster will admit characterizations, triggering cluster focusing if that is not the case.*

**Example 4.11** *Let us illustrate the computation of characterizations by considering the three clusters $C_1$, $C_2$, and $C_3$ depicted in Figure 4.5, over the attributes X and Y. For the sake of simplicity, let us assume that $\mathcal{A}_\omega = \{X, Y\}$ (*i.e. *the clustering was processed over another set of attributes). The correspondance between the three clusters and the attribute modalities is given in Table 4.4.*

*Let us first apply Algorithm 4. For $C_1$, we start by computing its maxSpec value, obtained with the characterization composed of all the attributes of $\mathcal{A}_\omega$, namely $E_{C_1} =$*

**Input:** $n$ clusters $C$ ; $|\mathcal{A}_\omega|$ attributes/values for each cluster ; one $maxSpec$ for each cluster ; specificity threshold $\lambda$ ;
**Output:** a set of characterizations for each cluster ;

```
1  begin
2      foreach cluster C_i do
3          Charact(C_i) ← ∅;
4          if maxSpec ⩾ λ then
5              for j ← 1 to |A_ω| do
6                  for every characterization E of size j that is not a superset of any
                     element of Charact(C_i) of specificity maxSpec do
7                      if μ_spec(E) ⩾ λ then
8                          if E is minimal then
9                              Charact(C_i) ← Charact(C_i) ⋃ E;
10                         end
11                     end
12                 end
13             end
14         end
15     end
16 end
```

**Algorithm 5:** Characterizations Finder

Table 4.4 – Correspondance between modalities and clusters: Example 4.11

| | $X$ | $Y$ |
|---|---|---|
| $C_1$ | $x_1(1)$ | $y_1(1)$ |
| $C_2$ | $x_1(1)$ | $y_2(1)$ |
| $C_3$ | $x_2(1)$ | $y_1(0.4) \vee y_2(0.6)$ |

$\{(X, \langle x_1/1\rangle), (Y, \langle y_1/1\rangle)\}$. *We compute its specificity degree with Equation* (4.7)*:*

$$\begin{aligned}\mu_{spec}(E_{C_1}) &= \min(\mu_{disj}(E_{C_1}, E_{C_2}), \mu_{disj}(E_{C_1}, E_{C_3})) \\ &= \min(1, 1) \\ &= 1.\end{aligned}$$

*The maxSpec degree for* $C_1$ *is maximum (1), so there is at least one characterization with a specificity degree of 1 to find. Assuming* $\lambda = 0.5$*, we have* $maxSpec \geqslant \lambda$*, there is no cluster focusing step here.*

*By doing the same for clusters* $C_2$ *and* $C_3$ *we respectively obtain* $\mu_{spec}(E_{C_2}) = 1$ *and* $\mu_{spec}(E_{C_3}) = 1$*. All three clusters have at least one characterization with the maximum specificity degree 1. We apply Algorithm 5 to find them.*

*For* $C_1$ *we start with the candidate characterizations with only one attribute. As there are two attributes in* $\mathcal{A}_\omega$*, there are two such candidate characterizations.*

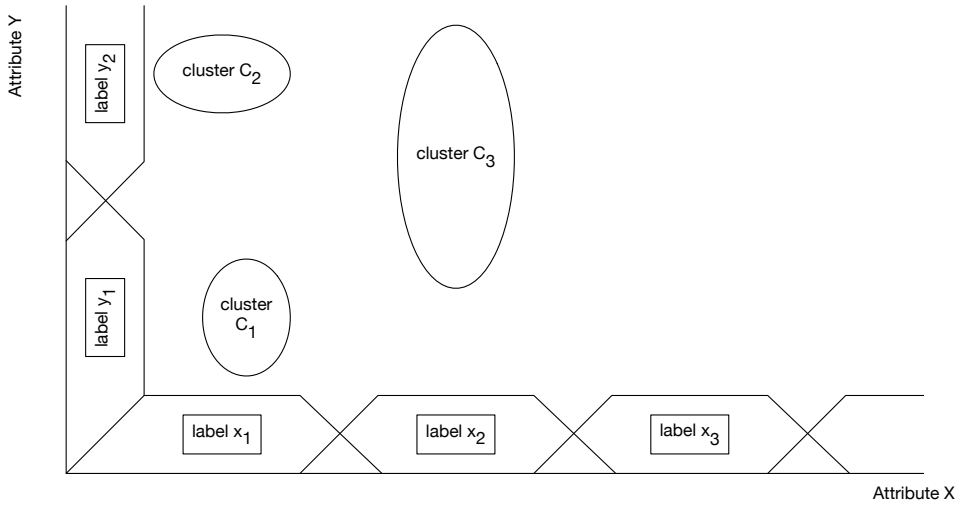



Figure 4.5 – Representation of three clusters on the attributes $X$ and $Y$: Example 4.11

Is $E_{C_1} = \{(X, \langle x_1/1\rangle)\}$ a characterization?

$$\begin{aligned}\mu_{spec}(E_{C_1}) &= \min(\mu_{disj}(E_{C_1}, E_{C_2}), \mu_{disj}(E_{C_1}, E_{C_3})) \\ &= \min(0, 1) \\ &= 0.\end{aligned}$$

*With a specificity degree lower than* $\lambda$, $E_{C_1} = \{(X, \langle x_1/1\rangle)\}$ *is not a characterization.*
Is $E_{C_1} = \{(Y, \langle x_1/1\rangle)\}$ a characterization?

$$\begin{aligned}\mu_{spec}(E_{C_1}) &= \min(\mu_{disj}(E_{C_1}, E_{C_2}), \mu_{disj}(E_{C_1}, E_{C_3})) \\ &= \min(1, 0.6) \\ &= 0.6.\end{aligned}$$

*With a specificity degree higher than* $\lambda$, $E_{C_1} = \{(Y, \langle y_1/1\rangle)\}$ *is a characterization. Out of the two one-attribute candidate characterizations, one is specific enough to be considered a characterization. The next step is to consider characterizations with more attributes. In this example the only remaining characterization is* $E_{C_1} = \{(X, \langle x_1/1\rangle), (Y, \langle y_1/1\rangle)\}$*, and we have already computed its specificity degree when computing* $maxSpec$*.*
Is $E_{C_1} = \{(X, \langle x_1/1\rangle), (Y, \langle y_1/1\rangle)\}$ a characterization? *Following Algorithm 5, we check whether* $E_{C_1}$ *is not a superset of any element of* $Charact(C_1)$ *with a specificity of* $maxSpec$*. While* $E_{C_1} = \{(X, \langle x_1/1\rangle), (Y, \langle y_1/1\rangle)\}$ is *a superset of the previously found characterization* $E_{C_1} = \{(Y, \langle y_1/1\rangle)\}$*, the latter does not have a specificity degree equal to* $maxSpec$*. The former is said to be* minimal*: even though it is the superset of a previously found characterization, its specificity degree is higher. As a result* $E_{C_1} = \{(X, \langle x_1/1\rangle), (Y, \langle y_1/1\rangle)\}$ *is added to* $Charac(C_1)$*.*$\diamond$

### 4.4.3 Improving the Characterization Format

The properties that characterizations possess have diverse uses in terms of understandability and explanation to the user:

- Specificity aims at providing characterizations with attribute labels that characterize one cluster only, and the specificity degree measures the extent to which that is so;

- Minimality aims at providing characterizations as small as possible to avoid overwhelming the user with attribute labels. It removes redundant labels that do not contribute to increasing the specificity degree.

To "minimize" explanations even more, we propose to leverage the vocabulary partitions so as to limit the size of overlong disjunctions of labels. To do so we suggest using *negative characterizations* that use labels not included in the original characterization. Let us consider a characterization over the attribute $A_j$, which is associated with a fuzzy partition $\mathcal{P}_j$ composed of $m_j$ predicates. We consider that a characterization for a given attribute $A_j$ is overlong if it is a disjunction of more than $m_j/2$ labels, *i.e.* more than half the number of modalities in the partition $\mathcal{P}_j$.

**Example 4.12** *Let us consider the characterization* price is very cheap (0.7) or cheap (0.25) or medium (0.05). *Its negative characterization is* price is not (expensive or very expensive). *It can be reformulated as* price is not expensive and not very expensive.◇

**Remark 4.9** *By using a negative characterization, we lose some information on the representativity of each modality for the considered cluster: in the above example the most representative label was* very cheap *with a degree of 0.5. With a negative characterization, there are no degrees attached to the labels to qualify how "representative" they are.*

To improve the understandability of disjunctions of labels, instead of displaying the membership degree of each label we can use linguistic quantifiers such as *few* or *most* to precise which label carries the most importance. Also, negligible labels (with a membership degree inferior to a given threshold *e.g.* 0.1) may be omitted from the characterizations.

**Example 4.13** *Let us consider the characterization* price is very cheap (0.7) or cheap (0.25) or medium (0.05). *The* medium *label has a degree of 0.05 and thus may be omitted as it is not particularly representative in the characterization for the attribute* price. *The characterization becomes* price is very cheap (0.7) or cheap (0.25). *By using linguistic quantifiers to translate the importance of the degrees, the characterization becomes* price is mostly very cheap or sometimes cheap.◇

## 4.5 Experiments

We present illustrative examples for both approaches and assess their performances depending on the numbers of tuples and attributes considered. We discuss these results after having presented both experiments.

### 4.5.1 Comparing Characterizations

As discussed in Section 4.2, we used the *l-cmed-select* algorithm, a crisp variant of the *l-fcmed-select* technique proposed in [Lesot and Revault d'Allonnes, 2012], which belongs to the framework of incremental clustering and combines relational clustering and medoid-based methods. To describe and characterize the data, we use an appropriate vocabulary that fits the domain attributes [Lesot et al., 2013].

#### 4.5.1.1 Crisp Illustrative Examples

In order to check the effectiveness of the approach, we performed a preliminary experimentation using a synthetic dataset with houses to let as in [De Calmès et al., 2003]. The attributes considered were *price*, *surface*, *garden area*, and *swimming pool*. The dataset was generated with the objective to obtain two distinguishable subgroups, hence a convenient distribution of the data.

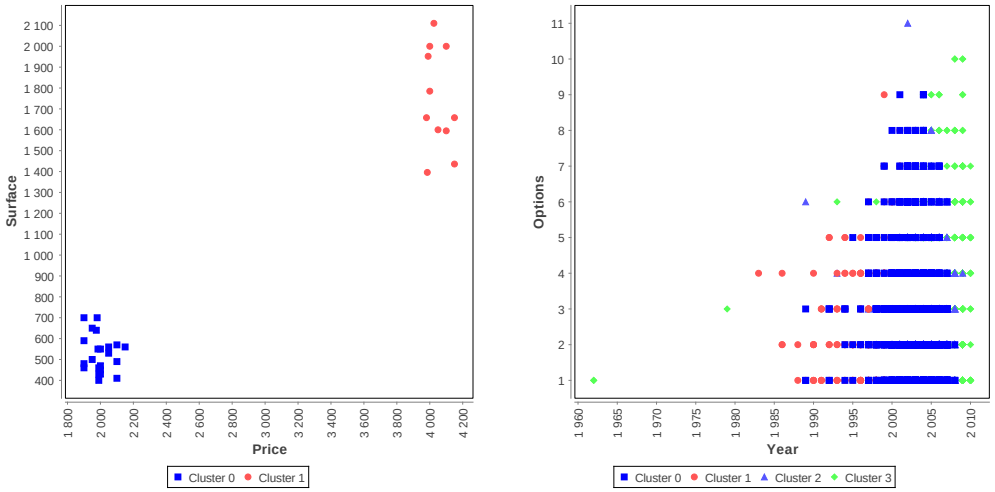
Consider that we are interested in querying the price and surface values of houses in a given city. The selection condition is on the attribute *city*, and the attributes in the projection are $\mathcal{A}_\pi = \{price, surface\}$. The remaining attributes are $\mathcal{A}_\omega = \{garden\text{-}area, swimming\text{-}pool\}$. The results of the clustering algorithm over the *price* and *surface* attributes are in Figure 4.6a. After processing Algorithm 2 on the data, several characterizations were found.

- Cluster 0 was described as: *price is cheap and surface is small*;
  The following characterizations were found:
  - *garden area is small*;
  - *swimming pool = no.*
- Cluster 1 was described as: *price is expensive and surface is big*;
  The following characterizations were found:
  - *garden area is large*;
  - *swimming pool = yes.*

Here, each cluster was associated with one label for each attribute. The first two attributes $\mathcal{A}_\pi = \{price, surface\}$ were the ones on which the clustering process was carried out, while the other two $\mathcal{A}_\omega = \{swimming\text{-}pool, garden\text{-}area\}$ each provided a characterization for each cluster, both specific and minimal.

With a real dataset, data is usually not as well-separated as in Figure 4.6a but closer to that of Figure 4.6b. Real data from second-hand cars ads were used here, with

attributes (*price*, *mileage*, *year*, *option level*, *security level*, *comfort level*, *brand*, *model*). Figure 4.6b is a representation of the data with the query looking for the prices and mileage of cars that cost between 25,000 and 30,000 € or below 10,000 €. In this case, some outliers are present and the border between clusters is not as clear-cut as in the former case, making it difficult (if not impossible) to find labels characterizing only one cluster. This leads us to using more flexible variants of the approach.




(a) Clusters of housing data over the attributes *price* and *surface*

(b) Clusters of second-hand cars over the attributes *year* and *option level*

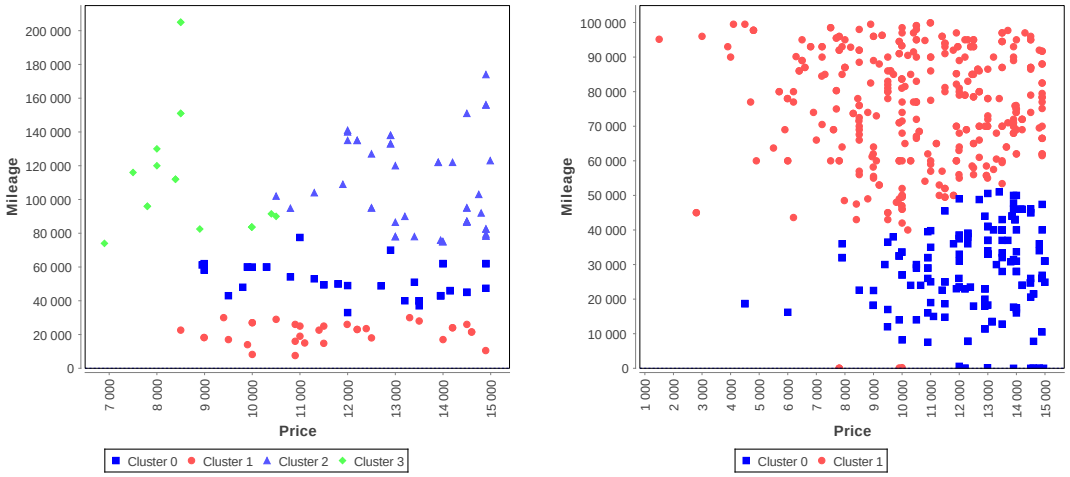
Figure 4.6 – Different clustering results

#### 4.5.1.2 Fuzzy Illustrative Examples

To test the fuzzy approach, we performed a preliminary experimentation with a real dataset of 700k second-hand cars ads extracted from LeBonCoin.fr. The attributes considered were *price*, *mileage*, *year*, *option level*, *consumption*, *horse power*, *brand* and *model*. The first two $\mathcal{A}_\pi = \{price, mileage\}$ were the ones according to which the groups of data were formed, while the others $\mathcal{A}_\omega = \{year, horse\text{-}power, ...\}$ were used to find characterizations for each cluster, both specific and minimal. Several examples are presented illustrating different situations.

Querying for the prices and mileage of cars of make 'Audi', from 2010 onwards and costing less than 15,000€ (Query 1), the clusters obtained on the result of that query are presented in Figure 4.7a. We empirically chose $\lambda = 0.7$ and got:

- Cluster 1: description: *(price is medium (0.69) or expensive (0.31)) and (mileage is very low (0.68) or low (0.32)*; characterization: specificity 0.83, *(year is recent (0.15) or very recent (0.85))*;
- Cluster 2: description: *(price is expensive (0.77) or medium (0.23)) and (mileage is medium (0.85) or high (0.15)*; characterization: specificity 0.71, *(option level is*




(a) Query 1 (b) Query 2

Figure 4.7 – Full clusters of second-hand cars over the attributes *price* and *mileage*

*high (0.70) or medium (0.13) or low (0.13)) and (consumption is high (0.76) or low (0.12) or medium (0.11)*;

- Cluster 3: description: *(price is medium (1)) and (mileage is medium (0.78) or high (0.22)*; characterization: specificity 0.75, *(year is recent (0.83) or very recent (0.17)) and (option level is medium (0.5) or low (0.28) or high (0.22))*;

but no characterizations for cluster 0. After a double focusing (62%), we got:

- Cluster 0 (62%): specificity 0.71, *(year is recent (0.87) or very recent (0.13)) and (consumption is low (0.33) or medium (0.33) or high (0.3)*.

We then considered cars of make 'BMW', 'Seat' or 'Volkswagen' costing less than 15,000€ with a mileage inferior to 100,000km (Query 2). The clusters are presented in Figure 4.7b.

- Cluster 0: description *(price is expensive (0.58) or medium (0.41)) and (mileage is low (0.62) or very low (0.38))*; characterization: specificity 0.74, *year is very recent (0.65) or recent (0.27)*;

- Cluster 1: description *(price is medium (0.64) or expensive (0.29)) and (mileage is medium (0.73) or low (0.26))*; characterization: specificity 0.74, *year is recent (0.63) or medium (0.3)*.

Two characterizations were found for the entire clusters, however since they were not very well separated, descriptions and characterizations have many labels in common, albeit with different degrees. Labels whose degree is inferior to 0.1 are omitted for the sake of readability, which explains why the sum of the description or characterization degrees is not always equal to 1.

#### 4.5.1.3 Discussion

The crisp approach cannot characterize clusters with mixed borders, unlike the fuzzy approach. Indeed, the fuzzy approach uses representative descriptions and characterizations of the clusters (with membership degrees for each label) so as to facilitate distinguishing clusters. Also, in the case of overlapping clusters, cluster focusing gives more chances for the characterization process to succeed. Nevertheless, there may not always be a characterization to find for each cluster.

**Remark 4.10** *Even in the extreme case where the cluster focusing step would reduce two clusters to their medoids, if they have identical labels for the attributes used in the characterization then they will not produce any specific characterizations.*

### 4.5.2 Performances

To assess the efficiency of the approach, we used a synthetic dataset with randomly-generated values on a Macbook Pro with a 3GHz Intel Core i7 processor and 16GB RAM. We checked the impact of two parameters on the processing times: the cardinality of the dataset and the number of attributes in $\mathcal{A}_\omega$. $|\mathcal{A}_\pi|$ was set to 3 for both experimentations. Let us note that the size of $\mathcal{A}_\pi$ does not influence the processing times for the characterization part: only the number of clusters does so. We compare the performances of both crisp and fuzzy approaches.

#### 4.5.2.1 Crisp Algorithm Performances

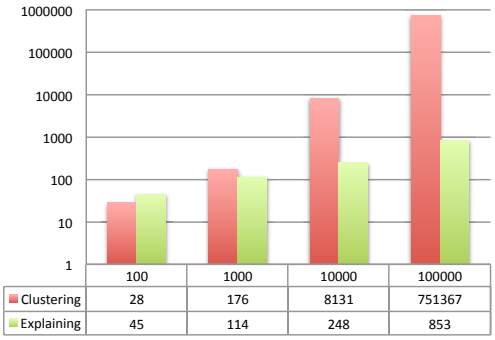
The results of the first experiment are presented in Figure 4.8a. $|\mathcal{A}_\omega|$ was set to 10. Processing times for the explanation process (description and characterization) are below one second. In the second experiment, we set the number of tuples to 10,000. The results, presented in Figure 4.8b, show that the processing time remains negligible as long as $|\mathcal{A}_\omega|$ is under 19 (which corresponds to a relation of a respectable arity).

#### 4.5.2.2 Fuzzy Algorithm Performances

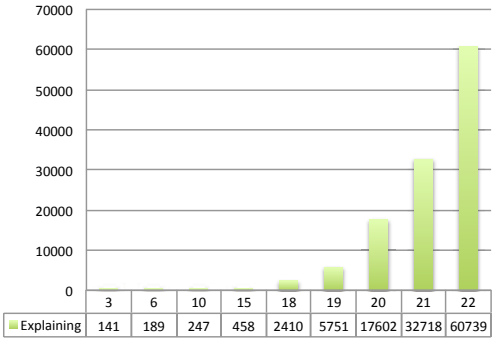
In the first experiment (Figure 4.9a), $|\mathcal{A}_\omega|$ was set to 10. Processing times for the explanation process (description and characterization) are below one second for answer sets of up to 10,000 tuples. The number of tuples raises the computation times of Table 4.1, which has to be updated for every focusing. However the rest of the characterization process is not impacted by the number of tuples considered. In the second experiment, we set the number of tuples to 10,000. The results (Figure 4.9b) show that the processing times remain low as long as $|\mathcal{A}_\omega|$ is under 15. The complexity of Algorithm 5 is exponential in the number of attributes $|\mathcal{A}_\omega|$, and follows the growth of $2^{|\mathcal{A}_\omega|}$.

#### 4.5.2.3 Discussion

In both approaches the clustering times are similar but not the same because two different sets of queries were used to compute them. By comparing Figures 4.8a and

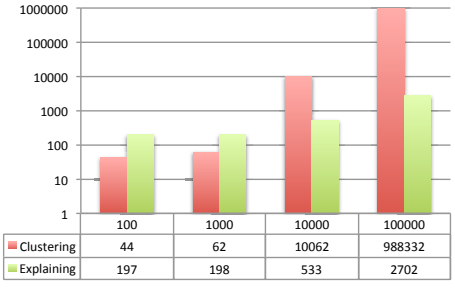



(a) Processing times (in ms, log scale) depending on the number of tuples processed for the clustering and explanation parts

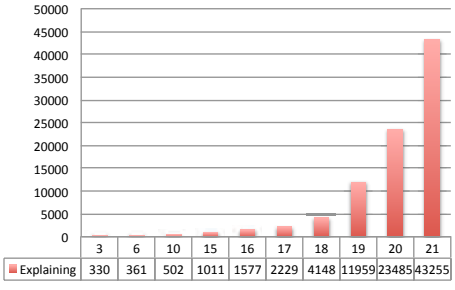



(b) Processing times (in ms) depending on the number of attributes in $\mathcal{A}_\omega$ for the explanation part

Figure 4.8 – Processing times in milliseconds (crisp approach).




(a) Overall processing time (in ms, log scale) depending on the number of tuples




(b) Processing time (in ms) depending on the number of attributes in $\mathcal{A}_\omega$ for the explanation part

Figure 4.9 – Processing times in milliseconds (fuzzy approach).

4.9a we can see that the explanation times are higher with the fuzzy approach regardless of the number of tuples in the answer set considered. Comparing Figures 4.8b and 4.9b also confirms this as for any considered number of attributes for the characterization part the explaining process is faster with the crisp approach than with the fuzzy approach. This higher cost of the fuzzy approach is induced by its added intermediary steps, such as cluster focusing, which requires that the table of correspondences between clusters and attributes be computed again — on which the number of elements has a direct impact. Also, the computation of the specificity degree in the fuzzy approach is longer than in the crisp approach: with the fuzzy approach we need to compute an exact degree while with the crisp approach only one overlapping condition need be found to obtain the non-specificity.

In both approaches the clustering part execution times are acceptable under 10,000 tuples of data. Let us emphasize that the clustering step is performed on a *set of answers*, not on a base relation, and one may consider that 10,000 already corresponds to a rather large answer set.

### 4.5.3 Specificity Threshold Values

The specificity threshold value $\lambda$ can be set between $0^+$ and 1. However, let us note that characterizations with a specificity degree below 0.5 are not *specific* in the sense that they are not representative of their cluster — because they are *more* representative of some other cluster. The minimal acceptable specificity threshold is then 0.5. The maximum specificity threshold value 1 is reminiscent of the crisp characterization approach: all elements of a cluster *must* be satisfied by this characterization. The higher the specificity threshold, the more difficult it gets to find characterizations, and the more chances there are that cluster focusing will be triggered. To limit the triggering of cluster focusing — and keep the clusters in their entirety for the characterization process — we propose to set the specificity threshold $\lambda$ to 0.5.

A low specificity threshold will result in more characterizations being found. This calls for the ranking of the obtained characterizations, which can be done with the specificity degree.

## 4.6 Discussion

In this section we position our work w.r.t. some approaches from other research directions including formal concept analysis, rough set theory, and data mining. We also discuss some limits and shortcomings of our approach.

### 4.6.1 Bridges with Formal Concept Analysis and Rough Sets

In [Farreny and Prade, 1984] the authors propose a method to designate objects so as to differentiate them from other objects according to a knowledge base. Their main focus is on providing *discriminating* designations, that are *specific* to a (set of) given

object(s). They define a *designation* as a class, possibly with adjectives and expressions of relations. They term a designation as *correct* "if it is strictly discriminating and it does only use properties and relations known or observable by the addressee." The authors favor finding "small" designations, suggesting that a shorter designation favors understandability. However they do not discuss the reasoning capabilities of the knowledge base, and focus only on the designation sentence generation.

Rough set theory [Pawlak, 1991] provides a framework to study sets of items which lack strict discriminating properties. A given set $X$ has a lower approximation and an upper approximation. Rules induced from the lower approximation are *certain* while rules induced from the upper approximation are *possible*. Elements with the same projection on vocabulary attributes in our (fuzzy) characterization approach are equally indiscernible. By using labels from the vocabulary to describe clusters of elements, we fulfill two objectives:

- We compare clusters based on their projection on attribute modalities, and thus remove computations over all elements when looking for characterizations;
- We formulate explanations with terms from the natural language that are understandable by users.

In formal concept analysis, a *formal context* can be viewed as a Boolean table representing the binary relation $R$ between a set of objects $\mathcal{O}$ and their sets of properties $\mathcal{P}$ [Dubois and Prade, 2016]. For each object $x \in \mathcal{O}$, $R(x)$ denotes the set of properties of $\mathcal{P}$ in $x$, and for each property $y \in \mathcal{P}$, $R^{-1}(y)$ denotes the set of objects of $\mathcal{O}$ having the property $y$. An operator $R^{\Delta}$ is defined, so that $R^{\Delta}(X)$ represents the set of properties shared by all elements in $X$. $R^{-1\Delta}$ is also defined, such that $R^{-1\Delta}(Y)$ represents the set of objects that share all properties of $Y$.

A *formal concept* is a pair $(\mathcal{X}, \mathcal{Y})$ where $\mathcal{X} \subseteq \mathcal{O}$ is a set of objects — the extension of the concept — and $\mathcal{Y} \subseteq \mathcal{P}$ is the set of properties that are shared by these objects — the intension of the concept [Gaume et al., 2013] — such that $R^{\Delta}(X) = Y$ and $R^{-1\Delta}(Y) = X$.

When considering bridges between formal concept analysis and our approach, $\mathcal{O}$ is akin to the content of the database yielded by the query of the user (the answer set), and $\mathcal{P}$ to the attribute labels. Assuming that all the elements of a cluster $C$ satisfy a given set of properties $D$, and that no other elements in the answer set satisfy all the properties of $D$, then $(C, D)$ can be viewed as a formal concept. Formal concepts are more restrictive than our approach: they correspond to finding a set of attribute labels that match *all* points from a given cluster. Ideally, we also aim for this objective, however in realistic situations not *all* points from a given cluster will fit. To this end we proposed to focus on the most central points of clusters, in order to consider graduality and to be able to find characterizations.

Another objective of ours is to find characterizations, which is similar to the task of finding *independent subcontexts*. By construction, clusters are independent sets of points — insofar as we consider crisp clustering. However their properties — the modalities

they satisfy — are not necessarily independent from other clusters. Finding such independent sub-contexts is akin to finding characterizations, while maintaining *some* flexibility. In a given subcontext, elements from a given cluster do not have to all satisfy a given set of properties as they would have to when considering a formal concept. However when considering independent subcontexts the property of specificity is central and must be upheld: not one element from a given cluster may possess a property from another cluster. In other words, finding independent subcontexts is the same as finding characterizations with a specificity degree equal to 1 (or simply finding characterizations with our crisp approach). As such, the drawbacks of our crisp approach (difficulty to find characterizations because of the mixed borders of clusters for instance) also apply to finding independent subcontexts.

### 4.6.2 Bridges with Data Mining Techniques

The first step of our approach is based on clustering, a classic data mining technique. We consider numerical and categorical attributes, each associated with a vocabulary. We rewrite each cluster with the (fuzzy) projection of its elements on the vocabulary partitions (and no longer manipulate the original values of its elements).

Item sets are at the heart of association rule mining. A one-item set is a set with one attribute value for one attribute. There are as many one-item sets as there are attribute values. Two-item sets contain two attributes values — one for each of two different attributes. Rule mining is done over the whole set of elements. In our approach, we look for characterizations (attribute sets: there are as many one-attribute sets as there are attributes, and not vocabulary modalities) for clusters (sets of elements). Unlike classic association rule mining, we do not review all items to look for characterizations but only the projection of the clusters. Furthermore, we are interested in finding discriminating rules: assuming that the description is the conclusion, and that candidate characterizations are the possible preconditions, then rules with the same preconditions for two different conclusions are eliminated.

### 4.6.3 Altering the Detection of Clusters

We wish to limit the scope of intervention of the user when it comes to complex parameters, such as the number of clusters. The clustering algorithm *lcmed-select* previously mentioned does not require the number of clusters. However it is fairly costly to run. Another clustering algorithm that does not require a precise number of clusters is DBSCAN: it requires a density instead. However the shape of the clusters produced with DBSCAN is different from those usually obtained with *k-means* or *k-medoids* algorithms: it groups together elements that are very close and that form a high density region. The shape obtained by these clusters may be more difficult to describe (by containing more disjunctions of labels) and to apprehend by users.

The result of clustering algorithms is altered by the choice of the similarity measure. It may be relevant to test more distance measures to evaluate their impact on the clustering results.

### 4.6.4 Altering the Characterization

The characterization process rests on two properties: minimality and specificity. The specificity degree is computed with several operators, including t-norms. If no characterizations are found it is possible to focus on the most central points of clusters and try again.

Several t-norms were mentioned in Table 2.1. Changing the t-(co)norm in Equations (4.5), (4.6), and (4.7) may also alter the characterization process entirely, including the cluster focusing optional step.

Characterizations have a conjunctive nature, conjunctively combining attributes and their (disjunctions of) labels. However it may be difficult to explain some clusters with a conjunctive normal form. There are cases where a disjunctive normal form may be more interesting. Consider two identifiable subsets in a cluster for instance: *(price is excessive and year is last model) or (price is very expensive and year is very recent)* corresponds to the two subsets, which as a whole would be conjunctively characterized by *(price is excessive or very expensive) and (year is last model or very recent)*, which is less informative than the disjunctive normal form. This would lead to generating characterizations more complex yet more precise.

## 4.7 Summary

In this chapter, we have presented an approach (in both its crisp and fuzzy versions) to characterize subsets of answers to database queries, using three steps: i) detection: the answers are grouped by means of a clustering algorithm; ii) description: the clusters obtained are described in terms of a fuzzy vocabulary; iii) characterization: other attributes (not involved in the clustering process) are used to highlight the particular properties of each cluster.

Experimental results show that the fuzzy approach is indeed effective in finding characterizations especially in cases where the crisp approach would fail because of its rigidity. While the fuzzy approach does not guarantee the robustness — there are not necessarily characterizations to find in each cluster — it does facilitate finding and ranking characterizations according to their specificity degree. Furthermore, the use of fuzzy sets to characterize clusters offers flexibility when dealing with clusters with mixed borders, and cluster focusing limits the impact of borderline elements.

Nevertheless *ClusterXplain* still requires a certain number of prerequisites before usage:

- A vocabulary adapted to the data, as covered in [Lesot et al., 2013, Smits et al., 2014c, Smits et al., 2017a], and that may be created by a domain expert or by end-users themselves with the help of tools such as ReqFlex [Smits et al., 2013];
- A given query formulated by the user (in the form of a conjunction of selection conditions for instance).

While the first prerequisite is made easier with tools enabling the creation of vocabulary partitions, the second prerequisite supposes that users can formulate queries. These

two prerequisites aside, *ClusterXplain* enables users to handle plethoric answer sets by dividing it into subsets that are then described with terms from the natural language. In addition *ClusterXplain* also enables users to understand the underlying structure of the data, by highlighting properties that are found in one subset and not in the others. Yet these prerequisites are fundamental to be able to formulate descriptions and characterizations. Using a (personalized) vocabulary contributes to the interpretability of these explanations, and to the transparency of the result.

Perspectives include conducting an extensive user study to assess the understandability of characterizations. We also plan further experiments over the specificity threshold value $\lambda$ with the objective to automatically adjust it to the quality of the clustering. Another perspective concerns the scalability of the approach which is good enough when considering answer sets limited in size. However as the volume of databases grows and the size of plethoric answer sets grows greater still this approach reaches some limits beyond 10,000 results. Let us precise that the description and characterization steps of *ClusterExplain* only so slightly take time in comparison to the clustering step. Improving the clustering process while continuing to *not* require the user to provide a desired number of clusters is yet another research perspective to consider: allowing users to select clustering algorithms of their choosing is possible, however this requires users to be knowledgeable about clustering algorithms and their parameters.

# Chapter 5

# Association-Based Recommendations and Explanations

E-commerce applications thrive on getting users to buy anything and everything. Displaying recommendations everywhere has now become the norm, by presenting users with items predicted as relevant or interesting to them. These recommended items have to satisfy a few properties, such as being new to the user (as opposed to recommending items too similar to those already owned by the user). Recommender systems do not require large amounts of data to provide users with suggestions: only a few rated items or some demographics are sufficient. Recommendations are computed by predicting a user's interest in items they have not used or bought yet. If the prediction score is high, then the item may be recommended. Predictions are ranked to fit the scale of the system, which could be for instance a 1-5 star scale. As detailed in subsection 3.2.3, recommender systems (RS) are divided into several categories including content-based systems (CB) and collaborative filtering systems (CF).

Recommender systems usually provide forward suggestions to users, without specifying based on which pieces of data nor how they were computed. Such obscure behavior leads to situations in which users may express varying degrees of annoyance such as:

- Perplexity: why is this recommended?
- Dissatisfaction: I do not like this.
- Worry: why does this system know me so well?

Facing these situations, a possible solution would consist in an explanation justifying the recommendation computed. This is a first step toward opening the black box of recommendation systems, which most of the time are seen as opaque. The computation of recommendations is not always straightforward and some methods cannot (currently) contribute to providing hints as to why their recommendations are put forward. On the other hand, there are several recommendation methods for which it is intuitive to

formulate explanations. In the following we will consider methods based on the notion of typicality, that permit us to consider recommendations that are effective for groups of people, or that exhibit typical associations with previously liked items. Typicality enables grouping together users or items that are linked with associations (and distinguishing them from other groups), whereas more classical recommendation approaches are based on the similarity between property or rating values. Recommendations based on typicality will enable us to formulate interpretable explanations based on these associations, so that users may understand why they obtain these recommendations.

The use of typicality in CF RS has already been investigated in [Cai et al., 2014]. The authors suggest creating item groups, in which items — in their case, movies — are fuzzily affected to by a clustering method — meaning that items may belong to several groups, to different degrees. Then, for each item group a corresponding user group is created, and populated with users who liked the items in these item groups. Users are more or less typical in user groups, depending on their appreciation of the items in the associated item groups. The affectation of users to user groups is done by a fuzzy clustering algorithm, meaning that users belong to a group to a certain degree $\in [0, 1]$. Recommendations for a given user are computed based on the ratings of other users in the given user's neighborhood. A neighborhood selects users with close typicality degrees in the different user groups — aggregated and compared with classical distance measures.

**Our objectives in this Chapter consist in extending a previous typicality-based approach from [Pivert et al., 2013] in several directions, driven by the need for explanations.** The different recommendations we provide all propose explanations as to how they were computed, and specify w.r.t. which items or users they are put forward, contributing to the robustness of our explanations. The associations used to explain these recommendations contribute to the interpretability of the explanations, by describing the associations between elements.

**Example 5.1** *Here is an example of the approach presented in [Pivert et al., 2013]: let us consider a user interested in actors querying the database who wishes to find actors similar to one (or several) in the result. Similarity may be based on values or/and on relations: we shall consider similarity based on relations between entities. In this cinematographic context, "similar" may have different meanings, such as working with the same directors, starring with the same co-actors, or playing in movies of the same genres. An actor such as Tom Cruise will have a set of typical directors, a set of typical co-actors, a set of typical genres and so on. To compute the similarity between Tom Cruise and other actors, we compute the set of typical elements of the other actors in the database and compare them to those of Tom Cruise. For instance if* $T_{Tom\ Cruise}$ = *{0.4/Cameron Crowe, 0.3/Steven Spielberg, 0.2/Christopher McQuarrie, ...} is the set of typical directors with whom Tom Cruise is associated, then actors with a similar set of typical directors will be considered similar to Tom Cuise.*$\diamond$

We propose to extend this approach so as to:

1. Propose explanations justifying the recommendations provided;

2. Leverage demographic data and combine it with ratings;

Explanations and justifications enable users to understand how recommendations are computed and tend to improve the trust of the user in the system.

**Example 5.2** *Users who enjoyed the movies* Star Wars IV *and* Star Wars V *may be recommended the movie* Star Wars VI *because these movies have similar ratings, similar actors, similar audiences, etc.*◇

**Remark 5.1** *The word "similar" may imply a similarity based on values, as that is the case with classical recommender systems. In this chapter we will sometimes use this word as a shortcut for "similar based on the associations between items" through misuse of language to lighten explanations.*

Demographics are a valuable source of information and we believe that in many domains (e.g. movies, music, literature) they can be used to suggest items to users. Approaches based on demographic data are popular in marketing papers, but the RS community has not carried out much research on the topic of pure demographic RS according to [Ricci et al., 2015]. Demographics are often used to improve CF methods by restricting the neighborhood based on the user's characteristics [Vozalis and Margaritis, 2007]. Krulwich leveraged demographics in [Krulwich, 1997] with LifeStyle Finder to create clusters of people. His approach focuses on acquiring large amounts of personal data through dialog. Pazzani described the needs of CB, CF and demographic recommenders to provide users with good recommendations [Pazzani, 1999]. He notes that the effort made to obtain demographic data is reflected with the quality of this information, hinting at the large amounts of information used by [Krulwich, 1997]. In his experiment, the demographic recommendations do not perform as well as CB and CF recommendations. Demographics can also help with the cold start problem, by using stereotypes to recommend items to new users who have not rated any item yet. However in some contexts such as tourist attractions [Wang et al., 2012] demographic RS have provided mitigated results. Users are not always inclined to share their personal data, even though these can sometimes be deduced from their own ratings and the personal data of some other users as reported in [Weinsberg et al., 2012].

We propose to find out the typical properties attached to each movie and find similarities with other movies based on typical demographic properties. We consider two visions to leverage demographic data with collaborative filtering and typicality in a movie context:

- Compute for all movies the demographic multisets representative of the users who liked them, compare them all with a similarity measure and provide users with movies similar to the ones they liked based on this notion of similarity;

- Compute the sets of typical movies liked by users with the same profile elements as the current user.

In other words, recommendations may be computed based on the user's profile or the user's ratings. We treat the *new user* problem with their demographic data. These computations are based on a typicality-based notion of similarity, as opposed to the classical similarity measures usually considered with recommender systems.

**Example 5.3** *Using our demographic approach, let us consider a user who likes Terminator and Tron. Let us assume that these movies are usually liked by young men in college. This frequent association between movies and demographics can be measured with a typicality degree: young and male are two typical properties of the audience of these two movies. The recommendation process may leverage these typicality links to provide the user (who is not necessarily a young man) with other movies appreciated by young men.*$\diamond$

There are many diverse recommendation approaches that focus on item properties, user ratings, links between items, etc. Striving to create a hybrid recommender leveraging the good properties of different recommending approaches is a classical problem in the RS community. Most approaches recommending movies predict ratings. A simple hybrid recommender can be built by aggregating these predictions together. More complex hybrid systems consist in fusing and intertwining the different steps of different recommendation approaches.

This chapter is structured as follows: in Section 5.1 we recall uses of typicality in fuzzy set theory. In Section 5.2 we present our approach leveraging associations between items, and in Section 5.3 we present our approach leveraging demographic data. For both approaches we provide explanations to enable users to understand the recommendations, as detailed in Section 5.4. The results of our experiments are detailed in Section 5.5 and we will conclude in Section 5.6.

## 5.1 Typicality in Fuzzy Set Theory

The concept of typicality has been studied in the fields of both cognitive psychology [Osherson and Smith, 1997] and fuzzy logic [Yager, 1997, Lesot et al., 2008]. In this work we will take inspiration from the definition presented by Zadeh in [Zadeh, 1987]. Let us note however that any other definition of typicality may be chosen instead without compromising the principles of the approaches described in this chapter.

In [Zadeh, 1987] Zadeh defines $x_i$ as a typical element in a fuzzy set $F$ if and only if:

1. the membership degree of $x_i$ in $F$ is high,
2. and most elements in $F$ are similar to $x_i$.

In the case where $F$ is a classical set then the first condition becomes $x_i \in F$, and the second condition remains the same. For the second condition of typicality to be satisfied, there must exist a similarity relation $S$ over the considered domain of values. Whether one such similarity relation is available or not, starting from a multiset $E$, the objective is to obtain a fuzzy set $T$ such that $\forall x_i$, $\mu_T(x_i)$ expresses the extent to which $x_i$ is typical in $E$.

### 5.1.1 Typicality Based on Frequency and Similarity

When a similarity relation $S$ over the considered domain is available, we may use the definition proposed by Dubois and Prade [Dubois and Prade, 1993], which says, following Zadeh's interpretation [Zadeh, 1987], that an element $x_i$ is all the more typical in a multiset $E$ as it is both frequent in $E$ and similar to most of the values of $E$:

$$\mu_T(x_i) = \frac{1}{n} \sum_{j=1}^{n} \mu_S(x_i,\, x_j) \tag{5.1}$$

where $n$ is the cardinality of $E$, with

$$\mu_S(x_i,\, x_j) = \max(0,\, \min(1,\, \frac{\alpha + \beta - d_{ij}}{\beta})), \tag{5.2}$$

where $d_{ij}$ denotes the distance between $x_i$ and $x_j$ with respect to $S$, and the values $\alpha$ and $\beta$ with $\alpha \leq \beta$ are positive real numbers which define a threshold of "indistinguishability" around each value $x$.

**Example 5.4** *Let us consider the multiset (of cardinality $n = 30$):*

$$E = \langle 1/0,\, 1/3,\, 1/4,\, 4/5,\, 7/6,\, 5/7,\, 3/8,\, 5/9,\, 2/12,\, 1/23 \rangle$$

*where $k/x_i$ means that element $x_i$ has $k$ copies in $E$. With $\alpha = 2$ and $\beta = 2$, and $d_{ij} = |x_i - x_j|$, one gets the fuzzy set of typical values:*

$$T = \{0.05/0,\, 0.33/3,\, 0.52/4,\, 0.65/5,\, 0.77/6,\, 0.82/7,\, 0.73/8,\, 0.58/9,\\ 0.15/12,\, 0.03/23\}$$

*where $\mu/x_i$ means that element $x_i$ belongs to $T$ (i.e., is typical in $E$) to the degree $\mu$.*$\diamond$

It may appear desirable to express that the number 7 is the most typical element of the collection, to a very high degree. We may then use:

$$\mu_T(x_i) = \mu_{most}\left(\frac{1}{n} \sum_{j=1}^{n} \mu_S(x_i,\, x_j)\right). \tag{5.3}$$

where *most* is a fuzzy quantifier [Zadeh, 1983] whose general form is given in Figure 5.1. In order to get the desired behavior, we may use the values $\delta = 0.4$ and $\gamma = 0.8$.

**Example 5.5** *Let us come back to the data of Example 5.4. Using the quantifier most defined by $\delta = 0.4$ and $\gamma = 0.8$, we obtain:*

$$T = \{0.3/4,\, 0.62/5,\, 0.92/6,\, 1/7,\, 0.82/8,\, 0.45/9\}.\diamond$$

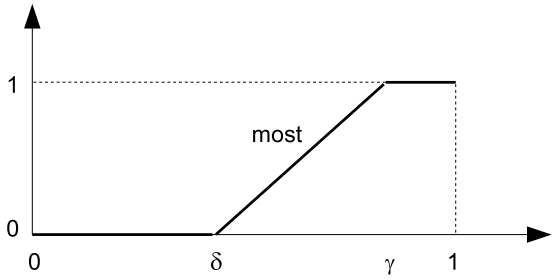



Figure 5.1 – A representation of the fuzzy quantifier *most*

### 5.1.2 Typicality Based on Strict Equality

Let us denote by $f_i$ the relative frequency of a value $x_i$ in a multiset $E$:

$$f_i = \frac{n_i}{n} \tag{5.4}$$

where $n_i$ is the number of copies of $x_i$ in $E$ and $n$ is the cardinality of $E$.

In the absence of any similarity measure, an obvious solution to compute the typicality of $x_i$ in $E$ is to take

$$\mu_T(x_i) = f_i. \tag{5.5}$$

Let us notice however that with this frequency-based approach, every element is considered somewhat typical. Those which have a low frequency get a low degree of typicality, but the elements which have a rather high frequency may also get a typicality degree significantly smaller than 1, since there are often several representative elements in a collection. Let us consider for instance a collection (multiset) of one hundred animals including thirty dogs, thirty cats, and various other animals with only one occurrence each. The element "dog" has the frequency value 0.3, as well as the element "cat". Now, it could appear desirable to express that "dog" and "cat" are the two typical elements of the collection, to a high degree. One may then use:

$$\mu_T(x_i) = \mu_{most}(f_i). \tag{5.6}$$

In order to get the desired behavior, we may use low values for $\delta$ and $\gamma$, for instance $\delta = 0.1$ and $\gamma = 0.5$ (which corresponds of course to a rather lax vision of *most*).

Now let us consider two actors $a_1$ and $a_2$, and suppose we wish to compute their similarity degree. We may consider several similarity criteria $c$ (based on co-actors, directors, genres) that will be aggregated. For both actors, we compute their multisets of elements w.r.t. each criterion $c$, such that:

$$E_c(a) = \{k/x \mid \text{element } x \text{ (linked to } a \text{ with } c) \text{ has } k \text{ occurrences}\}. \tag{5.7}$$

Based on each multiset we compute its fuzzy set of typical elements (or typical fuzzy set), using a frequency-based typicality measure $\mu_T$ (such as Formula (5.5)). With this typicality measure, we define the fuzzy set of typical values as:

$$T_c(a) = \{\mu_T(x)/x \mid \text{element } x \text{ has the typicality degree } \mu_T(x) \text{ in } E_c(a)\}. \tag{5.8}$$

**Example 5.6** *Let us consider the actor* Johnny Depp*, who has played 15 times for director* Tim Burton*, 9 times for* Gore Verbinski*, and 6 times for* Lasse Hallström*. His multiset of directors is:*

$$E_{directors}(Johnny\ Depp) = \{15/Tim\ Burton, 9/Gore\ Verbinski, 6/Lasse\ Hallström\}.$$

*Using a frequency-based typicality measure, his fuzzy set of typical directors is:*

$$\begin{aligned} T_{directors}(Johnny\ Depp) = \{&(15/(15+9+6))/Tim\ Burton,\\ &(9/(15+9+6))/Gore\ Verbinski,\\ &(6/(15+9+6))/Lasse\ Hallström\}\\ = \{&0.5/Tim\ Burton, 0.3/Gore\ Verbinski,\\ &0.2/Lasse\ Hallström\}.\diamond \end{aligned}$$

### 5.1.3 Comparing Fuzzy Sets of Typical Values

In our running example, our objective is to determine how close the two actors $a_1$ and $a_2$ are according to a similarity criterion. By computing their multisets of elements, we obtain a means to represent them. By computing the fuzzy sets of typical values of these multisets, we obtain a means to compare these actors. We propose to compute a matching degree between the multisets associated with $a_1$ and $a_2$, by using their fuzzy sets of typical values as a ground of computation.

Several interpretations of the condition $E_1$ *matches* $E_2$ — where $E_1$ and $E_2$ are two regular multisets of attribute values associated with different items — can be thought of. The problem comes down to assessing the equality of two fuzzy sets, and many measures have been proposed for doing so, see *e.g.* [Pappis and Karacapilidis, 1993, Bouchon-Meunier et al., 2010]. One may for instance:

- test the equality of the two fuzzy sets $T_1$ and $T_2$ of (more or less) typical elements in $E_1$ and $E_2$ respectively, for example by means of the Jaccard index:

$$\mu_{matches}(E_1,\ E_2) = \frac{\sum_{x\in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x\in U} \max(\mu_{T_1}(x),\ \mu_{T_2}(x))} \tag{5.9}$$

  where $U$ denotes the underlying domain of $E_1$ and $E_2$ — but this is rather drastic as in this case one does not expect to find *identical* fuzzy sets —, or by means of a measure such as:

$$\mu_{matches}(E_1,\ E_2) = \inf_{x\in U}\ 1 - |\mu_{T_1}(x) - \mu_{T_2}(x)|. \tag{5.10}$$

- check whether there exists at least one element which is typical both in $E_1$ and in $E_2$ (which corresponds to a rather lax view):

$$\mu_{matches}(E_1,\ E_2) = \sup_{x\in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x)). \tag{5.11}$$

- assess the extent to which most of the elements which are typical in $E_1$ are also typical in $E_2$ and reciprocally:

$$\mu_{matches}(E_1,\ E_2) = \min(\mu_{most\ \in T_2}(T_1),\ \mu_{most\ \in T_1}(T_2)). \tag{5.12}$$

The evaluation of Formula (5.12) is based on (one of) the interpretation(s) of fuzzy quantified statements of the form $Q\,X\,A\,\text{are}\,B$ where $A$ and $B$ are fuzzy predicates and $Q$ is a fuzzy quantifier (see [Zadeh, 1983, Yager, 1984, Yager, 1994]). The most commonly used interpretation was proposed by Zadeh [Zadeh, 1983] and is based on the ratio of elements which are $A$ and $B$ among those which are $A$:

$$\mu(Q\ X\ A\,\text{are}\,B) = \mu_Q\left(\frac{\sum_{x\in X} \top(\mu_A(x),\ \mu_B(x))}{\sum_{x\in X} \mu_A(x)}\right) \tag{5.13}$$

where $\top$ denotes a triangular norm, for instance the minimum. Then, Equation (5.12) rewrites (taking $\top = \min$):

$$\begin{aligned}\mu_{matches}(E_1,\ E_2) = \min(&\mu_{most}\left(\frac{\sum_{x\in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x\in X} \mu_{T_2}(x)}\right),\\ &\mu_{most}\left(\frac{\sum_{x\in X} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x\in X} \mu_{T_1}(x)}\right)).\end{aligned} \tag{5.14}$$

**Example 5.7** *Let us consider the cinematographic example introduced before and assume that two actors are considered similar if the typical sets of directors of the movies they star in are similar. Let us consider the typical sets:*

$$T(\textit{Johnny Depp}) = \{0.5/\textit{Tim Burton}, 0.3/\textit{Gore Verbinski}, 0.2/\textit{Lasse Hallström}\}$$

*and*

$$\begin{aligned}T(\textit{Helena Bonham-Carter}) = \{&0.45/\textit{Tim Burton}, 0.25/\textit{David Yates}, 0.2/\textit{James Ivory},\\ &0.1/\textit{Gore Verbinski}\}\end{aligned}$$

*The similarity degrees obtained using the previous measures are:*

- *with Formula (5.9):* $\mu_{matches}(E_{JD},\ E_{HBC}) = \frac{0.55}{1.45} \approx 0.38$
- *with Formula (5.10):* $\mu_{matches}(E_{JD},\ E_{HBC}) = \inf(0.95,\ 0.8,\ 0.8,\ 0.75,\ 0.8) = 0.75$
- *with Formula (5.11):* $\mu_{matches}(E_{JD},\ E_{HBC}) = \sup(0.45,\ 0.1,\ 0,\ 0.,\ 0,\ 0) = 0.45$
- *with Formula (5.14) using a quantifier "most" defined by* $\delta = 0.1$ *and* $\gamma = 0.5$*:*

$$\begin{aligned}\mu_{matches}(E_{JD},\ E_{HBC}) &= \min(\mu_{most}(\frac{0.45}{1}),\ \mu_{most}(\frac{0.45}{1}))\\ &= \min(\mu_{most}(0.45),\ \mu_{most}(0.45))\\ &= \min(0.87,\ 0.87)\\ &= 0.87.\diamond\end{aligned}$$

Example 5.7 shows that given two fuzzy sets of typical elements their matching degree varies greatly depending on the matching measure selected, ranging from 0.38 with Formula (5.9) to 0.87 with Formula (5.14).

Semantic proximity between values (if available) can also be taken into account during the computation of the similarity of two fuzzy sets. Such a matching measure, called *interchangeability*, is proposed in [Bosc and Pivert, 1997].

## 5.2 Association-Based Approach

In this section we describe the association-based approach presented in [Pivert et al., 2013] and extend some of its notions in a more global context. To explain in greater detail this approach, we resort to examples from the cinematographic domain, with data from the Internet Movie Database[1] that is structured according to the relational schema presented in Figure 5.2. This schema is composed of information tables (movies, characters, actors, directors, genres, etc) each of which have their own primary key, and of relation tables (mov_cast, mov_dir, mov_gen, etc) composed of foreign keys to link information tables together. The approach may be adapted to any given context, provided that some similarity criteria based on the notion of typicality may be defined. Our objective is to provide an extended answer to a query yielding a target actor $t$ in its direct answer. Given a similarity criterion, elements similar to $t$ will be returned along with $t$, making up the extended answer to the query.

We propose a method to compute extended answers — which may be viewed as recommendations — based on associations between entities. This approach is outlined with Algorithm 6. We implemented this approach into the **ReSO** (Retrieval of Similar Objects) prototype, and report experiment results in Section 5.5. We also applied the principle of this method to recommendation based on demographics, as covered in Section 5.3.

The first part of the algorithm consists in selecting the elements typically associated with the target object $c$ (from the direct answer of the query), and this must be performed for every criterion $i$ considered (cf subsection 5.2.1), $n$ being the number of criteria selected. From these elements, the multi-sets $E_i(c)$ associated with $c$ can be computed (Line 4) and then the fuzzy sets $T_i(c)$ can be computed from $E_i(c)$ with a membership function (Line 5). Then, for every potential similar item $x$ (Line 7), the same steps as above must be executed: computation of the multi-sets $E_i(x)$ for every criterion selected (Line 9) and then computation of the fuzzy sets $T_i(x)$ (Line 10). Following this step every $x$ can be compared to $c$ through the comparisons between the fuzzy sets $T_i(c)$ and $T_i(x)$ for every $i$ between 1 and $n$ (Line 11). A matching degree between the two fuzzy sets is obtained for every $i$, and the smallest one is kept as the matching degree between the two sets globally (Line 13). Finally if this degree $\mu$ is high enough — depending on the threshold $\alpha$ specified beforehand — then the element $x$ is deemed close enough to $c$ to be considered as a similar item and is added to the set $S(c)$ (Line 15).

[1] `http://www.imdb.com`

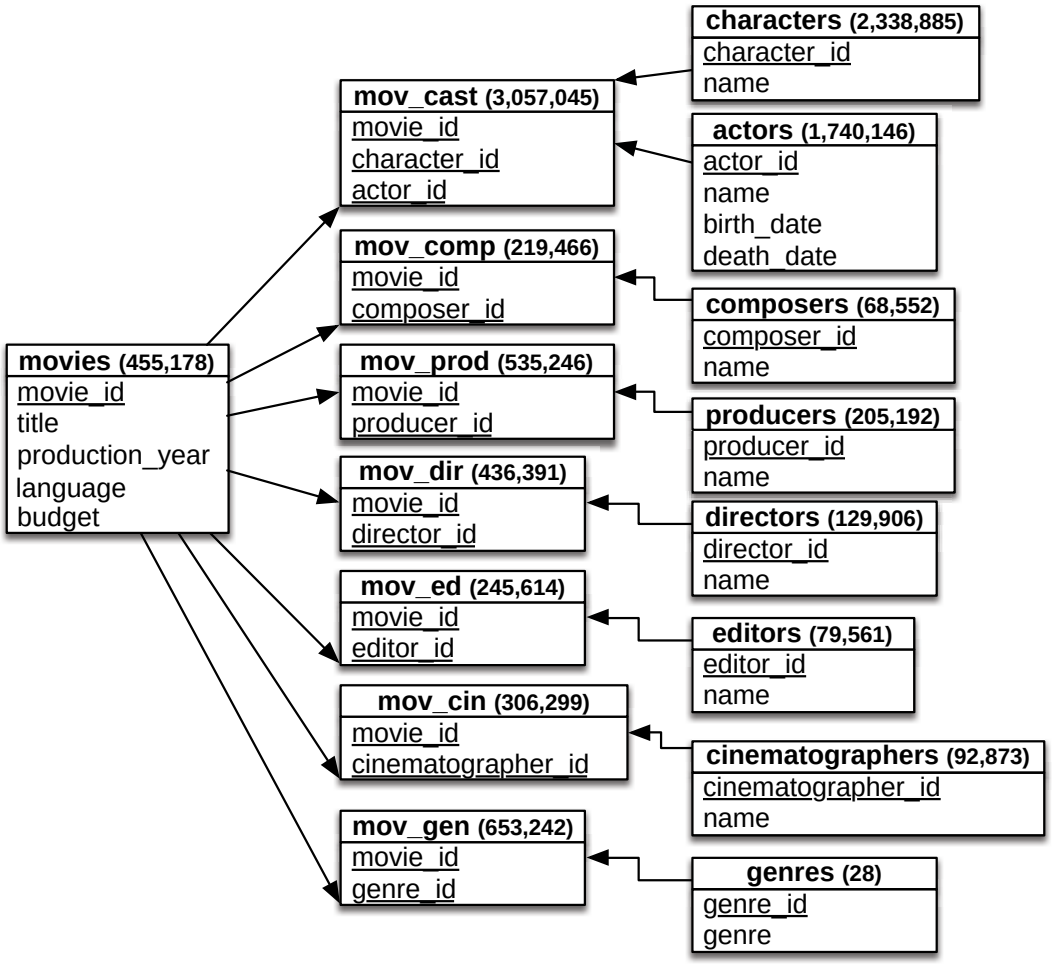



Figure 5.2 – Relational schema of the cinematographic database

**Input:** a target object $c$; $n$ specifications of multisets (i.e., $n$ subqueries); a threshold $\alpha \in (0, 1]$
**Output:** a fuzzy set $S(c)$ of objects similar to $c$

```
1  begin
2      S(c) ← ∅;
3      for i ← 1 to n do
4          compute E_i(c);
5          compute T_i(c) from E_i(c);
6      end
7      foreach item x in the relation concerned do
8          for i ← 1 to n do
9              compute E_i(x);
10             compute T_i(x) from E_i(x);
11             compute the degree of matching μ_i between T_i(c) and T_i(x);
12         end
13         μ ← min_{i=1..n} μ_i;
14         if μ ≥ α then
15             S(c) ← S(c) ∪ {μ/x}
16         end
17     end
18 end
```

**Algorithm 6:** Base ReSO algorithm

The computation of the matching degrees between $c$ and every element $x$ is rather time-consuming so it is essential to reduce the number of elements $x$ as much as possible, and to not consider every element in the relation concerned as stated in Algorithm 6. This leads to the use of filters (cf subsection 5.2.2) specific to every selected criterion $i$, to limit as much as possible the number of items to consider. Another (or complementary) alternative is to store the different matching degrees between elements so as to avoid recomputing all the fuzzy multisets and typical sets of elements.

### 5.2.1 Choice of Similarity Criteria

A similarity criterion is modeled as a set of primary-key–foreign-key relationships between tables that form a symmetric path between two instances of a given table.

**Example 5.8** *The relationship between actors that is based on the directors of the movies they played in is represented by the path*

$$actors - mov\_cast - \underline{mov\_dir - directors - mov\_dir} - mov\_cast - actors$$

*in the database schema, where the foreign-key relationship between tables* mov_cast *and* mov_directors *enables us to find the directors of the movies that each actor starred in.*$\diamond$

The central (set of) element(s) of the path is the considered similarity criterion. More complex paths can also be considered, although we limit this approach to simple paths that are interpretable: we also carry in mind the objective to explain these paths. Beyond the symmetric property, there is also a cardinality property that must be respected in order for typicality to have an interest in the approach. Indeed, repeated entity associations are required in order to label some relationships as typical.

**Remark 5.2** *This last sentence is true insofar as the notion of typicality we defined in subsection 5.1.2 is based on* frequency*. Another possibility to represent the notion of typicality in the cinematographic context would be to consider the* importance *of an actor in a movie,* e.g. *the more important a character is in a movie, the more typical his/her actor is to this movie.*

**Example 5.9** *A movie has a set of fixed actors, fixed directors, fixed genres. Each of these elements is unique and not repeated in a movie (aside from an actor playing several characters in a given movie). In other words, the path*

$$movies - mov\_cast - \underline{actor} - mov\_cast - movies$$

*cannot model a typical relationship between movies: it represents the link between movies that feature the same actors. Yet there is no notion of (frequency-based) typicality in the relationship between a movie and an actor starring in it, and as such it is not possible to define a set of typical actors for a given movie. Nevertheless the path*

$$movies - mov\_cast - \underline{actor - age - actor} - mov\_cast - movies$$

*can provide a similarity criterion between movies based on the age of the actors that starred in it (using age ranges for instance, we can determine whether two movies feature actors with the same typical ages, although one may question the relevance of one such criterion).*◇

In our cinematographic context, we consider three predefined similarity criteria for actors, and that actors are similar if they have:

- The same set of directors;
- The same set of co-actors;
- The same set of movie genres.

These criteria may be combined conjunctively — with the t-norm min for instance — or disjunctively — with the t-conorm max for instance — or simply with an arithmetic mean of their similarity degrees.

### 5.2.2 Filtering the Sets of Potentially Similar Items

To compute the extended answer of a query, *elements of interest* related to the direct answer must first be found. In order to determine which items should be recommended, a matching degree between the target element from the direct answer and *all* elements from the database is computed. However there is no need to actually compute the matching degrees for some elements (by first computing multisets and then fuzzy sets of typical elements) if we can already determine that these matching degrees will be very low (or equal to 0) and negligible compared to other elements.

Instead of computing all matching degrees we propose to introduce similarity filters in order to alleviate computation costs. These filters limit the number of candidate similar elements on the basis of some conditions specific to the similarity criterion. Two sorts of filters are considered:

- *obvious filters* that eliminate candidates that will yield a matching degree equal to 0;
- *restrictive filters* that strengthen the conditions to be considered "similar," and thus reduce the number of candidates.

The interest of these filters is to avoid the computation of multisets as well as fuzzy sets of typical values that will in no manner influence the extended answers presented to the user.

**Example 5.10** *Consider the co-actor similarity criterion. We propose to only consider candidate similar actors that have at least played twice with the target actor.*◇

**Example 5.11** *Consider the director similarity criterion. The obvious filter (to obtain a non-null matching degree) is to consider candidate similar actors that have starred in*

*at least one movie directed by one of the directors of the movies that the target actor starred in. For more restrictive filters, we can enforce that candidate similar actors must have starred in several movies directed by one (or several) director(s) of the target actor.*$\diamond$

The actual need for such filters is limited to real-time querying: all similarity degrees can be computed offline (and updated to reflect changes in the database). However in the case where storing these similarity degrees is not possible, then these filters greatly reduce the computation of extended answers.

## 5.3 Typicality-Based Approach Leveraging Demographic Data

In this section, we propose two approaches to compute recommendations leveraging demographic data, inspired by the typicality-based associations presented in Sections 5.1 and 5.2. We view demographics as statistical characteristics of human populations, including for instance age ranges such as [18-24] and [35-44]. A movie is said to be liked by a user if it was given 4 or 5 stars on a 5-star scale. In our movie context, we consider two visions to typicality-based collaborative filtering applied to demographic data.

- For each movie, link it to the typical sets of profession values, age values, and other characteristics of the people who liked it; then based on the movies liked by the current user, recommendations can be processed by comparing the fuzzy sets of typical values associated with these movies with those of other movies in the database. The current user's profile is not used in this approach, only the movies he/she liked;
- For each element (demographic feature) of the user's profile, a fuzzy set of typical movies liked by users with this profile element is computed. This time, the movies liked by the current user are not considered.

Each approach is first formalized and then illustrated with a detailed example. A hybrid approach leveraging the advantages of the two above is outlined at the end of this section. These approaches were implemented into the **TyDR** (Typicality and Demographics for Recommendations) prototype, and we report experiment results in Section 5.5. Let us denote by $C$ the set of demographic characteristics considered, by $M$ the set of movies, and by $U$ the set of users.

### 5.3.1 Using the User's Favorite Movies

This approach consists of several steps, the first two can be carried out offline, and the last one on-the-fly to provide recommendations to users. These steps are:

1. Compute for each item the sets of typical demographic features of the users who liked them;

2. Compare all items, based on these fuzzy sets of typical features;

3. Look for the items the most similar to those liked by the current user.

#### 5.3.1.1 Step 1: Computing the Fuzzy Sets of Typical Features

For each item $m \in M$ and for each demographic characteristic $c \in C$ (*e.g.* age, occupation, etc.) we compute the multi-set $E_c(m)$ representing the profile of the users who liked $m$ (we assume liking is the same as giving 4 or 5 stars on a 5-star rating scale), as in Subsection 5.1.2:

$$E_c(m) = \{(k/x) \mid x \text{ a modality of } c \text{ and } k \text{ the number of users having the characteristic } x \text{ and who liked } m\}.$$

Based on the multi-set $E_c(m)$, the fuzzy set $T_c(m)$ of typical values is computed:

$$T_c(m) = \{(\mu_T(x)/x) \mid x \text{ a modality of } c \text{ and } \mu_T(x) \text{ the frequency-based typicality of } x \text{ in } E_c(m)\}.$$

#### 5.3.1.2 Step 2: Computing Multisets

Several interpretations of the condition $E_1$ *matches* $E_2$ — where $E_1$ and $E_2$ are two regular multisets of attribute values associated with two different items — can be made. One may for instance test the equality of the two fuzzy sets $T_1$ and $T_2$ of (more or less) typical elements in $E_1$ and $E_2$ respectively, for example by means of the Jaccard index:

$$\mu_{matches}(E_1,\ E_2) = \frac{\sum_{x \in U} \min(\mu_{T_1}(x),\ \mu_{T_2}(x))}{\sum_{x \in U} \max(\mu_{T_1}(x),\ \mu_{T_2}(x))}, \tag{5.15}$$

where $U$ denotes the underlying domain of $E_1$ and $E_2$. Other comparison operators are proposed in Subsection 5.1.3.

A square matrix $S$ of size $|movies|$ is created, with each cell $s_{i,j}$ populated with the similarity degree between movies $m_i$ and $m_j$. This similarity degree is computed by aggregating the matching degrees between the typical fuzzy sets of all pairs of items. We use the Jaccard index as the matching measure, and the minimum as the aggregation operator.

$$s_{i,j} = \min_c(\mu_{matches}(E_c(m_i), E_c(m_j))) \tag{5.16}$$

**Remark 5.3** *Some features may be considered more important than others, and an aggregation operator such as the weighted average or the weighted minimum [Dubois and Prade, 1986] can then be used instead of the minimum.*

#### 5.3.1.3 Step 3: Browsing the Similarity Matrix

To predict the rating of a user $u$ for an item $m_i$, denoted by $p_{u,m_i}$ we search for the highest similarity value in the matrix $S$ between $m_i$ and the items that the user $u$ liked:

$$p_{u,m_i} = \max_j(s_{i,j} \in S \mid m_j \text{ liked by } u). \tag{5.17}$$

It must be noticed that instead of the maximum aggregation operator, others such as the weighted mean or the $k^{th}$ highest degree (which corresponds to a quantifier of the form "at least k" can be considered. The idea is to be as faithful to the demographic profiles as possible, leading to the utilization of less conventional aggregation operators than the max. An ordered list of recommendations is generated from $S$.

**Example 5.12** *Let us consider a user who liked the movies Star Wars IV and Tron.* ***Step 1*** *consists in computing the fuzzy sets of demographic data representing the users who liked these movies. As such, we obtain the following multisets for the* age *characteristic ([18-24] refers to people aged 18-24):*

$E_{age}$*(Star Wars IV) = {67/[-18], 378/[18-24], 813/[25-34], 442/[35-44], 162/[45-49], 147/[50-55], 87/[56+]}*

$E_{age}$*(Tron) = {9/[-18], 92/[18-24], 154/[25-34], 51/[35-44], 22/[45-49], 23/[50-55], 8/[56+]}*

*Based on these multisets, the associated fuzzy sets of typical demographic values are then computed:*

$T_{age}$*(SW IV) = {0.03/[-18], 0.18/[18-24], 0.39/[25-34], 0.21/[35-44], 0.08/[45-49], 0.07/[50-55], 0.04/[56+]}*

$T_{age}$*(Tron) = {0.03/[-18], 0.26/[18-24], 0.43/[25-34], 0.14/[35-44], 0.06/[45-49], 0.06/[50-55], 0.2/[56+]}*

*Let us recall that these fuzzy sets of typical values are computed offline for all movies in the database. Then for* ***Step 2*** *they are compared in order to compute the similarity matrix $S$. As such, we will compare Star Wars IV to all other movies in the database, such as Star Wars V. We will use the Jaccard index between their fuzzy sets of typical values, for all profile features considered. Then we will aggregate these matching degrees. Assuming that we have*

$T_{age}$*(SW V) = {0.03/[-18], 0.21/[18-24], 0.4/[25-34], 0.19/[35-44], 0.07/[45-49], 0.06/[50-55], 0.4/[56+]},*

*then we obtain* $\mu_{matches}(E_{age}(SW\ IV), E_{age}(SW\ V)) = 0.923$.

*For other characteristics such as* occupation *and* gender *we consider*
$\mu_{matches}(E_{occ}(SW\ IV), E_{occ}(SW\ V)) = 0.929$ *and*
$\mu_{matches}(E_{gender}(SW\ IV), E_{gender}(SW\ V)) = 0.925$.
*As a result,* $s_{SW\ IV,SW\ V} = \min(0.923, 0.929, 0.995) = 0.923$. *This process is repeated*

*for all unordered pairs of movies.*

*Once the similarity matrix S has been completed, we look for the most similar movies to those liked by the current user,* i.e. *movies with the highest degrees in S in the columns of the movies Star Wars IV and Tron.*$\diamond$

This approach is summarized in Algorithm 7. Lines 2 to 7 correspond to Step 1, lines 8 to 13 to Step 2, and lines 14 to 17 to Step 3.

**Input:** a target user $u \in U$ with characteristics $C$ ; a matrix of ratings $M \times U$
**Output:** a matrix of movies matching scores $S$, a set of predictions for user $u$

```
1  begin
2      foreach m ∈ M do
3          foreach c ∈ C do
4              compute E_c(m);
5              compute T_c(m) from E_c(m);
6          end
7      end
8      foreach pair of movies m_1, m_2 do
9          foreach c ∈ C do
10             compare T_c(m_1) and T_c(m_2);
11             s_{m_1,m_2} ← min_c(μ_match(E_c(m_1), E_c(m_2)));
12         end
13     end
14     foreach m ∈ M not rated by u do
15         compute p_{u,m};
16     end
17     return the top-k p_{u,m} movies;
18 end
```

**Algorithm 7:** Demographics-based recommendation algorithm using favorite movies

### 5.3.2 Using the User's Demographic Data

Typicality may also be used to define a method to compute recommendations based on the audience who typically liked some items. The method previously described in subsection 5.3.1 requires user ratings to provide recommendations, and is thus unable to handle the *new user* problem. In this subsection we address this issue by proposing a method that is capable of computing recommendations for a new user. The only data necessary include the new user's demographic data as well as ratings from *other* users — none from the new user. This method starts with computing the favorite items of users based on each profile feature value (*e.g.* for the age feature, we compute a set of favorite items for users [-18], another for users [18-24], etc). Then, we aggregate the sets of favorite items to fit the list of profile characteristics of a user.

Unlike the approach presented in subsection 5.3.1, this approach does not require any ratings to provide results, and as such constitutes a solution to the *new user* problem.

#### 5.3.2.1 Computing Items Typically Liked by People Based on One Characteristic

For each profile characteristic $c \in C$ of the current user $u_{current}$ of value $v$, select the other users $U_v \subseteq U$ with the same profile characteristic value $v$. For each such user $u \in U_v$, compute the fuzzy set of items $E_c(u)$ each user liked with elements of the form $r/m$ where $r$ is the rating (divided by 5 to fit the $[0, 1]$ scale, which provides values such as 0.2, 0.4, 0.4, 0.8 and 1) given by user $u$ to item $m$.

Then compute the multiset sum $E_c(U_v)$ from the different $E_c(u)$ sets with $u \in U_v$ by writing each element in the form $(r_{avg}, nb)/m$, where $r_{avg}$ is the average rating of users in $U_v$ for the item $m$, and $nb$ the number of users who rated it.

Then, compute $T_c(U_v)$ from $E_c(U_v)$, $T_c(U_v)$ being the typical set of items representing the users in $U_v$. In $T_c(U_v)$ each element is of the form $\mu/m$, where $\mu$ is the min between $r_{avg}$ and $nb/card(E_c(U_v))$ in $E_c(U_v)$ for $m$, cf Example 5.13 below.

**Remark 5.4** *The aggregation of the two degrees $r_{avg}$ and $nb/card(E_c(U_v))$ with a* min *may be debatable. $r_{avg}$ is an average satisfaction degree among the ratings obtained, while $nb/card(E_c(U_v))$ denotes whether the item has obtained many ratings. With this interpretation, "average" popular (with many ratings) items will be favored over excellent items unheard of (with few ratings).*

#### 5.3.2.2 Aggregating Typical Sets of Items

For all characteristics $c \in C$, we aggregate the multisets $T_c(U_v)$ with an intersection (for instance using the t-norm min) to obtain the items typically liked by people based on all criteria, resulting in $T_C(U_v)$. Items need to be in every $T_c(U_v)$ fuzzy set to appear in $T_C(U_v)$ with the min t-norm. Should no item satisfy this condition, another aggregation operator (such as the mean) should be considered instead of the intersection.

Final step: return the elements with the highest typicality degree in $T_C(U_v)$.

**Example 5.13** *Let us consider the users described in Table 5.1, and start looking for recommendations for Sophie. Based on her age, Sophie is similar to Melanie and Aaron. Based on her occupation, Sophie is similar to Hiroki. Based on her location, Sophie is similar to Alice and Nolan. Based on her gender, Sophie is identical to Melanie and Alice (the* MovieLens *dataset only features two genders). To assess to what extent users are similar, degrees of similarity — computed by the means of similarity relations, such as identity for the gender, a metric distance for the age and location and a semantic distance for the occupation — are considered. In this example we consider a crisp vision of similarity for the sake of clarity.*

*The known ratings associated with the users in Table 5.1 are presented in Table 5.2. Considering the age characteristic, we look for the favorite movies of both Melanie and*

Table 5.1 – User profile examples

| *Name* | *Gender* | *Age* | *Occupation* | *Area* |
|---|---|---|---|---|
| Sophie | F | 25 | Grad Student | Denmark |
| Hiroki | M | 16 | Pupil | Japan |
| Aaron | M | 32 | Engineer | Canada |
| Melanie | F | 29 | Sales Exec. | Spain |
| Alice | F | 43 | Educator | Germany |
| Nolan | M | 34 | Lawyer | Netherlands |

Table 5.2 – User ratings examples

| | Sophie | Hiroki | Aaron | Melanie | Alice | Nolan |
|---|---|---|---|---|---|---|
| Da Vinci Code (DVC) | ? | | | | 4 | |
| Harry Potter 2 (HP2) | ? | 4 | | | | 3 |
| Harry Potter 3 (HP3) | ? | | | 4 | | 5 |
| Harry Potter 4 (HP4) | ? | 1 | 3 | | | 4 |
| Star Wars IV (SW IV) | ? | | | | 2 | 4 |
| Terminator (T) | ? | 4 | 3 | 2 | 3 | |

*Aaron. By dividing their ratings by 5, we obtain:*
$E_{age}(Melanie) = \{0.4/T,\ 0.8/HP3\}$ *and* $E_{age}(Aaron) = \{0.6/T,\ 0.6/HP4\}$.

*Then we aggregate these sets and obtain:*
$E_{age}(U_v) = \{(0.5, 2)/T,\ (0.8, 1)/HP3,\ (0.6, 1)/HP4\}$.

***Remark 5.5*** *To compute average marks in $E_c(U)$ we only consider the ratings given by users and divide them by the number of users having rated the movie, leading to an average of 0.8 for the movie Harry Potter 3.*

*The typical set of movies obtained is:*
$T_{age}(U_v) = \{\min(0.5, 1)/T,\ \min(0.8, 0.5)/HP3,\ \min(0.6, 0.5)/HP4\}$
$T_{age}(U_v) = \{0.5/T,\ 0.5/HP3,\ 0.5/HP4\}$.

*Assuming that for the other characteristics we obtain:*
$T_{area}(U_v) = \{0.6/SW\ IV,\ 0.5/T,\ 0.5/DVC,\ 0.3/HP2,\ 0.5/HP3,\ 0.4/HP4\}$ *and*
$T_{occupation}(U_v) = \{0.5/T,\ 0.4/HP2,\ 0.2/HP4\}$.

*By aggregating these three sets, we finally get:*
$T_C(U_v) = \{0.5/T,\ 0.2/HP4\}$.$\diamond$

This approach is summarized in Algorithm 8. The first step corresponds to lines 2–9, and the second to lines 10–11.

### 5.3.3 Using the User's Favorite Movies and Demographic Data

In both approaches, it is possible to consider domain-dependent filters to reduce the number of items to consider. In this case, to reduce the number of movies and alleviate computational costs, we suggest limiting the search to one movie genre in particular, so permitting users to specify which movie genre they are interested in.

**Input:** a target user $u \in U$ with characteristics $C$ ; a matrix of ratings $M \times U$
**Output:** a set of predictions for user $u_{target}$

```
1  begin
2      foreach c ∈ C do
3          select u_target.c value;
4          foreach u ∈ U_v with u_target.c value do
5              compute E_c(u);
6          end
7          compute E_c(U_v);
8          compute T_c(U_v);
9      end
10     compute T_C(U_v);
11     return the top T_C(U_v) values;
12 end
```

**Algorithm 8:** Demographics-based recommendation algorithm using user demographics

The two approaches can be combined naively by computing the two separately and then intersecting the results (the first one uses similarity matching degrees between a set of movies liked by the user and other movies, and the second one uses typicality degrees of movies associated with a set of users). They can be seen as complementary as the first one requires more data to work, while the second one can handle the new user problem, simply using the user's demographic data. We intend to combine the two approaches by using the second one with new users, for so long as they do not have enough ratings. Once they have rated at least $k$ items, we only use the first approach.

## 5.4 Explaining Recommendations

Explanations to recommendations increase the trust of users in the system [McSherry, 2005] and satisfy their curiosity when facing unexpected recommendations, provided that they do not invade their privacy [Jeckmans et al., 2013]. We propose explanations leveraging the associations between the target items and the recommended items for the approaches presented in the previous sections.

### 5.4.1 Association-Based Approach

With this approach, explanations are based on the associations created by primary key-foreign key relationships. They highlight the typical associations between recommended actors and the target actor, as well as the atypical properties that distinguish the different recommended actors.

#### 5.4.1.1 Use of Foreign Keys

For every similarity criterion there is at least one element of explanation provided as to why an actor is returned. The first answers provided by the prototype were rather confusing since we did not know what kind of links there were between the target actor and the actors returned. Implementing the explanation mechanism contributed to justifying the returned recommendations.

The idea behind these explanations is to leverage the structure of the database, more precisely the primary key-foreign key relationships between the tables. The link between a target actor and a candidate similar actor is based on the similarity criterion used to compute extended answers. The query used to retrieve candidate similar actors also serves as an explanation as to why their are similar: candidate similar actors are part of the result and thus satisfy the criterion. Additional queries may be run to provide further information detailing the links between the target actor and the candidate similar actor once the top similar actors have been selected.

**Directors Criterion** To justify how related an actor and a director are, we need to look for movies in which the two collaborated. Here is the query:

```
SELECT d.name, m.title FROM movies m, directors d
WHERE (d.id, m.id) IN
(SELECT md.director_id, md.movie_id FROM mov_cast mc, mov_directors md
WHERE mc.actor_id = SIMILAR_ACTOR
AND mc.movie_id = md.movie_id
AND md.director_id IN (TYPICAL_DIRECTORS))
order by d.name;
```

Ordering the results by director names and then by years is an arbitrary choice for display purposes. The link between primary and foreign keys is rather explicit in this query, with the associations between the tables *movies* and *directors* with *mov_ cast* and *mov_ directors*.

**Actors Criterion** Actors can be related in two ways: the obvious one is that they have starred together in movies. The other one is that they have similar sets of co-starring actors. While the former helps to understand why actors are returned, the latter is used as a similarity criterion. Obtaining the movies in which the two actors starred is straightforward.

```
SELECT m.title, m.production_year FROM movies m
WHERE m.id IN
(SELECT mc1.movie_id FROM mov_cast mc1, mov_cast mc2
WHERE mc1.actor_id = TARGET_ACTOR
AND mc2.actor_id = SIMILAR_ACTOR
AND mc1.movie_id = mc2.movie_id)
order by m.production_year;
```

However getting the actors who played with both actors is more costly. Let us not forget that we decided that actors must have played together more than once to be considered similar. There are many joins in the query, so it can take a while to process (there are more than 1.7 million actors in the database).

```
SELECT actors.id, actors.name, count(actors.id)
FROM mov_cast mc1, mov_cast mc2, mov_cast mc3, mov_cast mc4, actors
WHERE mc1.actor_id = TARGET_ACTOR
AND mc1.movie_id = mc2.movie_id
AND mc2.actor_id != TARGET_ACTOR
AND mc2.actor_id = mc3.actor_id
AND mc3.actor_id != SIMILAR_ACTOR
AND mc3.movie_id = mc4.movie_id
AND mc4.actor_id = SIMILAR_ACTOR
AND actors.id = mc2.actor_id
GROUP BY actors.id, actors.name
HAVING COUNT(DISTINCT mc2.movie_id) > 1
ORDER BY count(actors.id) DESC;
```

**Genres Criterion** Figuring out the movie genres typically associated with actors is easy, and the most significant way to underline it is probably to count the number of movies of these genres in which the actor starred.

```
SELECT count(m.title) FROM movies m
WHERE m.id IN
(SELECT mg.movie_id FROM mov_genres mg, mov_cast mc
WHERE mc.actor_id = SIMILAR_ACTOR
AND mc.movie_id = mg.movie_id
AND mg.genre_id IN (TYPICAL_GENRES));
```

Other explanations were considered but to no avail due to the lack of data in the database – such as the order of importance of actors in a movie. When looking for actors similar to Tom Cruise, we can get Nicole Kidman, with the following explanation:

- Nicole Kidman played in Days of Thunder, directed by Tony Scott, director typically associated with Tom Cruise.
- Nicole Kidman and Tom Cruise played in Days of Thunder, Far and Away, Eyes Wide Shut, Stanley Kubrick: A Life in Pictures, Boffo! Tinseltown's Bombs and Blockbusters, The Queen, and August.
- Also they often played with Tom Hanks, Robert Redford, Steven Spielberg, Dustin Hoffman, and Sydney Pollack.
- Nicole Kidman played in 37 films from the genres Action, Thriller, and Drama, typically associated with Tom Cruise.

Let us note that it is straightforward in our case to provide these explanations as it is the same mechanism behind recommendations and explanations. In [Fang et al., 2011] the authors looked for ways to explain the relationships between pairs of entities independently of the way the relationships were discovered. They used a knowledge graph based on DBpedia[2] to compute their explanations. To do this they re-used and adapted algorithms from [Agrawal et al., 2002, Bhalotia et al., 2002, Hristidis and Papakonstantinou, 2002], as the explication of paths is analog to keyword search in databases.

#### 5.4.1.2 Use of Atypical Properties

Did you know that Michael Jackson used to be pretty good at karate? While this may seem unusual and rather out of the blue, this is an example of unexpected information given by the authors of [Tsukuda et al., 2013]. They use popularity and unpopularity to match objects so they may figure out how expected or unexpected some relationships may be. Our approach to compute similar objects also resides in knowledge discovery: it is not simply about recommending similar items, it is also about highlighting the differences between these items to present the user with informative answers. By defining atypicality we can figure out which properties stand out in a set of somewhat similar items (that have been found to be typical w.r.t. some similarity criterion), and what makes these items different. We provide an example below to illustrate our objective with atypical properties.

**Example 5.14** *Let us consider the IMDb database, and look for actors similar to a target actor $x$, say Tom Cruise. Let us call this set of similar actors $Y$, obtained by choosing several similarity criteria. Let us assume that we obtain $Y =$ [Nicole Kidman, Steven Spielberg, William Mapother, Tom Hanks, Tim Robbins]. Our objective is to highlight which atypical properties each actor from $Y$ has, and that the others actors from $Y$ do not have. For instance, we may obtain the following explanation:* "Unlike other actors similar to Tom Cruise, Steven Spielberg played mostly in movies of the genre Documentary."

Let us consider a database $D$ and a target object $x$. Let us choose a few similarity criteria – or associations – to compute elements typically associated with $x$. For each criterion $c$ we will compute for $x$ a multiset $E_c(x)$ and then a fuzzy set $T_c(x)$ representing how typical the elements are in $E_c(x)$. After having filtered the potential similar items, multisets and fuzzy sets are computed for them too, one for each similarity criterion $c$ considered. In order to compare them, we compute and compare the fuzzy sets and we get a matching degree. With these degrees we can now rank the potential similar items and only retain the top-$k$. Let us call this set of similar items $Y$, and choose $k = 5$, we get $Y = [y_1, y_2, y_3, y_4, y_5]$. Explanations are then formulated to justify how close (in terms of shared typical properties) each similar item $y \in Y$ is to the target object $x$. Now we desire to highlight what distinguishes every element $y$ from the other members

[2] http://wiki.dbpedia.org/About

of the set $Y$. We are going to consider the similarity criteria used before to compute the similar items. For every item $y$ we have already computed the multisets and fuzzy sets $E_c(y)$ and $T_c(y)$ associated with each criterion $c$. The objective is to compute the fuzzy sets $T_c(Y)$ associated with each criterion $c$, representing how typical the elements are in the multiset $E_c(Y)$. Here are two ways to do this:

- compute $E_c(Y)$ from the multisets $E_c(y)$ – through a multiset sum, see equation 5.18 – and then compute $T_c(Y)$;

$$E_c(Y) = \biguplus E_c(y) \tag{5.18}$$

- compute $T_c(Y)$ directly from the $T_c(y)$ – through an arithmetic mean, see equation 5.19.

$$T_c(Y) = \{\mu/x_i, \mu = \frac{1}{|Y|}\sum_{y \in Y} \mu_y, \mu_y/x_i \in T_c(y)\} \tag{5.19}$$

Let us consider the following fuzzy sets associated with the items $y$:
$T_c(y_1) = \{0.1/a,\ 0.2/b,\ 0.05/e,\ 0.1/k\}$
$T_c(y_2) = \{0.1/b,\ 0.2/c,\ 0.05/h,\ 0.1/i\}$
$T_c(y_3) = \{0.1/a,\ 0.2/f,\ 0.05/g,\ 0.1/k\}$
$T_c(y_4) = \{0.1/d,\ 0.2/g,\ 0.05/h,\ 0.1/k\}$
$T_c(y_5) = \{0.1/e,\ 0.2/f,\ 0.05/g,\ 0.1/h\}$
With Equation 5.19 the resulting fuzzy set associated with the set $Y$ is:
$T_c(Y) = \{0.04/a,\ 0.06/b,\ 0.04/c,\ 0.02/d,\ 0.03/e,\ 0.08/f,\ 0.06/g,\ 0.4/h,\ 0.02/i,\ 0.06/k\}$

Once this fuzzy set $T_c(Y)$ has been computed, we can start comparing each element $y$ to the set $Y$ thanks to the fuzzy sets $T_c(y)$ and $T_c(Y)$, the objective being to find differences between them to distinguish $y$ from other similar elements. As $T_c(y)$ and $T_c(Y)$ can be assimilated to vectors, we can compute the difference between each coordinates and retain the highest differences obtained.

**Remark 5.6** *It may be interesting to separate two categories of elements here: elements which are prominently present in $y$ and absent in $Y$, and those prominently present in $Y$ and absent in $y$.*

As actors tend to have rather dense careers, and to collaborate with many directors and fellow actors, there may be many of them with which only one of the similar actors would have interacted. Should this number of individuals be too high for a given similar actor, selecting an adequate number of atypical properties would be needed to limit the results. A contrario, should the distribution of the results be not sparse enough, thresholds indicating from which point a property becomes atypical would also be needed.

One starting point would be to look at the distribution of the values in $T_c(Y)$ and compute the arithmetic mean of the typicality degrees for instance. Then, all values above this mean should represent properties rather typical of the set. Any item $y$ which

would have a 0 associated with this property may stand out. On the other hand, values which are below the arithmetic mean should represent properties rather typical of only one or very few items. Considering the values above, we get a mean of 0.045 for the values of $T_c(Y)$. The properties $b$, $f$, $g$, $k$ are all above the arithmetic mean, and $f$ is the highest. Let us now compute the difference between the typicality degrees of $T_c(Y)$ and each $T_c(y)$. The results are presented in Table 5.3. The highest degrees are highlighted in green, and refer to properties that are far above the mean of the typicality degrees, so typical of the set $T_c(y)$. The lowest ones are highlighted in orange, and refer to properties rather typical of $T_c(Y)$.

Table 5.3 – Difference degrees between each item

| | y1 | y2 | y3 | y4 | y5 |
|---|---|---|---|---|---|
| a | 0.06 | -0.04 | 0.06 | -0.04 | -0.04 |
| b | 0.14 | 0.04 | -0.06 | -0.06 | -0.06 |
| c | -0.04 | 0.16 | -0.04 | -0.04 | -0.04 |
| d | -0.02 | -0.02 | -0.02 | 0.08 | -0.02 |
| e | 0.02 | -0.03 | -0.03 | -0.03 | 0.07 |
| f | -0.08 | -0.08 | 0.12 | -0.08 | 0.12 |
| g | -0.06 | -0.06 | -0.01 | 0.14 | -0.01 |
| h | -0.04 | 0.01 | -0.04 | 0.01 | 0.06 |
| i | -0.02 | 0.08 | -0.02 | -0.02 | -0.02 |
| k | 0.04 | -0.06 | 0.04 | 0.04 | -0.06 |

While the computation of the differences is possible for the aforementioned similarity criteria – used to compute the similar actors in the first place – it should be noted that new criteria could be used to compute differences as well. Indeed there may be some criteria that do not seem interesting to be used to compute similarity, but that would provide interesting information on atypical properties. Beyond associations, it could also be attributes associated with the actor such as their date of birth. In the example presented below, it could be noted that Nicole Kidman is the only woman in the set of actors recommended. Also, other data sources would provide valuable information to highlight atypical properties. Bar charts representing the value of each property for each actor would highlight which ones are singular. It is also possible to compute atypical properties that define $x$ with regard to the set of similar elements $Y$.

Algorithm 9 describes a method to compute these "unique" properties that distinguish objects. The first part (Lines 3 to 17) is the same as in Algorithm 6. Then for every criterion $c$ the fuzzy set $T_c(Y)$ is computed (Line 19). For every element $y \in Y$ the differences between $y$ and $Y$ are computed (Line 21). Finally the differences between the target object $x$ and the set of similar items $Y$ are computed (Line 23).

**Example 5.15** *Let us consider the IMDb database, and look for actors similar to a target actor $x$, say Tom Cruise. Let us choose a few similarity criteria – or associations – to compute elements typically associated with $x$. We can choose the actors, directors and genres criteria for instance, and for each criterion $c$ we will compute for $x$ a multiset*

**Input:** a target object $x$ ; $n$ specifications of multisets (i.e., $n$ subqueries) ;
**Output:** a set of distinguishable properties for each object $y$ similar to $x$

```
1  begin
2      Y(x) ← ∅;
3      for i ← 1 to n do
4          compute E_i(x);
5          compute T_i(x) from E_i(x);
6      end
7      foreach item y in the relation concerned do
8          for i ← 1 to n do
9              compute E_i(y);
10             compute T_i(y) from E_i(y);
11             compute the degree of matching μ_i between T_i(x) and T_i(y);
12         end
13         μ ← min_{i=1..n} μ_i;
14         if μ ≥ α then
15             Y(x) ← Y(x) ∪ {μ/y}
16         end
17     end
18     for c ← 1 to n do
19         compute T_c(Y) from E_c(Y) or the different T_c(y)
20         foreach item y in Y(x) do
21             Compute the difference between T_c(y) and T_c(Y) for each property and
                return the highest
22         end
23         Compute the biggest differences between T_c(x) and T_c(Y)
24     end
25 end
```

**Algorithm 9:** How to compute atypical elements

*$E_c(x)$ and then a fuzzy set $T_c(x)$ representing how typical the elements are in $E_c(x)$. After having filtered the potential similar actors, multisets and fuzzy sets are computed for them too, one for each similarity criterion c considered. In order to compare them, we compute and compare the fuzzy sets and we get a matching degree. With these degrees we can now rank the potential similar actors and only retain the top-k. Let us call this set of similar actors $Y$, and choose $k = 5$. We get $Y$ = [Nicole Kidman, Steven Spielberg, William Mapother, Tom Hanks, Tim Robbins].*

*Let us consider the genres criterion. For every genre g and for every actor y, a typicality degree $\alpha_{y,g}$ has been computed. Then the difference between each typicality degree and the typicality degrees of $T_{genres}(Y)$ – with equations 5.18 and 5.19 – is computed for each actor. A positive number will indicate that the actor is typically more associated with this genre than the set of actors, while a negative number will indicate the opposite.*

*Thus, the prototype will provide indications such as:*

- *Unlike other actors similar to Tom Cruise, Steven Spielberg played mostly in movies of the genre Documentary.*
- *Unlike other actors similar to Tom Cruise, Steven Spielberg did not often play in movies of the genre Drama.*
- *Unlike other actors similar to Tom Cruise, Tom Hanks played mostly in movies of the genre Comedy.*$\diamond$

It is also possible to provide information as to what makes the target actor unique compared to the recommendations provided. This can be done in the same manner by computing the difference between the typicality degrees of the target actor with those of the set of similar actors.

### 5.4.2 Typicality-Based Approach Leveraging Demographic Data

Demographic explanations may be considered as "awkward" since people may feel uncomfortable receiving recommendations seemingly based on stereotypes. Two forms of demographic explanations are considered here, based on:

- the movies associated with those liked by the user (Subsection 5.3.1), leading to explanations such as "X-Men *was recommended to you because you liked movies such as* Harry Potter *and* LOTR*, and all of these movies have the same typical audience who liked them;"*
- the user's own characteristics in the case of the new user problem (Subsection 5.3.2), leading to explanations such as "X-Men *was recommended to you because this movie is typically liked by* young men in college *such as yourself."*

As the similarity score is based on the minimum between the matching degrees for all profile characteristics, and the occupation characteristic often turns up in the matrix, providing more detailed explanations — than "have a similar audience" — for the first

approach may be difficult. Indeed, the occupation characteristic has the highest number of modality values (21 in the 1M MovieLens dataset) so it may be misguiding to present a modality actually explaining the recommendation if it is so only by a small degree.

## 5.5 Experiments

In this Section we present results from our two prototypes: **ReSO** and **TyDR**. We evaluate the interest of explanations accompanying recommendations for **ReSO**, and compare the recommendations computed by **TyDR** to other state-of-the-art recommenders.

### 5.5.1 Computing Actors with Associations

A user study was conducted to evaluate the relevance of the explanations accompanying recommendations proposed by **ReSO**, and check whether explanations were important for users to accept recommendations. We did so by asking users whether recommendations were relevant, first without giving any explanations and then by giving some (if the recommendations alone were deemed irrelevant). Figure 5.3 features the survey along with an example of recommendation provided to users.

Users were asked to rate the relevance of recommended actors from a target actor without directly being aware of the criteria used to compute the similarity. However the format of the explanation gave the user an idea of the latter. Two factors are considered here: the opinion of the user from only the names of the two actors, and the opinion of the user once he has taken into account the explanation. We considered four target actors: Tom Cruise, Johnny Depp, Hugh Jackman and Nicole Kidman. The user study is divided into four parts (one per actor). The study proposes similar actors based on the director, actor and genre criteria, with the matching measure described by equation (5.9) and aggregation by the min. The study required a couple of minutes to be completed. Twenty-one users completed the study. For all four target actors, the results presented in Table 5.4 are obtained. It shows that the proposed actors are considered relevant on an average of 45.7 % prior to the explanation, and then 80.8 % after explanation. This shows the importance of explanations in the process of understanding recommendations.

Figure 5.4 details the relevance scores for each actor before having explained why they were returned: the gap between Johnny Depp and Nicole Kidman is close to 25% which may be explained by their notoriety. However Figure 5.5 underlines that regardless of the actor, the average relevance is always very close to 80%, which is very comforting in the idea that the explanations are convincing.

### 5.5.2 Computing Movies Based on the Audience

In this Subsection we evaluated our approach presented in Subsection 5.3.1 (henceforth referred to as **TyDR**) on other aspects: we sought to determine how it fares against state-of-the-art recommenders on several metrics defined below. We experimented with

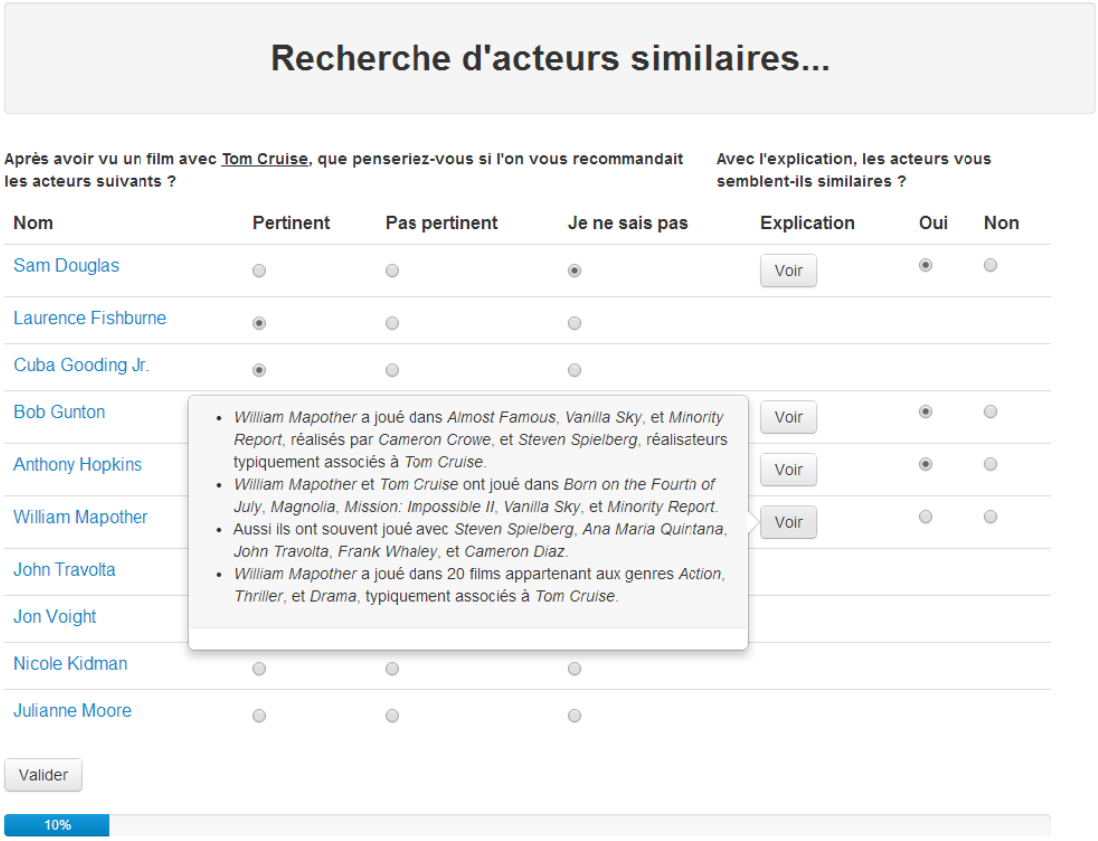



Figure 5.3 – Survey screenshot.

Table 5.4 – Evaluation of the relevance of actors, in %

| | Before explanation | After explanation |
|---|---|---|
| Relevant | 45.7 | 80.8 |
| Not relevant | 12.9 | 19.2 |
| Does not know | 41.4 | 0 |

the 1M MovieLens[3] dataset which contains some demographic information about users, such as their age, occupation, gender and zip code. The 1M dataset (1,000,209 ratings) contains 3,883 movies partially rated by 6,040 users. The sparsity of the dataset is $1 - \frac{1,000,209}{3,883*6,040} = 0.957$. The dataset is split into a training set $R_{train}$ and a test set $R_{test}$ with ratios 80% and 20% respectively. These sets are subsets randomly extracted from the 1M dataset.

Following the **TyDR** algorithm, a prediction score between 0 and 1 is given for each movie for each user based on some of their ratings. This score is then transformed into

[3]http://grouplens.org/datasets/movielens/

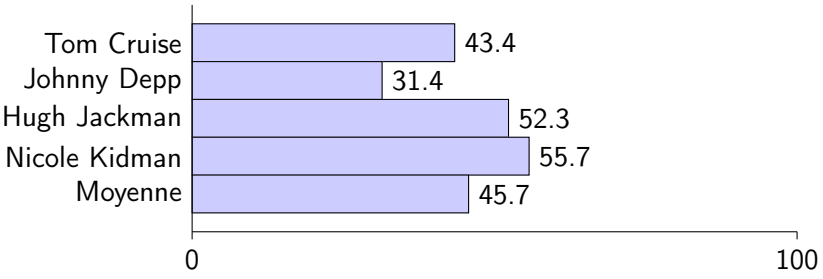



Figure 5.4 – Relevance **before** explanation

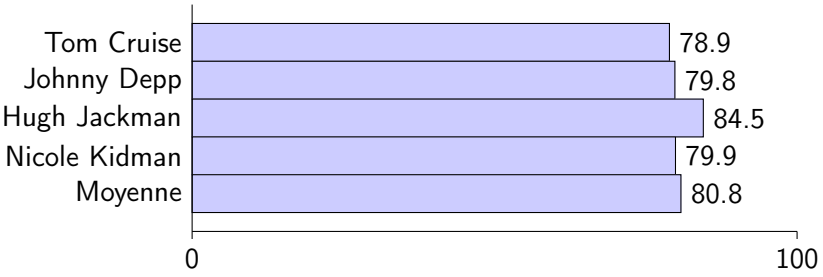



Figure 5.5 – Relevance **after** explanation

a rating on the 1-5 star scale. To this end, we considered the following mapping:

$$\text{prediction} = 4 * \text{score} + 1.$$

We evaluated **TyDR** and compared it to other standard approaches such as user-based collaborative filtering (UB CF), item-based collaborative filtering (IB CF), personalized mean [Lemire and Maclachlan, 2005] (PM) and matrix factorization [Funk, 2006] (MF). To the best of our knowledge, given the low interest of the RS community for demographic RS, there is no baseline to evaluate recommendations purely based on demographic data. We used the following metrics: mean average error (MAE), root mean square error (RMSE), precision, recall, F-measure, and coverage. For a given user $u$ and item $i$, we have a hidden reference rating $r_{ui}$, and we compute a prediction $p_{ui}$.

The MAE and RMSE are popular metrics for evaluating the difference between the predicted ratings and the actual ratings.

$$\text{MAE} = \frac{1}{|R_{test}|} \sum_{r_{ui} \in R_{test}} (|r_{ui} - p_{ui}|) \qquad \text{RMSE} = \sqrt{\frac{1}{|R_{test}|} \sum_{r_{ui} \in R_{test}} (r_{ui} - p_{ui})^2}$$

The precision, recall, F-measure and coverage enable offline evaluation of recommender systems. They are computed based on the classification of ratings according to Table 5.5.

$$\text{Precision} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fp}} \qquad \text{Recall} = \frac{\#\text{tp}}{\#\text{tp} + \#\text{fn}}$$

Table 5.5 – Classification of a recommendation

| | Recommended ($p \geqslant 4$) | Not Recommended ($p < 4$) |
|---|---|---|
| Liked (rated 4+) | True Positive (tp) | False Negative (fn) |
| Not Liked (rated 3-) | False Positive (ft) | True Negative (tn) |

Table 5.6 – Experiment results

| Algorithm | MAE | RMSE | Precision | Recall | F-measure | Coverage |
|---|---|---|---|---|---|---|
| TyDR | 0.85 | 1.09 | 0.65 | 0.81 | 0.72 | 0.35 |
| UB CF | 0.72 | 0.89 | 0.83 | 0.53 | 0.65 | 0.61 |
| IB CF | 0.69 | 0.85 | 0.87 | 0.50 | 0.63 | 0.63 |
| MF | 0.70 | 0.86 | 0.86 | 0.48 | 0.62 | 0.67 |
| PM | 0.73 | 0.89 | 0.85 | 0.41 | 0.56 | 0.52 |

$$\text{F-measure} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \qquad \text{Coverage} = \frac{\#\text{distinct items recommended}}{\#\text{distinct items in the database}}$$

We present a comparison of our approach with other approaches in Table 5.6. Our method is not as effective as the others in terms of MAE and RMSE: indeed, many predictions (over 50%) fall in the 4 stars category, somewhat showing signs of leniency of our approach. This is highlighted with the precision scores. However, the recall is much higher with our approach, and the resulting F-measure is also higher than that of all other tested approaches.

We also compare the correct predictions of all evaluated systems together in Table 5.7. As it turns out, our approach can correctly predict around 14% of the ratings that another approach would not. Assuming we would fuse all other approaches and always acquire the best prediction possible from all of them, our approach can still provide as many as up to 5.9% good predictions that the others combined cannot. Typicality-based CF leveraging demographic data offers another way to compare items, and so is a good complementary approach that would deserve to be fused with other methods.

**Example 5.16** *Let us present a few recommendations obtained by our system that other tested systems could not provide.*

Table 5.7 – Comparison of recommendation results: proportions of correct predictions found by Algorithm 1 and not found by Algorithm 2

| | | Algorithm 2 | | | | |
|---|---|---|---|---|---|---|
| | | TyDR | UB CF | IB CF | MF | PM |
| Algorithm 1 | TyDR | 0 | 0.13 | 0.13 | 0.15 | 0.14 |
| | UB CF | 0.21 | 0 | 0.09 | 0.13 | 0.10 |
| | IB CF | 0.22 | 0.11 | 0 | 0.12 | 0.10 |
| | MF | 0.24 | 0.14 | 0.01 | 0 | 0.14 |
| | PM | 0.19 | 0.08 | 0.07 | 0.11 | 0 |

- *User 5811 was recommended William Shakespeare's Romeo and Juliet (1996) because he liked movies such as Welcome to the Dollhouse (1995), Cruel Intentions (1999), and Great Muppet Caper, The (1981), which all have very similar audiences.*

- *User 1051 was recommended Ghostbusters (1984) because she liked movies such as Silence of the Lambs, The (1991), Saving Private Ryan (1998), and Indiana Jones and the Last Crusade (1989), which all have very similar audiences.*

- *User 4022 was recommended Star Wars: Episode IV - A New Hope (1977) because he liked movies such as Raiders of the Lost Ark (1981), Back to the Future (1985), and Saving Private Ryan (1998), which all have very similar audiences.*◇

A reason why classical CF approaches do not recommend these movies can be simply that they have not been rated by any user who has a rating behavior sufficiently similar to the current user's. Our approach recommends items based purely on the demographics of the audience who liked movies. This is an advantage when considering users that may have an unusual rating pattern and who may have difficulty getting good recommendations from user-based CF systems.

## 5.6 Summary

In this chapter we have presented recommendation approaches leveraging associations between entities and user demographics. The approaches were applied to cinematographic databases and can be extended to any context, provided that some similarity criteria or demographic data are available. The recommendations are provided with explanations as to how they were computed so that users may understand why they are proposed. For both approaches, providing explanations demonstrates a form of transparency toward users, enabling them to understand how and why they are computed. Indeed, our approaches can provide explanations for every recommendation formulated. These explanations were regarded as helping users understand recommendations, and deeming them relevant in the case of **ReSO**.

We proposed to use typicality in recommendation with the objective to propose more sensible recommendations, all the while having a basis to formulate explanations. Typicality has already been used with RS in [Cai et al., 2014], however we focus on individual elements (users or items) and their associations to determine which of these associations are typical, while the authors of [Cai et al., 2014] focus on groups (of users or items) created with clustering algorithms to identify typical behaviors. However the explanations obtained for demographic-based recommendations were not always particularly relevant, and could be lacking in interpretability.

We considered demographics as an additional source of data to propose recommendations. However pure demographic recommenders and demographic data are scarce: as such there is no benchmark to easily compare demographic approaches. While some results concerning **TyDR** are not satisfactory, others prove that the approach has a few

advantages : predicting ratings that other state-of-the-art systems cannot, and having a rather good F-measure.

Perspectives include applying the principle of the approach leveraging demographics to other contexts, while still considering associations as the basis to compute recommendations with explanations. We also wish to reconsider ratings more globally with **TyDR** so as to differentiate ratings of 4 and 5. Another direction we consider is leveraging the items not liked (with low reviews) to find items of interest. Finally, we also wish to conduct another user study to measure the quality of the explanations provided, such as their interpretability and their helpfulness.

# Chapter 6

# Building and Explaining Queries with Examples

Storing and structuring information efficiently are some of the keys to facilitating data browsing and retrieval. Commercial websites provide easy-to-use interfaces to navigate around their data, *e.g.* with keyword search. Other search paradigms — such as faceted search or attribute filtering — were also introduced to ease date browsing.

Query by Example is a paradigm in information retrieval to acquire results based either on:

- one (or several) input tuple(s) provided by the user;
- or the evaluation of prototypical examples (positively, negatively, on a predefined scale, ...) reflecting the content of the database.

The expected output contains elements that are similar to the input tuple(s) provided as example(s), or that reflect the choices of the user if prototypical examples were evaluated. For example, if a user browses houses and positively evaluates houses with 3 bedrooms and negatively evaluates houses with a small garden, then the results will include houses with 3 bedrooms and a "not small" garden.

In this chapter we focus on the prototypical-example-based Query By Example (QBE) paradigm, in which users are asked to evaluate samples from the database to find items similar to the ones they evaluate positively — *positive examples* — while dismissing those evaluated negatively — *counter-examples*. The prototypical-example-based QBE implies two main challenges: (i) the selection of examples to submit for user evaluation, and (ii) the interpretation of these user evaluations to infer a query retrieving relevant results.

The selection of examples representative of a dataset can be viewed as a machine learning task similar to *boosting* algorithms [Kearns and Valiant, 1994]. Starting from one "good" example, the search for new examples is a compromise between *exploitation* — ensuring obtaining other "good" examples with similar values — and *exploration* — aiming at obtaining more diverse examples, albeit not so "good." When selecting examples we aim at balancing exploitation and exploration.

The elicitation of user preferences stems from the domain of *preference learning* [Fürnkranz and Hüllermeier, 2011]. Preference learning techniques aim at inferring a preference model (utility function or binary relation) from examples or user feedbacks. In this approach we do not aim at learning a preference model but descriptions of a set of examples. However we do rank examples to provide to the user in preparation of the preference learning step.

**In this chapter we present Fuzzy Query By Example (FQBE), a QBE approach based on fuzzy logic principles. Fuzzy logic provides tools to express and infer preferences in a flexible way. Our objective is to help users obtain answers with a simple binary evaluation of examples.** As in [Zadrozny et al., 2010], we consider a fuzzy vocabulary for each attribute domain in the database. This vocabulary enables us to formulate, in a linguistic way, descriptions of the attribute values shared by positive examples that are not shared by counter-examples (later defined as *characterizations*).

**Example 6.1** *Let us consider that the user is given to evaluate the examples in Table 6.1. Based on the obtained evaluations, with FQBE we infer that the user is interested in cars that have:* small *engine size,* medium *price, and* very low, low or medium *consumption (mostly low).*

In addition to returning the items that best match the examples evaluated by the user, we also provide the user with an interpretable linguistic explanation of the fuzzy query inferred by the system. In other words, the user *knows* what the system *believes* about his/her preferences. We implemented FQBE and conducted experiments to demonstrate the interest of our approach. In this chapter we show how to:

1. Select which examples in a dataset should be submitted to the user for evaluation, leveraging a fuzzy vocabulary;

2. Use the evaluations to deduce the data properties the user is interested in;

3. Translate these properties into a query;

4. Provide the user with understandable explanations.

This chapter is divided as follows: first we recall notations for fuzzy queries and fuzzy vocabularies (Section 6.1). We determine which examples should be submitted for evaluation (Section 6.2), and how to infer user preferences from these evaluations (Section 6.3). Then we translate these preferences into a query (Section 6.4) and explain the query — as well as its results — to the user (Section 6.5). We conduct experiments to demonstrate the interest of our approach (Section 6.6). Finally we discuss the differences between FQBE and other approaches belonging to the QBE paradigm (Section 6.7) before concluding (Section 6.8).

Table 6.1 – Some attribute values of the example used in this paper

| year | fuel con. | mileage | engine | HP | price | make | eval |
|---|---|---|---|---|---|---|---|
| 2009 | 6 | 32500 | 1.4 | 85 | 9900 | seat | + |
| 2009 | 6 | 59000 | 1.4 | 85 | 8900 | seat | + |
| 2008 | 10 | 25000 | 2 | 136 | 20900 | volvo | - |
| 2008 | 10 | 50000 | 2 | 136 | 20900 | volvo | - |
| 2009 | 8 | 45000 | 1.4 | 85 | 9900 | seat | + |
| 2008 | 5 | 32500 | 1.4 | 70 | 10000 | mini | + |
| 2008 | 9 | 35000 | 2 | 140 | 27000 | ford | - |
| 2009 | 6 | 12000 | 1.6 | 115 | 16000 | chevrolet | - |
| 2009 | 8 | 6000 | 1.6 | 115 | 16000 | chevrolet | - |
| 2005 | 12 | 39000 | 2 | 140 | 15500 | audi | - |
| 2010 | 9 | 4000 | 1.9 | 105 | 20500 | seat | - |
| 2008 | 9 | 60000 | 2 | 140 | 23000 | audi | - |
| 2009 | 7 | 17000 | 2 | 136 | 31500 | peugeot | - |

## 6.1 Fuzzy Vocabulary

Before detailing FQBE we first recall some notions and notations on fuzzy vocabularies. The fuzzy vocabulary notations, definitions and properties are the same as those in Subsection 4.3.1.

Let $R$ be a relation defined on a set $\mathcal{A}$ of $q$ categorical or numerical attributes $\{A_1, A_2, \ldots, A_q\}$. A fuzzy vocabulary, denoted by $\mathcal{V}$, on $R$ is defined by means of fuzzy partitions of the $q$ domains. A fuzzy partition $\mathcal{P}_i$ associated with the domain $\mathcal{D}_i$ of attribute $A_i$ is composed of $m_i$ fuzzy sets $\{P_{i,1}, P_{i,2}, \ldots, P_{i,m_i}\}$, such that for all $x \in \mathcal{D}_i$:

$$\sum_{j=1}^{m_i} \mu_{P_{i,j}}(x) = 1,$$

where $\mu_{P_{i,j}}(x)$ denotes the degree of membership of $x$ to the fuzzy set $P_{i,j}$. Each fuzzy partition $\mathcal{P}_i$ is associated with a set of linguistic labels $\{L_1^i, L_2^i, \ldots, L_{m_i}^i\}$.

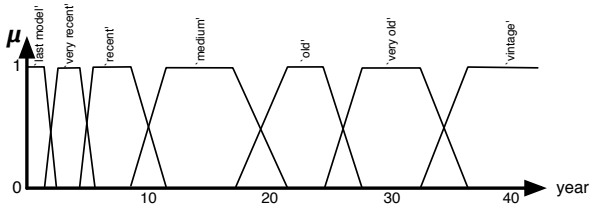



Figure 6.1 – A partition over the domain of the attribute *year*

For the case of numerical attributes (Fig. 6.1 illustrates a possible definition of

a partition over the attribute *year* of second-hand cars), it is also imposed, for the sake of interpretability, that a fuzzy set $P_i$ in a partition $\mathcal{P}_i$ can only overlap with its predecessor $P_{i-1}$ or/and its successor $P_{i+1}$ (when they exist). In this work we only consider numerical attributes. However FQBE is also applicable to categorical attributes: we only impose that for each attribute domain the sum of the membership degrees on all modalities of a partition be 1.

In the approach we propose, it is assumed that a vocabulary has already been manually defined on the data considered by a domain expert using a dedicated graphical interface such as ReqFlex [Smits et al., 2013], or using an automatic method of vocabulary elicitation from data [Marsala and Bouchon-Meunier, 1996, Guillaume and Charnomordic, 2004, Smits et al., 2017a].

**Definition 6.1 (Item rewriting vector)** *An item $x$ may be rewritten in terms of a vocabulary $\mathcal{V}$ as a vector of $\sum_{k=1}^{q} m_k$ membership degrees. This vector, denoted by $RV_x$, is of the form $\langle \mu_{P_{1,1}}(x.A_1), \ldots, \mu_{P_{1,m_1}}(x.A_1), \ldots, \mu_{P_{q,1}}(x.A_q), \ldots, \mu_{P_{q,m_q}}(x.A_q) \rangle$.*

**Example 6.2** *The rewriting of a car from 2007 is rewritten according to the* year *partition illustrated by the Figure 6.1 into the following vector: $\langle 0, 0, 0.5, 0.5, 0, 0, 0 \rangle$, or, only using non-zero values: $\langle 0.5/recent, 0.5/medium \rangle$.*$\diamond$

## 6.2 Selecting Examples

The generated query $Q'$ is inferred from the binary evaluations of items provided by the user. The first question to address concerns the selection of items to submit to the user such that their evaluation, whether positive or negative, will make it possible to determine if a term of the vocabulary should appear in the inferred query.

As the authors of [De Calmès et al., 2003] and [Zadrozny et al., 2010], we share the point of view of several existing approaches to QBE that it is more meaningful to evaluate real items from the DB instead of artificial ones. An optimal but unrealistic solution would be to find for each term of the vocabulary a set of items among which only one item fully characterizes the concerned term, all other values (i.e. rewritings) being equal. As it appears unconceivable to find such sets of items and to task the user to evaluate too many items, we propose a strategy that builds offline a reduced set $\mathcal{S}$ of $k$ items from the DB. These items are selected in such a way that they are representative of the vocabulary, of the dataset, and that they are as mutually diverse as possible.

The representativity of an item $x$ w.r.t. the vocabulary quantifies the extent to which the item may be precisely described by the different linguistic terms. This representativity degree w.r.t. the vocabulary, denoted by $RepV(x)$, is computed from the rewriting vector $RV_x$ as follows:

$$RepV(x) = \frac{1}{q} \sum_{A_i, i=1..q} \max_{j=1..m_i} \mu_{P_{i,j}}(x), \tag{6.1}$$

where $q$ is the number of dimensions over which each item is described. This representativity degree is thus maximal if $x$ fully satisfies one fuzzy set on each dimension (as opposed to somewhat satisfying more than one fuzzy set).

An item $x$ also has to be representative of a sufficiently large data subset. This second representativity degree, this time w.r.t. the dataset $\mathcal{D}$, is denoted by $RepD(x)$ and defined by:

$$RepD(x) = \frac{1}{|\mathcal{D}|} \times \sum_{x' \in \mathcal{D}} 1 - d(RV_{x'}, RV_x), \tag{6.2}$$

where $d(RV_{x'}, RV_x)$ is a distance measure between two rewriting vectors computed in the following way:

$$d(RV_{x'}, RV_x) = \frac{\sum_{\mathcal{P}_i \in \mathcal{V}} \sum_{P_{i,j} \in \mathcal{P}_i} |\mu_{P_{i,j}}(x) - \mu_{P_{i,j}}(x')|}{\sum_{k=1}^{q} m_k}, \tag{6.3}$$

where $\mathcal{P}_i$ is a partition of the vocabulary $\mathcal{V}$, and that $P_{i,j}$ is a modality of the partition $\mathcal{P}_i$. The first element of $\mathcal{S}$ is computed with:

$$Sc(x, \emptyset) = \top(RepV(x), RepD(x)). \tag{6.4}$$

Then, the set $\mathcal{S}$ to build has to be as diverse as possible so as to capture the different term combinations that may be used to retrieve items. The diversity of an item $x$, or more precisely of its rewriting vector $RV_x$, w.r.t. to the set $\mathcal{S}$ of previously selected items, denoted by $Div_{\mathcal{S}}(x)$, represents the extent to which $RV_x$ is disjoint from the rewriting vectors of the items in $\mathcal{S}$.

$$Div_{\mathcal{S}}(x) = \frac{1}{|\mathcal{S}|} \sum_{x' \in \mathcal{S}} \mu_{disjoint}(RV_x, RV_{x'}), \tag{6.5}$$

where $\mu_{disjoint}(RV_x, RV_{x'})$ quantifies how much the two rewriting vectors are disjoint:

$$\mu_{disjoint}(RV_x, RV_{x'}) = \frac{1}{q} \sum_{A_i} [1 - \perp_{P_{i,j}} \top(\mu_{P_{i,j}}(x), \mu_{P_{i,j}}(x'))]. \tag{6.6}$$

**Remark 6.1** *In Equation* (6.6) *the t-conorm operator* $\perp$ *is applied over all the modalities* $P_{i,j}$ *of the partition* $\mathcal{P}_i$ *associated with the domain of the attribute* $A_i$*. For each partition we check whether there exists a modality that is not shared by* $x$ *and* $x'$*, and compute the average over the* $q$ *partitions.*

W.r.t. the current content of the set $\mathcal{S}$, a global score may be computed for each item $x \in \mathcal{D} \backslash \mathcal{S}$ based on the three previously defined notions:

$$Sc(x, \mathcal{S}) = \top(RepV(x), RepD(x), Div_{\mathcal{S}}(x)), \tag{6.7}$$

where the Lukasiewicz t-norm and t-conorm ($\top_{Luk}(x, y) = \max(0, x + y - 1)$ and $\perp_{Luk}(x, y) = \min(x + y, 1)$) have been used in our case to allow for some compensation between the combined degrees.

The algorithm (Algo. 10) used to build $\mathcal{S}$ consists in first identifying the item $x^1$ from $\mathcal{D}$ maximizing the score $Sc(x, \emptyset)$, an item is arbitrarily picked in case of tie, and then to incrementally complete this set with the next best item, and so on until the cardinality of $\mathcal{S}$ is equal to $k$. It is worth recalling that this costly algorithm, $k \times |\mathcal{D}|$ steps, that computes a locally optimal diversified set (depending on the choice of the first $x^1$), is performed offline and only once, unless significant changes on $\mathcal{D}$ have been done. The second question concerns the choice of the value for the parameter $k$. The main goal of the approach is to display a small set of items that can be quickly evaluated by the user. Thus, as no more than 20 or 30 items can be displayed simultaneously, we consider that setting $k$ to 100 is generally enough. Obviously, additional interesting items may be identified on-the-fly if $k$ items are not enough, at the cost of new scans of the data.

**Input:** data $\mathcal{D}$; $k$ the number of expected examples in $\mathcal{S}$
**Output:** set of diversified examples $\mathcal{S}$

```
1  begin
2      S ← ∅;
3      x^1 ← arg max_{x∈D}(Sc(x, ∅));
4      S ← {x^1};
5      while |S| < k do
6          maxSc ← 0;
7          x^1 ← NULL;
8          foreach x ∈ D\S do
9              tmpSc ← Sc(x, S);
10             if tmpSc > maxSc then
11                 maxSc ← tmpSc;
12                 x^1 ← x;
13             end
14         end
15         S ← {x^1} ∪ S;
16     end
17     return S;
18 end
```

**Algorithm 10:** Construction of $\mathcal{S}$

FQBE requires users to evaluate examples in order to infer their preferences. To clarify the different kinds of examples we provide the following definitions:

**Definition 6.2** *An* example *is an item from the set $\mathcal{S}$ to be evaluated by the user, obtained with Algorithm 10*

**Definition 6.3** *A* positive example *(resp. a* counter-example*) is an example evaluated positively (resp. negatively) by the user.*

FQBE consists of the following steps:

- The user chooses which examples are "good" or "bad" according to his/her expectations. They are put in a positive and a negative set respectively.
- The sets are characterized as in [Moreau et al., 2016a] so as to discover the modalities that represent them best.
- The positive set provides a conjunctive selection statement reflecting fuzzy properties desired by the user.
- The answers to this query are submitted to the user, along with explanations of the user preferences in a linguistic interpretable form.

After each example evaluation, the inferred query is updated and displayed. If they are satisfactory, the generated query is executed. Otherwise, the user can evaluate more examples to modify the inferred characterizations.

In the following we use a running example to illustrate the last three steps mentioned at the beginning of the section. For the sake of clarity we select our own examples and the associated evaluations. Table 6.1 contains the data elements used in this example with the associated (positive or negative) evaluations. Positive examples are put in the set $\mathcal{E}_+$ and counter-examples are put in the set $\mathcal{E}_-$. Items in this example are described according to the attributes: year, fuel consumption, mileage, option level, comfort level, security level, engine size, horse power, price and make.

## 6.3 Inferring User Preferences from Evaluations

The objective is to deduce the user's expectations from the evaluations of the suggested examples. We aim at finding which properties are satisfied by items of $\mathcal{E}_+$ and not by those of $\mathcal{E}_-$ (and vice-versa). These properties are called *characterizations*, as defined in Chapter 4.

**Definition 6.4** *A characterization $E_{\mathcal{E}}$ attached to a set $\mathcal{E}$ is a conjunction of couples (attribute, fuzzy set of labels) of the form*

$$E_{\mathcal{E}} = \{(A_i,\ F_i) \mid A_i \in \mathcal{A} \text{ and } F_i \text{ is a fuzzy set of linguistic labels from the partition of the domain of } A_i\},$$

*where the attributes $A_i$ in $E_{\mathcal{E}}$ form a subset of $\mathcal{A}$.*

The main difference with Chapter 4 regarding characterization concerns the set of attributes considered. In Chapter 4 characterizations contained attributed from $\mathcal{A}_\omega$, the set of attributes that were not in the query. However in the current chapter one of our objectives is to obtain a query, and we propose to do so by looking for characterizations over all the attributes available.

As in Chapter 4, items from the sets $\mathcal{E}_+$ and $\mathcal{E}_-$ are projected on the vocabulary in order to obtain a characterization of the sets using terms of the natural language, i.e.

terms from the vocabulary. The projection of a set $\mathcal{E}$ on the partition of an attribute $A_i \in \mathcal{A}$ is represented by a fuzzy set of labels $F_i = \{L_j^i / \mu_{L_j^i}(\mathcal{E}) \mid L_j^i \in \mathcal{P}_i\}$ where

$$\mu_{L_j^i}(\mathcal{E}) = \frac{\sum_{x \in \mathcal{E}} \mu_{L_j^i}(x)}{|\mathcal{E}|}, \tag{6.8}$$

and $\mu_{L_j^i}(x)$ is the degree of membership of $x$ to $L_j^i$. It is assumed that the only labels that appear in $F_i$ are such that $\mu_{L_j^i}(\mathcal{E}) > 0$. The degree associated with each label is related to the number of points verifying it and to their membership degrees, hence making characterizations representative of each set. The projection of the sets $\mathcal{E}_+$ and $\mathcal{E}_-$ on the modalities of the vocabulary leads to a table as the one illustrated in Table 6.2.

Table 6.2 – Projection of the sets on the vocabulary

| Set | year | mileage | price |
|---|---|---|---|
| $\mathcal{E}_+$ | very recent (1) | low (1) | medium (1) |
| $\mathcal{E}_-$ | recent (0.11)<br>very recent (0.78)<br>last model (0.11) | very low (0.44)<br>low (0.56) | expensive (0.54)<br>very expensive (0.46) |

With FQBE we currently focus on conjunctive queries. Asking users to evaluate examples with all possible conjunctive combinations of properties is unrealistic, so we infer a conjunctive condition from the most representative properties of each set of examples. Instead of browsing candidate characterizations of all sizes, we propose to look for mono-attribute characterizations to explain the sets of examples. A "good" characterization would convey the meaning that there exists at least one modality on the attribute domain over which the two sets are disjoint. Indeed, a mono-attribute characterization represents a disjunction of modalities, so for a characterization to be representative of a set there must exist a discriminating modality. We try to capture this idea with the notion of specificity. In order to measure the extent to which a characterization $CE_{\mathcal{E}+}$ explains the set of positive examples, we consider its counterpart $CE_{\mathcal{E}-}$ (with the same attributes as in $CE_{\mathcal{E}+}$, but with the labels corresponding to the elements of $\mathcal{E}-$, see Example 6.3 below), and compute its degree of specificity with the mutually exclusive disjunction between the fuzzy sets:

$$sp(CE_{\mathcal{E}+}, CE_{\mathcal{E}-}) = 1 - \max_{L_j^i \in \mathcal{P}_i} \min(\mu_{L_j^i}(\mathcal{E}_+),\ \mu_{L_j^i}(\mathcal{E}_-)), \tag{6.9}$$

where $\mathcal{P}_i$ the partition of the attribute $A_i$ (in the candidate characterizations $CE_{\mathcal{E}+}$ and $CE_{\mathcal{E}-}$) on the vocabulary, and $L_j^i \in \mathcal{P}_i$ the modalities covered by the characterizations.

**Example 6.3** *Let us consider a candidate characterization $CE_{\mathcal{E}+}$ for $\mathcal{E}_+$, such as* price is medium (1) *in Table 6.2. To check whether it is specific, we consider its counterpart $CE_{\mathcal{E}-}$ for $\mathcal{E}_-$ with the same attributes (we project the elements of $\mathcal{E}_-$ on the attributes of $CE_{\mathcal{E}+}$),* e.g. price is expensive (0.54) or very expensive (0.46) *according to Table 6.2.*

*Using the data in Table 6.2, we present some of the possible candidate characterizations for $\mathcal{E}_+$:*

- $CE_+^1$ = "year is very recent (1)"
- $CE_+^2$ = "mileage is low (1)"
- $CE_+^3$ = "price is medium (1)."

*Candidate characterizations for $\mathcal{E}_-$ include:*

- $CE_-^1$ = "year is recent (0.11) or very recent (0.78) or last model (0.11)"
- $CE_-^2$ = "mileage is very low (0.44) or low (0.56)"
- $CE_-^3$ = "price is expensive (0.54) or very exp. (0.46)."

*In this running example we consider candidate characterizations with only one attribute, and get:*

$$\begin{aligned} sp(CE_+^1,\, CE_-^1) &= 1 - \max(\min(0, 0.11), \min(1, 0.78), \min(0, 0.11)) \\ &= 1 - \max(0, 0.78, 0) \\ &= 0.22. \end{aligned}$$

*We obtain $sp(CE_+^2,\, CE_-^2) = 0.44$ and $sp(CE_+^3,\, CE_-^3) = 1$. As the label* very recent *is present in both $CE_+^1$ and $CE_-^1$ with a high degree, their specificity degree is low. Among these candidate characterizations, $CE^3$ is the best choice according to the specificity degree.*

*After reviewing all possible candidate characterizations, we order them by decreasing specificity degree and find the following characterizations for $\mathcal{E}_+$ (top-3):*

- $E_+^1$ ="engine size is small" *(specificity 1);*
- $E_+^2$ ="price is medium" *(specificity 1);*
- $E_+^3$ ="consumption is very low (0.13) or low (0.62) or medium (0.25)" *(specificity 0.78).*

*We find the following characterizations for $\mathcal{E}_-$ (top-3):*

- $E_-^1$ ="engine size is medium (0.28) or big (0.72)" *(specificity 1);*
- $E_-^2$ ="price is expensive (0.54) or very expensive (0.46)" *(specificity 1);*
- $E_-^3$ ="consumption is low (0.22) or medium (0.11) or high (0.67)" *(specificity 0.78).*$\diamond$

Only the characterizations with a high enough specificity degree (above a threshold $\lambda$) become a conjunct in the final selection condition.

**Remark 6.2** *If either $\mathcal{E}_+$ or $\mathcal{E}_-$ is empty then all candidate characterizations for the non-empty set are fully specific.*

## 6.4 Translating Preferences into Queries

From the characterizations of a set of positive examples $\mathcal{E}_+$ and those of a set of counter-examples $\mathcal{E}_-$ we seek to build a query that will look for elements similar (in terms of properties) to the positive examples.

### 6.4.1 Translating Preferences

The characterizations are first "refined," so as to render them coherent. A label present in the positive and negative characterizations may or may not be removed depending on its membership degree. We propose to apply the following rule.

**Translation rule**: *if the positive and the negative degrees are both greater or equal to 0.5, then they are both removed from the positive and negative characterizations. Otherwise, the label with the highest degree is kept.*

Characterizations with a specificity degree equal to 1 are not affected by this rule. Indeed, a specificity degree equal to 1 means that there are no labels shared between the positive and negative characterizations: their sets are fully disjoint.
In our running example, $E_-^3$ is concerned, the *low* modality is removed (0.62 is higher than 0.22) and we get:
$E_-^3$ =*"consumption is medium (0.11) or high (0.67)"*. The *medium* modality is also removed (0.25 is higher than 0.11), and as a result: $E_-^3$ =*"consumption is high (0.67)"*.

Finally we translate the characterizations into a conjunctive query, each characterization becoming a selection condition. The selection conditions are in Listing 6.1.

```
engine size is small
and price is medium
and consumption is (very low (0.13) or low (0.62)
   or medium (0.25))
```

Listing 6.1 – Conjunction of selection conditions

### 6.4.2 Weighted Disjunction

For some attributes one label does not fully cover the sets of examples or counter-examples: this leads to a disjunction of labels (such as *consumption is very low or low or medium*). These labels do not carry the same weight: for instance here the label *low* has the highest membership degree (0.62). We treat these degrees as importances that must be taken into account in the query. To differentiate labels we propose to use the weighted disjunction as proposed in [Dubois and Prade, 1986]. This weighted disjunction enables us to take into account the weights (here the membership degrees) assigned to each modality in the characterization, so that we obtain:

$$\mu_{A_i}(t) = \max_{P_{i,j} \in \mathcal{P}_i} \min(w_j, \mu_{P_{i,j}}(t)) \tag{6.10}$$

where $w_j$ denotes the weight associated with the modality $P_{i,j}$ related to attribute $A_i$ in the characterization used to generate the query.

**Example 6.4** *Let us consider the selection condition* "consumption is very low (0.13) or low (0.62) or medium (0.25)". *In order to use the weighted disjunction, the weights must first be normalized so that the maximum weight is equal to 1. The selection condition thus becomes* "consumption is very low (0.2) or low (1) or medium (0.4)". *When reviewing a tuple $t$ with a* fuel consumption *that is fully "low," we get:*

$$\begin{aligned}\mu_{\textit{fuel con.}}(t) &= \max(\min(w_{\textit{very low}}, \mu_{\textit{very low}}(t)), \min(w_{low}, \mu_{low}(t)),\\ &\qquad \min(w_{medium}, \mu_{medium}(t)))\\ &= \max(\min(0.2, 0), \min(1, 1), \min(0.4, 0))\\ &= 1,\end{aligned}$$

*which is the maximum possible obtainable value.*$\diamond$

Each selected characterization leads to a selection condition. By default we propose a conjunction of selection conditions, however a disjunction of conditions could also be considered: then, not all characterizations necessarily need be true for the user to be satisfied.

## 6.5 Explaining the Query and its Results

Query results are displayed to the user, along with (refined) characterizations. They enable the user to know what preferences were inferred from his/her evaluations. However these preferences may not satisfy the user, who may wish to improve the results by providing some feedback regarding these inferred preferences.

### 6.5.1 Explaining Inferred Preferences

Negative explanations can greatly improve readability if for any given attribute the positive explanation is a disjunction of most modalities. In this case only providing one or two modalities as the negative explanation is more cooperative than providing a disjunction covering almost the entire partition. For instance, the explanation "*not high* consumption" is more intelligible than the disjunction in $E^3_+$. In our running example, the user gets the following explanations:

In this example, the user should see the following explanation:
*We believe you have an interest in:*

- *small* engine size;
- *medium* price.
- *not high* consumption.

Another explanation format proposed is attribute-centered:
*We believe you have the following interests:*

- engine size: you are interested in *small* but not in *medium or big*;
- price: you are interested in *medium* but not in *expensive or very expensive*;
- fuel consumption: you are interested in *very low or low or medium* but not in *high*.

Several objectives must be met when formulating explanations:

- no contradiction: for any label an attribute can appear in both the examples characterization and in the counter-examples characterization. The explanations must not reflect this for the sake of clarity. The translation rule (Subsection 6.4.1) applied before the generation of the fuzzy query handles this issue.
- as readable as possible: negligible labels (with a very low membership degree) may be omitted, and important labels should be emphasized. When considering a disjunction of labels, they usually do not hold the same importance. In this example we have the explanation *"very low, low or medium* consumption (mostly low)". Pointing out "mostly low" enables the user to know that this attribute is the most important in the selection condition. In practice, the adverb *"mostly"* could be added to labels with a degree greater than 0.5, when relevant (when considering the characterization *low (0.51) or very low (0.49)*, then emphasizing the label *low* is irrelevant). Other adverbs, *e.g. "rarely,"* may be added to labels with a degree lower than 0.2 for instance — instead of omitting them.

Beyond the inferred preferences displayed, the results provided can also be explained. Indeed, the user may wish to know the extent to which the results match the inferred preferences, which can be provided by displaying membership degrees.

### 6.5.2 Interacting with the User to Refine Results

As inferred preferences may not effectively capture the user's desires, we designed an optional step to enable the user to provide a form of feedback to the system, based on the evaluation of these inferred preferences.

Inferred preferences may either be validated or discarded by the user:

- If some (at least one) preferences are validated: the query is executed;
- If all preferences are discarded: the example evaluation starts anew.

If the query results are not satisfactory in spite of the validated inferred preferences, this may be because the preferences are not precise enough, leading to too many results uninteresting to the user. Such a case may happen when:

- The vocabulary partitions are not adapted to the data (although we assume that they were created by an expert of the domain);
- The inferred preferences take the form of a disjunction of many labels, and cover a large part of the partition of the domain.

In the second case, evaluating more examples may help to refine the user's preferences.

## 6.6 Experiments

We study the effectiveness of FQBE using a real dataset of second-hand cars scraped from LeBonCoin.fr of cardinality 10k+, with 9 different numerical attributes that each have a predefined vocabulary. We set two objectives: (i) use *precision* to compare our example selection scoring method $Sc$ to a random selection, and (ii) determine how well we infer user preferences, and the impact of the specificity threshold $\lambda$ on *precision* and *recall*.

To evaluate the proposed approach, we consider the set of queries in Listing 6.2. These queries have been selected for the diversity of the vocabulary elements involved in their selection statement. For each such query $Q$ that we try to infer, we evaluate examples based on their belonging to the result set $R$ of $Q$.

Each tested fuzzy query $Q$ yields a fuzzy result set $R$. The *core* of $R$ is denoted by $R_c$ and contains only the fully satisfactory results of $R$. Its support, denoted by $R_s$ contains all elements from $R$ with a non-zero degree. With FQBE, by evaluating examples we generate a query $Q'$ that yields a result set $R'$. To evaluate the interest of the generated query $Q'$, we compare its result set $R'$ to the original result set $R$. Obtaining the fuzzy set equality $R = R'$ is not our primary objective here. Our first objective is precision: finding only "good" results. Only afterwards we shall focus on recall: finding all "good" results. We compute the degree of inclusion of $R'$ in $R$ with $\mu_{\subseteq}(R', R)$, in order to know how "good" the generated results are. This is a fuzzy interpretation of the *precision* in information retrieval. Similarly, we compute the inclusion of $R$ in $R'$ with $\mu_{\subseteq}(R, R')$ to obtain the *recall*.

$$\text{Precision} = \mu_{\subseteq}(R', R) = \frac{\sum_{x \in R'} \min(\mu_R(x), \mu_{R'}(x))}{\sum_{x \in R'} \mu_{R'}(x)}. \tag{6.11}$$

```
Q1: (price is medium) and (mileage is low)
Q2: (price is expensive or very expensive) and (year is last model)
Q3: (consumption is low or very low) and (mileage is medium)
    and (year is recent or very recent)
Q4: (price is medium) and (consumption is low)
    and (mileage is very low) and (year is recent)
Q5: (price is cheap) and (mileage is low)
Q6: (consumption is low) and (year is very recent)
Q7: (mileage is very low) and (horse power is very high)
Q8: (price is cheap) and (horse power is very high)
Q9: (mileage is medium) and (consumption is very low)
Q10: (year is medium) and (price is cheap) and (mileage is medium)
```

Listing 6.2 – Conjonctions of selection conditions

### 6.6.1 Comparison of Example Selection Methods

We compare two sets of examples: a random selection, and $\mathcal{S}$, the set of examples *selected* with $Sc$. In both cases we consider a set of 150 examples to evaluate (let us recall

that $\mathcal{S}$ is ordered). For each query $Q$ considered, we positively evaluate the examples that match it (that are in $R$), and negatively evaluate the others. Characterizations found such that $sp \geqslant 0.7$ are kept. In Table 6.3 we specify the size of the result set $R$ (the cardinality $|R_s|$ of its support) of each query $Q$ considered, as well as the number of results $|R_c|$ fully satisfying $Q$. We present the number of consecutive examples to evaluate to attain certain precision values (0.5 and 0.9) between the set of generated results and that of original results. We only focus on precision ($R' \subseteq R$) as we aim to find original results only, not all original results.

In Table 6.3 there is only one line for $Sc$ as the 0.9 precision value is always attained with the same number of examples necessary to attain the 0.5 precision value. Random scores are obtained by averaging their results over 10 runs, and using the number 151 when no minimal number of examples has been found to attain the expected precision values (*e.g.* if for $Q1$ the precision 0.9 is attained by evaluating 49 examples on the first run, and is never attained by evaluating all examples on the second run, the average over these two runs is $(49 + 151)/2 = 100$). Nevertheless, the average numbers over 10 selections of randomly-chosen examples are far greater than those of $Sc$. In Table 6.3 we also show the ratio between the minimal number of random examples and the minimal number of examples from $\mathcal{S}$ to attain a precision of 0.9. On average, this ratio is of 15.2, showing that $\mathcal{S}$ is more efficient at building $Q'$. Only $Q4$ is not inferred: R4 may be too small compared to the dataset.

Table 6.3 – Result sets sizes, and inclusion degrees with Random and Sc selection methods

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|R_c\|$ | 685 | 156 | 536 | 3 | 29 | 462 | 458 | 175 | 116 | 385 |
| $\|R_s\|$ | 1312 | 157 | 714 | 47 | 72 | 654 | 912 | 382 | 459 | 477 |
| Random | | | | | | | | | | |
| Precision > 0.5 | 27.7 | 71.6 | 53.1 | +150 | 139.2 | 27.5 | 15.2 | 51.7 | 121.1 | 57.9 |
| Precision > 0.9 | 86.5 | 84.5 | 121.6 | +150 | 139.2 | 51.4 | 34.5 | 59 | 121.4 | 84.1 |
| $Sc$ | | | | | | | | | | |
| Precision > 0.9 | 22 | 2 | 14 | +150 | 3 | 3 | 2 | 17 | 14 | 26 |
| Ratio @ 0.9 | 3.93 | 42.25 | 8.69 | 1 | 46.4 | 17.13 | 17,25 | 3.47 | 8.67 | 3.23 |

### 6.6.2 Impact of the Specificity Threshold $\lambda$ on Precision and Recall

We now evaluate the precision and recall over examples selected with $Sc$ only. In Table 6.4 we present the cardinalities of the generated sets of answers for the 10 inferred queries considered (which can be compared to the cardinalities of the original sets of answers from Table 6.3), as well as the precision and recall for three different specificity threshold values $\lambda$. We positively evaluate examples from $\mathcal{S}$ that are in $R$, and negatively evaluate the others. $\mathcal{S}(R)$ is the set of examples from $\mathcal{S}$ that are also in $R$.

With a specificity threshold of 0.7, the generated queries do not yield many fully satisfactory results. While the precision is excellent (maximal for all queries but $Q4$ and $Q8$), the recall is very low, as are the numbers of generated results. The generated queries all have a very high number of selection conditions (see Table 6.5), almost one

Table 6.4 – Number of elements from $S$ in $R$, and cardinalities of result sets

| Q | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert S(R_c)\rvert$ | 5 | 19 | 5 | 0 | 6 | 4 | 19 | 4 | 7 | 6 |
| $\lvert S(R)\rvert$ | 6 | 19 | 6 | 0 | 6 | 4 | 21 | 4 | 8 | 6 |
| $\lambda = 0.7$ | | | | | | | | | | |
| $\lvert R'_c\rvert$ | 4 | 3 | 74 | 3575 | 3 | 1 | 1 | 26 | 2 | 1 |
| $\lvert R'_s\rvert$ | 21 | 38 | 225 | 10479 | 4 | 8 | 43 | 119 | 27 | 12 |
| Precision | 1 | 1 | 1 | | 1 | 1 | 1 | 0,192 | 1 | 1 |
| Recall | 0,01 | 0,05 | 0,22 | | 0,09 | 0,01 | 0,02 | 0,03 | 0,05 | 0,02 |
| $\lambda = 0.75$ | | | | | | | | | | |
| $\lvert R'_c\rvert$ | 42 | 36 | 357 | 3575 | 8 | 2 | 211 | 605 | 20 | 5 |
| $\lvert R'_s\rvert$ | 167 | 98 | 644 | 10479 | 27 | 32 | 523 | 931 | 116 | 29 |
| Precision | 1 | 1 | 1 | | 1 | 1 | 0,88 | 0,18 | 1 | 1 |
| Recall | 0,07 | 0,47 | 0,79 | | 0,33 | 0,02 | 0,5 | 0,49 | 0,25 | 0,04 |
| $\lambda = 0.8$ | | | | | | | | | | |
| $\lvert R'_c\rvert$ | 685 | 124 | 357 | 3575 | 24 | 1232 | 287 | 1458 | 97 | 22 |
| $\lvert R'_s\rvert$ | 1312 | 157 | 644 | 10479 | 64 | 1952 | 588 | 1729 | 407 | 91 |
| Precision | 1 | 1 | 1 | | 1 | 0,31 | 0,83 | 0,17 | 1 | 1 |
| Recall | 1 | 0,96 | 0,79 | | 0,88 | 0,87 | 0,55 | 0,92 | 0,9 | 0,12 |

for every attribute (out of 9).

With a specificity threshold of 0.75, we obtain the maximal precision for queries $Q1, Q2, Q3, Q5, Q6, Q9$, and $Q10$. In other words, all fully satisfactory results from $R1'$ also fully satisfy $R1$, and the top-42 (unordered) results between these two sets are the same. However we are still far from covering the whole $R1$, although it is already a good improvement from finding only 4 elements with $\lambda = 0.7$. On average, the generated queries cover half the attributes of the dataset.

With the specificity threshold $\lambda = 0.8$, we obtain the maximal precision for queries $Q1, Q2, Q3, Q5, Q9$, and $Q10$. In other words, all fully satisfactory results from $R1'$ also fully satisfy $R1$, and the top-685 (unordered) results between these two sets are the same. Furthermore, we obtain $R1_s = R1'_s$. Furthermore, the recall for $R1$ is also maximal: we have $R1 = R1'$. This is a major improvement in terms of result set size, although the equality was not found in as many cases as before. The queries obtained with $\lambda = 0.8$ are described in Listing 6.3. On average the queries only cover a few attributes of the dataset (between 2 and 4).

```
Q1‘ (price is medium (1.0)) and (mileage is low (1.0))
Q2‘ (year is last_models (1.0)) and (price is expensive (0.462)
    or very_expensive (1.0))
Q3‘ (year is recent (1.0)) and (price is medium (1.0)
    or expensive (0.25)) and (mileage is medium (1.0))
    and (consumption is very_low (0.5) or low (1.0))
Q4‘ true
Q5‘ (price is cheap (1.0)) and (mileage is low (1.0))
    and (year is medium (1.0) or recent (1.0))
Q6‘ (price is very_cheap (0.5) or medium (1.0)
    or expensive (0.5)) and (consumption is low (1.0))
Q7‘ (price is very_expensive (1.0) or excessively_expensive (1.0))
    and (consumption is high (0.666) or very_high (1.0))
```

```
    and (mileage is very_low (1.0))
Q8‘ (price is cheap (1.0)) and (year is very_old (0.5) or old (0.5)
    or medium (1.0))
Q9‘ (consumption is very_low (1.0)) and (mileage is medium (1.0))
    and (price is very_cheap (0.429) or cheap (1.0) or medium (0.476))
    and (year is medium (1.0) or recent (0.4))
Q10‘ (consumption is very_low (1.0)) and (price is cheap (1.0))
    and (mileage is medium (1.0)) and (year is medium (1.0))
```

Listing 6.3 – Conjunctions of selection conditions obtained

The differences between recall values lie on the specificity threshold. We propose to check how many characterizations are kept for each query and for different specificity thresholds in Table 6.5. With 0.7, a query is generated over almost all attributes, with 0.75 over half the attributes and with 0.8 over one third of the attributes. This may explain the earlier gaps between result set sizes in Table 6.4. A high specificity threshold will limit the number of characterizations considered to generate the query, and in turn less selection conditions will increase the size of the result set. Let us note that is it crucial to find *some* characterizations for the approach to work, and that should there be no characterization with a specificity degree above a given $\lambda$, then the top-k characterizations should be considered instead (with $k = 3$ for instance). $\lambda$ values higher than 0.8 become too restrictive.

Table 6.5 – Number of characterizations obtained for the positive set of examples with different specificity thresholds

| $\lambda$ | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.7 | 8 | 9 | 8 | 0 | 9 | 8 | 8 | 8 | 8 | 8 |
| 0.75 | 5 | 5 | 4 | 0 | 5 | 4 | 5 | 4 | 5 | 5 |
| 0.8 | 2 | 2 | 4 | 0 | 3 | 2 | 3 | 2 | 4 | 4 |

Let us observe that with queries that have at least one selection condition in the form of a disjunction of labels — such as $Q2$: price is expensive or very expensive — one label in the disjunction will be advantaged (here *very expensive*) in comparison to the other as a consequence of the weighted disjunction. This explains why we have $|R2'_c| = 124$: the remaining elements feature the label that is "disadvantaged" are thus do not fully satisfy the generated query, despite all the correct labels having been found in $Q2'$.

## 6.7 Discussion

In this Section we discuss the main differences between FQBE and other approaches that belong to the QBE paradigm.

### 6.7.1 User-Example-based Approaches

In some cases it is not easy to specify exactly what we are looking for with a query, and yet have an example of it. In graph databases it is possible to provide some systems with a graph to find others that share the same patterns. Examples include Exemplar Search [Mottin et al., 2014a] that takes as input user-provided graphs.

Graph Query by Example (GQBE) [Jayaram et al., 2016] enables users to find graph patterns based on input tuples, and not on graphs. Only providing tuples enables the user to not have to specify the relation between the instances in the tuples. For example, if a user inputs the tuple ⟨Jerry Yang, Yahoo!⟩, then answers such as ⟨David Filo, Yahoo!⟩ and ⟨Sergei Brin, Google⟩ are returned, without ever providing the relation (in this case, company founder) between the elements in the tuple given as input.

Recent industrial implementations of QBE include "Search by Ideal Candidate" [Ha-Thuc et al., 2016] at LinkedIn, which offers recruitment services to companies. The authors of [Ha-Thuc et al., 2016] aim to help HR recruit the best candidate by helping them to create a personalized query using the profile of ideal candidates, such as those of people already working in the company. Based on these profiles, some of their attributes (e.g. skills) are retrieved and compiled into a query targeting candidates with similar profiles. They look for the skills with the highest *expertise scores* among ideal candidates to rank the skills to be found in candidates. The components of the query are graphically featured so the user can easily modify the query. The adequacy of potential candidate is measured twofold: (i) by estimating how this candidate is relevant to the query obtained and (ii) by measuring the similarity between this candidate and the input candidates. Over time, the ranking algorithm gives more importance to the relevance of the query results than to the similarity with the ideal profiles originally selected. The authors assume that the iterative refining of the query will return more interesting results to the user.

### 6.7.2 Prototypical-Example-based Approaches

A variant of QBE using uncertainty is presented in [Tatemura et al., 2008], to reflect the user's ignorance of the combined schema of multiple sources. The user provides tuples of interest to them and chooses the components of the output schema. The system has to match the different data sources and provide the user with queries that satisfy their requirements. The user then helps the system to refine the query by evaluating the presented results.

In [De Calmès et al., 2003] the authors present a fuzzy logic-based QBE approach. They ask the user to review (on a scale, positively or negatively) prototypical examples from the database — which may be fictitious, as long as they reflect the contrast between the attribute values of the items in the database. Based on these evaluations, a query is generated to look up items that are "similar to at least one example (w.r.t. all the attributes) and which are dissimilar to all counter-examples (each time w.r.t. at least one attribute)." Each example and counter-example can be weighted and the same goes for the different attributes of the database, in order to obtain answers as representative

as possible of the user's choices. The counter-examples have a limited influence on the result: they intervene only when the user has not selected any positive examples, or has made incoherent choices when reviewing examples.

The authors of [Zadrozny et al., 2010] present an approach inspired from QBE. Assuming that some users do not necessarily know how to browse a database system, they are presented with items from the database and asked to classify them as positive or negative. The evaluation can be global — on the whole item — and local — attribute by attribute. They use the $k$-nearest neighbors algorithm to find the next examples to evaluate based on those freshly evaluated by the user. A model of the user's profile is derived from their choices, so the user can check whether the system correctly interprets the user's choices. This profile is obtained by determining which linguistic labels are relevant to the user — computing how many times each label was in an example positively evaluated. The computation of this user profile is done as an entire different step from the computation of next examples to evaluate, although the authors believe that the profile computation could also provide the next examples to evaluate.

Unlike most papers cited above, this work focuses on an evaluation-based QBE approach. We do not need users to have the exact idea of what they are looking for in mind; instead we ask them to evaluate (positively or negatively) examples of our choosing. We highlight the differences between the works of [De Calmès et al., 2003], [Zadrozny et al., 2010] and ours on three points: the selection of examples, their evaluation, and the use of these evaluations.

**On the Selection of Prototypical Examples** The authors of [De Calmès et al., 2003] suggest using: (i) actual items that are representative of categories, or (ii) items that feature diversified values for the attributes the user is interested in, or (iii) items generated by the user. The authors of [Zadrozny et al., 2010] propose to resort to a random or partially random selection of examples, either using some information on the user if available, or representing predefined categories of data. Also, they take into account previous evaluations to select new examples with a partitioning method. In both [De Calmès et al., 2003] and [Zadrozny et al., 2010] the authors do not specify how to obtain examples representative of (categories of) the dataset. In this work we propose to use the vocabulary describing the data as well as the representativity of the data to select examples to evaluate.

**On the Evaluation of Prototypical Examples** The authors of [De Calmès et al., 2003] propose to give an importance to attributes, and to evaluate the representativity of an example on a scale. The authors of [Zadrozny et al., 2010] propose to evaluate each example value for each attribute on a scale, as well as to give a global evaluation to each example. In this work we ask users to simply give a binary global evaluation to each example: no attribute preferences, nor any single attribute value preferences. Thus we do not require users to have an understanding of the structure of the database, and try to keep the number of interactions with the system reasonable.

**On Matching Evaluations, Results, and User Preferences** The authors of [De Calmès et al., 2003] check whether items in the database are "similar to at least one example (w.r.t. all the attributes) and which are dissimilar to all counter-examples (each time w.r.t. at least one attribute)." They do not infer user preferences — with linguistic labels — but provide items that match the user's preferences, according to a measure of their own. For each database item $x$, this measure is computed with (i) the similarity degree between positive examples and $x$, and (ii) the dissimilarity degree between counter-examples and $x$, on all attribute domains. The authors of [Zadrozny et al., 2010] provide items to evaluate until the user is satisfied, by selecting new items to evaluate with the k-NN algorithm. The discovery of user preferences — to obtain new examples — is done in an "extensional way" through the selection of desirable items. The reconstruction of user preferences — to keep the user aware of the preference elicitation process — however is done in an "intensional way" by describing the liked items. Global positive and negative evaluations are used to find association rules determining whether some linguistic terms are relevant to the user. Then for each attribute, at best one linguistic term is selected to express the user preferences for this attribute, only if the support, confidence and lift for the association rule that found it are "high enough." The authors note that they could also use this preference reconstruction method to select new examples to evaluate. They generate a fuzzy query based on user preferences reconstructed but never run it: they only use it for explanation purposes. In this work we look for common *properties* between positive examples on the one hand, and common *properties* between counter-examples on the other hand to infer user preferences and generate a query matching those. Unlike the approach described in [De Calmès et al., 2003], we do not use similarity relations to infer user preferences. Also, we use the inferred preferences to compute a fuzzy query and present its results, unlike the approach in [Zadrozny et al., 2010] which resorts to an iterative evaluation of examples until the user is satisfied (without ever evaluating a fuzzy query).

### 6.7.3 On Finer Evaluations by the User

With FQBE we only ask users to provide a binary evaluation of examples. A finer evaluation, *e.g.* on a [-5 ; 5] scale, is also possible. This leads to several changes in the different steps of FQBE:

- Example representation;
- Example sets projection on attribute partitions.

FQBE considers two sets of examples: the set of positive examples $\mathcal{E}_+$ and the set of counter-examples $\mathcal{E}_-$. By considering finer evaluations, each evaluation $ev$ is associated with its example $ex$ and they form a couple denoted $(ex, ev)$. As a consequence, examples $ex$ with an evaluation $ev$ higher than 0 are added to $\mathcal{E}_+$ and examples with an evaluation lower than 0 are added to $\mathcal{E}_-$.

**Remark 6.3** *We consider that evaluating an example with 0 is neutral, and is the same as not evaluating it. We do not leverage this information.*

The projection of an example set $\mathcal{E}$ on attribute partitions (Equation (6.8)) must take into account the evaluation value $ev$ associated with each example $ex \in \mathcal{E}$. It may be taken into account *e.g.* with a weighted average (Equation (6.12)) or with a dynamic average with the OWA operator (Equation (6.13)):

$$\mu_{L_j^i}(\mathcal{E}) = \frac{\sum_{(ex,ev)\in\mathcal{E}} \mu_{L_j^i}(ex) * ev}{\sum_{(ex,ev)\in\mathcal{E}} ev}, \tag{6.12}$$

$$\mu_{L_j^i}(\mathcal{E}) = \frac{\sum_{(ex,ev)\in\mathcal{E}} \mu_{L_j^i}(ex') * ev}{\sum_{(ex,ev)\in\mathcal{E}} ev}, \tag{6.13}$$

where the $\mu_{L_j^i}(ex')$ values correspond to the $\mu_{L_j^i}(ex)$ values sorted in ascending order.

## 6.8 Summary

In this chapter we presented FQBE, a Query by Example approach to help users browse databases by simply evaluating examples. We deemed that communication was the key, and aimed to provide users with explanations every step of the way. Indeed, FQBE is transparent : the user *knows* what the system "*believes*" about him/her. FQBE elicits user preferences based on evaluations: these user preferences are explained to the user, using terms from a predefined vocabulary in order to improve the interpretability of these explanations.

With FQBE, we proposed a method to select which examples to submit for user evaluation, which provides better results than a random selection of examples. We also showed that for most tested queries, FQBE is capable of inferring the user preferences and returning only original satisfactory results. These results are encouraging, and call for more selection methods in order to improve the examples submitted for user evaluation.

Perspectives include conducting experiments on other real-world datasets, as well as taking into account user feedback to alter the inferred user preferences. Reconsidering the conjunctive nature of the query generated with FQBE may also help to capture more complex preferences, which may be captured with a disjunctive normal form for instance, or less restrictive aggregation functions, such as quantifiers or weighted sums. We also intend to conduct a user study so as to evaluate the benefits of FQBE.

# Chapter 7

# Conclusion

The cooperative nature of database systems is very questionable: most of them are merely equipped to simply and directly answer queries, without ever taking into consideration the human nature of end-users. However, simple and direct answers are responsible for many questionable results to the users, *e.g.* be it plethoric answer sets or unexpected results. Diverse approaches enabling users to better understand the "behaviors" of the systems have been put forward in the past. However, few to none actually focus on both robustness and interpretability to help users grasp the logic behind the computation of their query results. In this thesis, we presented three situations in which explanations greatly contribute to making database systems more cooperative.

## Summary

In Chapter 4, we addressed the situation where users face overwhelming results and wish to get an overview, by understanding the structure of the answer set for instance. We proposed a crisp and a fuzzy method to characterize subsets of answers to database queries, by combining the use of clustering algorithms — to gather results — along with a fuzzy vocabulary — adapted to the data — to describe subsets of answers with terms from the natural language. The subsequent objective was to highlight the particular properties of each subset of answers, by formulating characterizations with terms from the fuzzy vocabulary. While robustness cannot always be guaranteed (it depends on the quality and the existence of characterizations to be found), the interpretability was obtained by the use of a (personalized) fuzzy vocabulary.

In Chapter 5, we proposed an original recommendation approach based on the use of typicality, along with its explanations. We proposed a solution to the *cold start* problem, as well as other solutions requiring previous ratings, making the combination of these solutions flexible and robust to all recommendations cases. The formulation of explanations was made possible by the typicality-based associations computing the recommendations, and contributed to helping users understand the reasoning behind the recommendations they obtained.

In Chapter 6, we proposed a cooperative adaptation of the QBE paradigm to help

users browse databases by simply evaluating examples. This query-building paradigm infers user preferences from evaluations, which are used to both retrieve results of interest and to explain these preferences to users. We proposed a method to select which examples should be submitted for user evaluation, which would provide better results than a random selection of examples. We also showed that for most tested queries, FQBE is capable of inferring the user preferences and returning only original satisfactory results. Moreover, the inferred preferences are formulated with a fuzzy vocabulary, guaranteeing the interpretability of these preferences, and thus providing explanations for the results.

## Perspectives

In the previous chapters we pointed out some limits in our work that would need investigating. In the case of the explanation of clusters of answers, perspectives include:

- Modifying the writing of characterizations in order to better fit some clusters that are not necessarily of an elliptic shape (with a disjunctive normal form, in the case of two identifiable subsets in a cluster for instance: *(price is excessive and year is last model) or (price is very expensive and year is very recent)* corresponds to two subsets, which as a whole would be conjunctively characterized by *(price is excessive or very expensive) and (year is last model or very recent)*, which is less informative), leading to the generation of characterizations more complex yet more precise;

- Suggesting pruning strategies to enable finding characterizations in reasonable time when there are many attributes available (by limiting the number of characterizations to return for instance).

The absence of any benchmark to compare demographic recommenders led us to compare our approach to state-of-the-art approaches from other recommender categories. Perspectives for our recommender include:

- Improving the MAE, RMSE, and precision metrics, (by considering more precise ratings in the computation of recommendations);

- Combining our approach with state-of-the-art approaches to leverage the advantages of each of them;

- Extending the principle of the recommendation approach to data sources other than demographics, *e.g.* contextual data (including localization, time, current activity, etc.).

Perspectives for Fuzzy Query By Example include:

- Taking into account user feedback to dynamically update inferred user preferences, by setting aside misleading preferences and proposing to evaluate more examples

that would focus on eliciting the actual user preferences (*e.g.* on the same attributes that were inferred in the misleading preferences), or by enabling the user to point out that he/she is not interested in preferences related to this attribute in particular for instance;

- Comparing different user preferences combination methods when building the query (*e.g.* conjunctively or disjunctively);
- Extend the approach to non-binary evaluations, and possibly propose to detail evaluations (*e.g.* attribute by attribute). Evaluating exemples on a [-5 ; 5] scale for instance would lead to altering the projection of sets of examples on the partitions from the vocabulary.

The approaches presented in this thesis are restricted to the context of relational databases. All of these approaches can be extended and improved with semantic data. The reasoning capabilities of RDF enable knowledge to be generated more easily out of data than using relational databases. We propose to discuss below possible future research directions, as well as some recent works related to cooperative answering over RDF data, classified according to three axes:

- Existing tools to facilitate the navigation of users in RDF databases, as well as taking into account their preferences;
- Helping users with unsatisfactory results;
- The computation of additional answers in RDF databases.

### Browsing Assistance

The most widespread query language for RDF data is SPARQL. Similar to SQL, it is not very user-friendly, if not the opposite. To simplify its use — or to avoid manipulating SPARQL — browsing alternatives have been proposed, such as faceted search guidance [Ferré, 2016], keyword query construction [Smits et al., 2014b, Smits et al., 2015], or user-interaction-based keyword query building [Dramé et al., 2015]. Users new to SPARQL can also check their queries by translating them into natural language [Ngonga Ngomo et al., 2013, Ell et al., 2014], with varying degrees of success. To enable users to successfully browse RDF databases, mechanisms such as query reformulation, summaries, or query by example have been proposed.

In Chapter 6 we proposed Fuzzy Query By Example, removing the need for users to be familiar with a query language in order to obtain results in the case of relational databases. This approach could be extended to other database models, provided that we found a way to adapt the approach while leveraging the benefits of other models. The underlying challenges would include:

- Determining an example selection method to find representative items in a given dataset. Our current approach considers representativity w.r.t. the vocabulary as well as w.r.t. the data. The major difference between relational and RDF

databases lies in the representation of the data, between a tabular representation and a triple representation. An RDF triple $\langle s, p, o \rangle$ denotes that the subject $s$ has a property $p$ with a value $o$. An example to evaluate would be a set of triples (*i.e. a subgraph*), where subjects, properties and objects would contribute — each in their own way — to the representativeness of the example in the RDF dataset. For instance it may be interesting to use resources that have many links in the graph, as well as properties that are associated with many resources, along with their most representative object values.

- Leveraging the class hierarchy of the ontology associated with the graph to tune the inferred user preferences: as some classes may be too precise and lead to answers too specific, "relaxing" some of the user preferences would be possible for the properties for instance (considering a property higher in the schema). The issue of failing queries in relation to user preferences has been addressed in [Dolog et al., 2009], where the authors leverage user preferences to relax failing user queries. While they address the issue of eliciting domain preferences, they do not address the elicitation of user preferences, and instead require users to provide these as additional input along with their queries.

In Section 3.2 we classified Query By Examples approaches in two categories: input-based ones as well as evaluation-based ones. There already exist approches inspired from the Query By Example paradigm in the context of graph databases, such as Exemplar Search [Mottin et al., 2014a, Mottin et al., 2014b] that takes as input user-provided graphs. Another framework allowing the search by graph pattern is NESS [Khan et al., 2011]. More generally, Graph Query by Example (GQBE) [Jayaram et al., 2014, Jayaram et al., 2016] enables users to find graph patterns based on input tuples, and not on graphs. This enables the user to not have to specify the relation between the instances in the tuples. For example, if a user inputs the tuple <Jerry Yang, Yahoo!>, then answers such as <David Filo, Yahoo!> and <Sergei Brin, Google> are returned, without ever providing the relation (in this case, company founder) between the elements in the tuple given as input. All these approaches were designed in the spirit of the "input-based" QBE techniques presented earlier in Section 3.2. However to the best of our knowledge no evaluation-based QBE approach like ours has been proposed in the RDF model.

### Facing Unsatisfactory Results

Unsatisfactory results include the empty answer, the plethoric answer, as well as the missing answer problems. Several works have already tackled the empty answer problem as well as the missing answer problem, however not many studies have been devoted to handling the plethoric answer problem. Nevertheless some data-exploration techniques (*e.g.* summaries) may be used instead to address this problem.

In Chapter 4 we considered the issue of plethoric answer sets, which we addressed by helping the user to understand the structure of the answer set. We proposed an approach in three steps, namely detection, description, and characterization. The objective of

helping users to understand their query results may very well be transposed to the RDF model. However this raises some new challenges w.r.t. this model:

- The clustering of the data was done over attributes in the relational setting. While simply doing it over the object values taken by the properties of resources is possible, it may also be interesting to leverage the links between resources. Let us note that finding subsets of answers should not require any previous knowledge of partitioning algorithms from the users, nor require them to specify parameters (*e.g.* number of clusters expected).

- Enabling users to use their own personalized vocabulary (defined with ReqFlex [Smits et al., 2013] for instance) to formulate queries is possible with FURQL [Pivert et al., 2016], however the terms from the vocabulary must be defined in the query.

Summaries may also be considered to tackle the plethoric answer problem. Graph summaries have several natures: they can be query-oriented [Cebiric et al., 2015] (and be queried as RDF graphs), or resource-oriented [Khatchadourian and Consens, 2010] (providing information on the distribution of classes and properties, and providing a summary with unlabeled edges). The empty answer problem has been addressed in the RDF context with approaches similar to those used in a relational context, most especially relaxation methods [Hurtado et al., 2006, Huang et al., 2008, Hadjali et al., 2015]. Another challenge in this direction consists in detecting whether a query is likely to yield an empty answer without having to run the query. Some relaxation methods require an intensive querying of the database to identify faulty conditions, while it may be possible to take advantage of the schema to determine whether a query is "suspicious" (*i.e.* is likely to fail).

### Additional Answers

Offering users additional results to their queries such as recommendations may be welcome when users have trouble manipulating the system or seizing its contents. The issue of improving and encouraging the usability of recommenders in database systems has been met with recommender languages, including REQUEST in relational databases [Adomavicius et al., 2011], and RecSparql as a recommendation language for SPARQL queries in RDF databases [Arrascue Ayala et al., 2014].

In Chapter 5 we designed an approach to propose recommendations, leveraging associations between entities using demographic data. Aside from computing recommendations, we also provided explanations to enlighten users as to why each recommendation is proposed. Enabling users to precisely understand the reasoning behind each recommendation greatly contributes to pointing out the relevance of the proposed recommendations. Our recommendations are based on the relational model and use some of its properties. By representing the recommendation data with RDF triples and by using its associated schema instead, it is possible to leverage the semantics of the RDF model:

- In a relational context we used the primary key–foreign key relationships to determine the links between entities, define similarity criteria, and formulate explanations. With the RDF model it is possible to find paths between resources (see *e.g.* the DBpedia Relationship Finder [Lehmann et al., 2007]), and to evaluate this distance (with the *Linked Data Semantic Distance* from [Passant, 2010b] for instance).
- The choice of similarity criteria may be easier to formalize with the RDF model, by simply using the properties of triples, or more complex paths in the graph.

Recommendations based on Linked Data and RDF data started about 10 years ago with domain-centric works such as Foafing the Music [Celma, 2008], followed with dbrec [Passant, 2010a] in music, or [Peska and Vojtas, 2013] with books, or [Di Noia et al., 2012] with movies. Computing similarity with Linked Data requires similarity measures between RDF ressources. These can be semantic — based on the links — or more classical — based on the property values, as with content-based systems.

As we can see, there is a great number of interesting research directions to explore, and this work, we hope, should contribute to opening new possibilities to enrich the new generation of database management systems with "intelligent" functionalities that bring them ever closer to the user, which also resonates with the now ever-growing issue of explainable and interpretable AI.

# Bibliography

[IEC, 2017] (2017). 24765-2017 - ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. Technical report.

[Adomavicius and Tuzhilin, 2001] Adomavicius, G. and Tuzhilin, A. (2001). Multidimensional Recommender Systems: A Data Warehousing Approach. *Electronic Commerce*, 2232:180–192.

[Adomavicius et al., 2011] Adomavicius, G., Tuzhilin, A., and Zheng, R. (2011). REQUEST: A query language for customizing recommendations. *Information Systems Research*, 22(1):99–117.

[Agrawal et al., 2009] Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. (2009). Diversifying search results. *Proceedings of the Second ACM International Conference on Web Search and Data Mining - WSDM '09*, page 5.

[Agrawal et al., 2002] Agrawal, S., Chaudhuri, S., and Das, G. (2002). DBXplorer: a system for keyword-based search over relational databases. In *Proceedings 18th International Conference on Data Engineering*, pages 5–16. IEEE Comput. Soc.

[Agrawal et al., 2003] Agrawal, S., Chaudhuri, S., Das, G., and Gionis, A. (2003). Automated Ranking of Database Query Results. *Innovative Data Systems Research (CIDR) Conference*, 31(1).

[Arrascue Ayala et al., 2014] Arrascue Ayala, V. A., Przyjaciel-Zablocki, M., Hornung, T., Schätzle, A., and Lausen, G. (2014). Extending SPARQL for Recommendations. In *Proceedings of Semantic Web Information Management on Semantic Web Information Management*, pages 1–8. ACM.

[Bhalotia et al., 2002] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *Proceedings 18th International Conference on Data Engineering*, pages 431–440. IEEE Comput. Soc.

[Borzsony et al., 2001] Borzsony, S., Kossmann, D., and Stocker, K. (2001). The Skyline operator. In *Proceedings 17th International Conference on Data Engineering*, pages 421–430. IEEE Comput. Soc.

[Bosc et al., 2008] Bosc, P., Hadjali, A., and Pivert, O. (2008). Empty versus overabundant answers to flexible relational queries. *Fuzzy Sets and Systems*, 159(12):1450–1467.

[Bosc et al., 2010] Bosc, P., Hadjali, A., Pivert, O., and Smits, G. (2010). On the use of fuzzy cardinalities for reducing plethoric answers to fuzzy queries. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6379 LNAI, pages 98–111.

[Bosc et al., 1995] Bosc, P., Lietard, L., and Pivert, O. (1995). Quantified Statements and Database Fuzzy Querying. In Bosc, P. and Kacprzyk, J., editors, *Fuzziness in Database Management Systems*, pages 275–308, Heidelberg. Physica-Verlag HD.

[Bosc and Pivert, 1995] Bosc, P. and Pivert, O. (1995). SQLf: A Relational Database Language for Fuzzy Querying. *IEEE Transactions on Fuzzy Systems*, 3(1):1–17.

[Bosc and Pivert, 1997] Bosc, P. and Pivert, O. (1997). On the comparison of imprecise values in fuzzy databases. In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97)*, volume 2, pages 707–712, Barcelona, Spain. IEEE.

[Bouchon-Meunier et al., 2010] Bouchon-Meunier, B., Coletti, G., Lesot, M. J., and Rifqi, M. (2010). Towards a conscious choice of a fuzzy similarity measure: A qualitative point of view. In Hüllermeier, E., Kruse, R., and Hoffmann, F., editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6178 LNAI of *Lecture Notes in Computer Science*, pages 1–10. Springer.

[Cai et al., 2014] Cai, Y., Leung, H. F., Li, Q., Min, H., Tang, J., and Li, J. (2014). Typicality-based collaborative filtering recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):766–779.

[Cai et al., 2010] Cai, Y., Leung, H.-f., Li, Q., Tang, J., and Li, J. (2010). Recommendation based on object typicality. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10*, page 1529, New York, New York, USA. ACM Press.

[Cebiric et al., 2015] Cebiric, S., Goasdoué, F., and Manolescu, I. (2015). Query-Oriented Summarization of RDF Graphs. *Proceedings of the VLDB Endowment*, 8(12):2012–2015.

[Celma, 2008] Celma, Ò. (2008). FOAFing the music: Bridging the semantic gap in music recommendation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):250–256.

[Chapman and Jagadish, 2009] Chapman, A. and Jagadish, H. V. (2009). Why not? In *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 523, New York, New York, USA. ACM Press.

[Cheney et al., 2007] Cheney, J., Chiticariu, L., and Tan, W.-C. (2007). Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474.

[Chu and Chen, 1994] Chu, W. W. and Chen, Q. (1994). A Structured Approach for Cooperative Query Answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738–749.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

[De Calmès et al., 2003] De Calmès, M., Dubois, D., Hullermeier, E., Prade, H., and Sedes, F. (2003). Flexibility and fuzzy case-based evaluation in querying: An illustration in an experimental setting. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11(01):43–66.

[Demidova et al., 2010] Demidova, E., Fankhauser, P., Zhou, X., and Nejdl, W. (2010). DivQ: diversification for keyword search over structured databases. *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338.

[Deshpande and Karypis, 2004] Deshpande, M. and Karypis, G. (2004). Item-based top-N recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177.

[Di Noia et al., 2012] Di Noia, T., Mirizzi, R., Ostuni, V. C., Romito, D., and Zanker, M. (2012). Linked open data to support content-based recommender systems. *Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS '12*, page 1.

[Dodgson, 1883] Dodgson, C. L., editor (1883). *Euclid books I, II.* MacMillan, London, 2nd edition.

[Dolog et al., 2009] Dolog, P., Stuckenschmidt, H., Wache, H., and Diederich, J. (2009). Relaxing RDF queries based on user and domain preferences. *Journal of Intelligent Information Systems*, 33(3):239–260.

[Dramé et al., 2015] Dramé, K., Smits, G., and Pivert, O. (2015). Coarse to fine keyword queries with user interactions. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications &Services - iiWAS '15*, pages 1–10, New York, New York, USA. ACM Press.

[Dubois and Prade, 1980] Dubois, D. and Prade, H. (1980). *Fuzzy sets and systems: theory and applications.* Number 144. Academic Press.

[Dubois and Prade, 1986] Dubois, D. and Prade, H. (1986). Weighted minimum and maximum operations in fuzzy set theory. *Information Sciences*, 39(2):205–210.

[Dubois and Prade, 1993] Dubois, D. and Prade, H. (1993). On data summarization with fuzzy sets. In *Proc. of the 5th Inter. Fuzzy Systems Assoc. World Congress (IFSA'93)*, pages 465–468, Seoul, Korea.

[Dubois and Prade, 2016] Dubois, D. and Prade, H. (2016). Bridging gaps between several forms of granular computing. *Granular Computing*, 1(2):115–126.

[Ell et al., 2014] Ell, B., Harth, A., and Simperl, E. (2014). SPARQL Query Verbalization for Explaining Semantic Search Engine Queries. In *11th International Conference, ESWC 2014*, pages 426–441.

[Fang et al., 2011] Fang, L., Sarma, A. A. D., Yu, C., and Bohannon, P. (2011). REX: Explaining Relationships Between Entity Pairs. *Proceedings of the VLDB Endowment (PVLDB)*, 5(3):241–252.

[Farreny and Prade, 1984] Farreny, H. and Prade, H. (1984). On the Best Way of Designating Objects in Sentence Generation. *Kybernetes*, 13(1):43–46.

[Felfernig and Burke, 2008] Felfernig, A. and Burke, R. (2008). Constraint-based recommender systems. In *Proceedings of the 10th international conference on Electronic commerce - ICEC '08*, page 1, New York, New York, USA. ACM Press.

[Ferré, 2016] Ferré, S. (2016). Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web*, 8(3):405–418.

[Funk, 2006] Funk, S. (2006). Netflix Update: Try This at Home.

[Fürnkranz and Hüllermeier, 2011] Fürnkranz, J. and Hüllermeier, E. (2011). *Preference Learning*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Gaasterland et al., 1992] Gaasterland, T., Godfrey, P., and Minker, J. (1992). An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157.

[Gaume et al., 2013] Gaume, B., Navarro, E., and Prade, H. (2013). Clustering bipartite graphs in terms of approximate formal concepts and sub-contexts. *International Journal of Computational Intelligence Systems*, 6(6):1125–1142.

[Gedikli et al., 2014] Gedikli, F., Jannach, D., and Ge, M. (2014). How should i explain? A comparison of different explanation types for recommender systems. *International Journal of Human Computer Studies*, 72(4):367–382.

[Grice, 1975] Grice, P. (1975). Logic and Conversation. In *Syntax and Semantics*, volume 3, pages 41–58. Academic Press.

[Guillaume and Charnomordic, 2004] Guillaume, S. and Charnomordic, B. (2004). Generating an interpretable family of fuzzy partitions from data. *IEEE Transactions on Fuzzy Systems*, 12(3):324–335.

[Ha-Thuc et al., 2016] Ha-Thuc, V., Xu, Y., Kanduri, S. P., Wu, X., Dialani, V., Yan, Y., Gupta, A., and Sinha, S. (2016). Search by Ideal Candidates: Next Generation of Talent Search at LinkedIn. *The 26th International World Wide Web Conference*, pages 195–198.

[Hadjali et al., 2015] Hadjali, A., Baron, M., Fokou, G., Jean, S., Hadjali, A., and Baron, M. (2015). Cooperative Techniques for SPARQL Query Relaxation in RDF Databases. *The Semantic Web. Latest Advances and New Domains*, pages 237–252.

[Herschel, 2013] Herschel, M. (2013). Wondering Why Data are Missing from Query Results? Ask Conseil Why-Not. *International Conference on Information and Knowledge Management, Proceedings*, pages 2213–2218.

[Herschel and Hernández, 2010] Herschel, M. and Hernández, M. A. (2010). Explaining missing answers to SPJUA queries. *Proceedings of the VLDB Endowment*, 3(1-2):185–196.

[Hristidis and Papakonstantinou, 2002] Hristidis, V. and Papakonstantinou, Y. (2002). DISCOVER: keyword search in relational databases. In *Vldb*, pages 670–681. VLDB Endowment.

[Huang et al., 2008] Huang, H., Liu, C., and Zhou, X. (2008). Computing relaxed answers on RDF databases. In *Web Information Systems Engineering - WISE 2008*, pages 163–175.

[Hurtado et al., 2006] Hurtado, C., Poulovassilis, A., and Wood, P. (2006). A relaxed approach to RDF querying. *The Semantic Web-ISWC 2006*, pages 314–328.

[Ilyas et al., 2008] Ilyas, I. F., Beskales, G., and Soliman, M. a. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11:1—-11:58.

[Jayaram et al., 2014] Jayaram, N., Khan, A., Li, C., Yan, X., and Elmasri, R. (2014). Towards a Query-by-Example System for Knowledge Graphs. In *Proceedings of Workshop on GRAph Data management Experiences and Systems - GRADES'14*, pages 1–6, New York, New York, USA. ACM Press.

[Jayaram et al., 2016] Jayaram, N., Khan, A., Li, C., Yan, X., and Elmasri, R. (2016). Querying knowledge Graphs By Example entity tuples. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, volume 27, pages 1494–1495. IEEE.

[Jeckmans et al., 2013] Jeckmans, A. J. P., Beye, M., Erkin, Z., Hartel, P., Lagendijk, R. L., and Tang, Q. (2013). Privacy in Recommender Systems. In *Social Media Retrieval*, pages 263–281.

[Kacprzyk and Zadrozny, 1999] Kacprzyk, J. and Zadrozny, S. (1999). *On interactive linguistic summarization of databases via a fuzzy-logic-based querying add-on to microsoft access?*, volume 1625.

[Kaplan, 1982] Kaplan, S. (1982). Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19(2):165–187.

[Kato et al., 2012] Kato, M., Ohshima, H., and Tanaka, K. (2012). Content-based retrieval for heterogeneous domains: Domain adaptation by relative aggregation points. In *SIGIR'12 - Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 811–820, New York, New York, USA. ACM Press.

[Kearns and Valiant, 1994] Kearns, M. and Valiant, L. (1994). Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *Journal of the ACM*, 41(1):67–95.

[Khan et al., 2011] Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., and Tao, S. (2011). Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 international conference on Management of data - SIGMOD '11*, volume 1, page 901, New York, New York, USA. ACM Press.

[Khatchadourian and Consens, 2010] Khatchadourian, S. and Consens, M. P. (2010). ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud. In *The Semantic Web: Research and Applications. ESWC 2010*, pages 272–287.

[Koudas et al., 2006] Koudas, N., Li, C., Tung, A. K. H., and Vernica, R. (2006). Relaxing Join and Selection Queries. In *Proceedings of the 32nd international conference on Very large data bases*, pages 199–210.

[Koutrika et al., 2009] Koutrika, G., Bercovitz, B., and Garcia-Molina, H. (2009). FlexRecs: Expressing and Combining Flexible Recommendations. In *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 745. ACM.

[Koutrika and Simitsis, 2013] Koutrika, G. and Simitsis, A. (2013). Mirror mirror on the wall , which query's fairest of them all ? In *Cidr*. Citeseer.

[Krishnapuram et al., 2001] Krishnapuram, R., Joshi, A., Nasraoui, O., and Yi, L. (2001). Low-complexity fuzzy relational clustering algorithms for Web mining. *IEEE Transactions on Fuzzy Systems*, 9(4):595–607.

[Krulwich, 1997] Krulwich, B. (1997). LIFESTYLE FINDER: Intelligent User Profiling Using Large-Scale Demographic Data. *AI Magazine*, 18(2):37.

[Lehmann et al., 2007] Lehmann, J., Schüppel, J., and Auer, S. (2007). Discovering Unknown Connections - the DBpedia Relationship Finder. *Proceedings of the 1st SABRE Conference on Social Semantic Web - CSSW '07*, pages 99–110.

[Lemire and Maclachlan, 2005] Lemire, D. and Maclachlan, A. (2005). Slope One Predictors for Online Rating-Based Collaborative Filtering. In *Proceedings of the 2005*

*{SIAM} International Conference on Data Mining, {SDM} 2005, Newport Beach, CA, USA, April 21-23, 2005*, pages 471—-475.

[Lesot and Revault d'Allonnes, 2012] Lesot, M.-J. and Revault d'Allonnes, A. (2012). Credit-Card Fraud Profiling Using a Hybrid Incremental Clustering Methodology. In *6th International Conference, SUM 2012*, pages 325–336.

[Lesot et al., 2008] Lesot, M. J., Rifqi, M., and Bouchon-Meunier, B. (2008). Fuzzy prototypes: From a cognitive view to a machine learning principle. *Studies in Fuzziness and Soft Computing*, 220:431–452.

[Lesot et al., 2013] Lesot, M. J., Smits, G., and Pivert, O. (2013). Adequacy of a user-defined vocabulary to the data structure. In *IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE.

[Li and Jagadish, 2014] Li, F. and Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

[Liu and Jagadish, 2009] Liu, B. and Jagadish, H. V. (2009). DataLens: Making a Good First Impression. *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, page 1115.

[Lorenzi and Ricci, 2005] Lorenzi, F. and Ricci, F. (2005). Case-based recommender systems: A unifying view. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3169 LNAI:89–113.

[Marsala and Bouchon-Meunier, 1996] Marsala, C. and Bouchon-Meunier, B. (1996). Fuzzy partitioning using mathematical morphology in a learning scheme. In *Proceedings of IEEE 5th International Fuzzy Systems*, volume 2, pages 1512–1517. IEEE.

[McCoy, 1988] McCoy, K. F. (1988). Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics - Special issue on user modeling*, 14(3):52–63.

[McSherry, 2005] McSherry, D. (2005). Explanation in recommender systems. *Artificial Intelligence Review*, 24(2):179–197.

[Meliou et al., 2010] Meliou, A., Gatterbauer, W., Halpern, J. Y., Koch, C., Moore, K. F., and Suciu, D. (2010). Causality in databases. *IEEE Data Eng. Bull.*, 33(EPFL-ARTICLE-165841):59–67.

[Middleton et al., 2009] Middleton, S. E., Roure, D. D., and Shadbolt, N. R. (2009). Ontology-Based Recommender Systems. In *Handbook on Ontologies*, pages 779–796. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Minker, 1998] Minker, J. (1998). An overview of cooperative answering in databases. In *Flexible Query Answering Systems*, pages 282–285. Springer.

[Mishra and Koudas, 2009] Mishra, C. and Koudas, N. (2009). Interactive query refinement. *Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology - EDBT '09*, page 862.

[Moreau et al., 2015] Moreau, A., Pivert, O., and Smits, G. (2015). A Clustering-Based Approach to the Explanation of Database Query Answers. In *Proc. of the 11th International Conference on Flexible Query Answering Systems (FQAS'15)*, Krakow, Poland.

[Moreau et al., 2016a] Moreau, A., Pivert, O., and Smits, G. (2016a). A Fuzzy Approach to the Characterization of Database Query Answers. In *Proc. of the 16th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'16)*, Eindhoven, Netherlands.

[Moreau et al., 2016b] Moreau, A., Pivert, O., and Smits, G. (2016b). Caractérisation floue de clusters de réponses. In *Actes des Rencontres Francophones sur la Logique Floue et ses Applications (LFA'16)*, La Rochelle, France.

[Moreau et al., 2017] Moreau, A., Pivert, O., and Smits, G. (2017). A Typicality-Based Recommendation Approach Leveraging Demographic Data. In *Proc. of the 12th International Conference on Flexible Query Answering Systems (FQAS'17)*, pages 71–83, London, UK.

[Moreau et al., 2018] Moreau, A., Pivert, O., and Smits, G. (2018). Fuzzy Query By Example. In *Proc. of the 33rd ACM Symposium on Applied Computing (SAC'18)*, Pau, France.

[Motro, 1994] Motro, A. (1994). Intensional Answers to Database Queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):444–454.

[Mottin et al., 2014a] Mottin, D., Lissandrini, M., Velegrakis, Y., and Palpanas, T. (2014a). Exemplar Queries: Give me an Example of What You Need. *Proceedings of the VLDB Endowment*, pages 365–376.

[Mottin et al., 2014b] Mottin, D., Lissandrini, M., Velegrakis, Y., and Palpanas, T. (2014b). Searching with XQ. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD'14*, pages 901–904.

[Ngonga Ngomo et al., 2013] Ngonga Ngomo, A.-C., Bühmann, L., Unger, C., Lehmann, J., and Gerber, D. (2013). Sorry, i don't speak SPARQL. In *Proceedings of the 22nd international conference on World Wide Web - WWW '13*, pages 977–988, New York, New York, USA. ACM Press.

[Osherson and Smith, 1997] Osherson, D. and Smith, E. E. (1997). On typicality and vagueness. *Cognition*, 64(2):189–206.

[Pappis and Karacapilidis, 1993] Pappis, C. P. and Karacapilidis, N. I. (1993). A comparative assessment of measures of similarity of fuzzy values. *Fuzzy Sets and Systems*, 56(2):171–174.

[Passant, 2010a] Passant, A. (2010a). dbrec - Music Recommendations Using DBpedia.pdf. In *ISWC'10 Proceedings of the 9th international semantic web conference on The semantic web - Volume Part II*, volume 1380, pages 1–16.

[Passant, 2010b] Passant, A. (2010b). Measuring semantic distance on linking data and using it for resources recommendations. *Proceedings of the AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, SS-10-07:93–98.

[Pawlak, 1991] Pawlak, Z. (1991). *Rough Sets*. Springer Netherlands, Dordrecht.

[Pazzani, 1999] Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408.

[Peska and Vojtas, 2013] Peska, L. and Vojtas, P. (2013). Using Linked Open Data in Recommender Systems. In *SeRSy'13*, pages 1–6. ACM Press.

[Pivert and Bosc, 2012] Pivert, O. and Bosc, P. (2012). *Fuzzy Preference Queries to Relational Databases*. Imperial College Press.

[Pivert and Prade, 2012] Pivert, O. and Prade, H. (2012). Detecting suspect answers in the presence of inconsistent information. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7153 LNCS:278–297.

[Pivert et al., 2016] Pivert, O., Slama, O., Thion, V., Pivert, O., Slama, O., and Thion, V. (2016). An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data.

[Pivert and Smits, 2014] Pivert, O. and Smits, G. (2014). Plethoric Answers to Fuzzy Queries: A Reduction Method Based on Query Mining. In *21st International Symposium, ISMIS 2014*, pages 295–304.

[Pivert et al., 2013] Pivert, O., Smits, G., and Jaudoin, H. (2013). Finding Similar Objects in Relational Databases – An Association-Based Fuzzy Approach. In *10th International Conference, FQAS 2013*, pages 425–436. Springer.

[Qin et al., 2012] Qin, L., Yu, J. X., and Chang, L. (2012). Diversifying top-k results. *Proceedings of the VLDB Endowment*, 5(11):1124–1135.

[Quilici et al., 1988] Quilici, A., G. Dyer, M., and Flowers, M. (1988). Recognizing and Responding To Plan-Oriented Misconceptions. *Computational Linguistics, Volume 14, Number 3, September 1988*, 14(3).

[Ricci et al., 2015] Ricci, F., Rokach, L., and Shapira, B., editors (2015). *Recommender Systems Handbook*. Springer US, Boston, MA, 2 edition.

[Roy and Suciu, 2014] Roy, S. and Suciu, D. (2014). A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD'14*, pages 1579–1590, New York, New York, USA. ACM Press.

[Ruotsalo et al., 2013] Ruotsalo, T., Haav, K., Stoyanov, A., Roche, S., Fani, E., Deliai, R., Mäkelä, E., Kauppinen, T., and Hyvönen, E. (2013). SMARTMUSEUM: A mobile recommender system for the Web of Data. *Journal of Web Semantics*, 20:50–67.

[Ruspini, 1969] Ruspini, E. H. P. (1969). New Approach to Clustering. *Information and Control*, 15(1):22–32.

[Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th ...*, volume 1, pages 285–295, New York, New York, USA. ACM Press.

[Sarwat et al., 2013] Sarwat, M., Avery, J., and Mokbel, M. F. (2013). RecDB in action. *Proceedings of the VLDB Endowment*, 6(12):1242–1245.

[Semeraro et al., 2009] Semeraro, G., Lops, P., Basile, P., and de Gemmis, M. (2009). Knowledge infusion into content-based recommender systems. In *Proceedings of the third ACM conference on Recommender systems - RecSys'09*, page 301, New York, New York, USA. ACM Press.

[Simitsis et al., 2007] Simitsis, A., Koutrika, G., and Ioannidis, Y. (2007). Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal*, 17(1):117–149.

[Singh et al., 2016] Singh, M., Cafarella, M. J., Arbor, A., and Jagadish, H. V. (2016). DBExplorer: Exploratory Search in Databases. In *Proc. 19th International Conference on Extending Database Technology (EDBT)*, pages 89–100.

[Smits et al., 2017a] Smits, G., Lesot, M.-J., and Pivert, O. (2017a). Vocabulary Elicitation for Informative Descriptions of Classes. In *joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCSI'17)*, Otsu, Japan.

[Smits et al., 2013] Smits, G., Pivert, O., and Girault, T. (2013). ReqFlex: Fuzzy Queries for Everyone. *Proceedings of the VLDB Endowment*, 6(12):1206–1209.

[Smits et al., 2014a] Smits, G., Pivert, O., and Hadjali, A. (2014a). Fuzzy cardinalities as a basis to cooperative answering. *Studies in Computational Intelligence*, 497:261–289.

[Smits et al., 2014b] Smits, G., Pivert, O., Jaudoin, H., and Paulus, F. (2014b). AGGREGO SEARCH: Interactive Keyword Query Construction. In *International Conference on Extending Data Base technology (EDBT'14)*, pages 636–639.

[Smits et al., 2014c] Smits, G., Pivert, O., and Lesot, M.-j. (2014c). A Vocabulary Revision Method Based on Modality Splitting. In *15th International Conference, IPMU 2014*, pages 140–149.

[Smits et al., 2015] Smits, G., Pivert, O., and Thion, V. (2015). Connected keywords. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 112–120. IEEE.

[Smits et al., 2017b] Smits, G., Yager, R. R., and Pivert, O. (2017b). Interactive Data Exploration on Top of Linguistic Summaries. In *In Proc. of the 26th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'17)*, Naples, Italy.

[Stefanidis et al., 2009] Stefanidis, K., Drosou, M., and Pitoura, E. (2009). "You May Also Like" results in relational databases. *Proc. of PersDB*, pages 37–42.

[Stoica et al., 2007] Stoica, E., Hearst, M. A., and Richardson, M. (2007). Automating Creation of Hierarchical Faceted Metadata Structures. In *Proceedings of NAACL HLT 2007*, pages 244–251.

[Sun et al., 2011] Sun, Y., Han, J., Yan, X., Yu, P. S., and Wu, T. (2011). PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003.

[Tahani, 1977] Tahani, V. (1977). A conceptual framework for fuzzy query processing-A step toward very intelligent database systems. *Information Processing and Management*, 13(5):289–303.

[Tatemura et al., 2008] Tatemura, J., Chen, S., Liao, F., Po, O., Candan, K. S., and Agrawal, D. (2008). UQBE: Uncertain Query By Example for Web Service Mashup. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD'08*, page 1275, New York, New York, USA. ACM Press.

[Tintarev and Masthoff, 2012] Tintarev, N. and Masthoff, J. (2012). Evaluating the effectiveness of explanations for recommender systems: Methodological issues and empirical studies on the impact of personalization. *User Modeling and User-Adapted Interaction*, 22(4-5):399–439.

[Tran and Chan, 2010] Tran, Q. T. and Chan, C.-Y. (2010). How to ConQueR why-not questions. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 15–26, New York, New York, USA. ACM Press.

[Tsukuda et al., 2013] Tsukuda, K., Ohshima, H., Yamamoto, M., Iwasaki, H., and Tanaka, K. (2013). Discovering unexpected information on the basis of popularity/unpopularity analysis of coordinate objects and their relationships. In *Proceedings of the ACM Symposium on Applied Computing*, pages 878–885, New York, New York, USA. ACM.

[Ughetto et al., 2008] Ughetto, L., Voglozin, W. A., and Mouaddib, N. (2008). Database querying with personalized vocabulary using data summaries. *Fuzzy Sets and Systems*, 159(15):2030–2046.

[Vozalis and Margaritis, 2007] Vozalis, M. G. and Margaritis, K. G. (2007). Using SVD and demographic data for the enhancement of generalized Collaborative Filtering. *Information Sciences*, 177(15):3017–3037.

[Wang et al., 2013] Wang, C. J., Lin, Y. W., Tsai, M. F., and Chen, H. H. (2013). *Mining subtopics from different aspects for diversifying search results*, volume 16.

[Wang et al., 2012] Wang, Y., Chan, S. C. F., and Ngai, G. (2012). Applicability of demographic recommender system to tourist attractions: A case study on TripAdvisor. *Proceedings of the 2012 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops, WI-IAT 2012*, pages 97–101.

[Weinsberg et al., 2012] Weinsberg, U., Bhagat, S., Ioannidis, S., and Taft, N. (2012). BlurMe: Inferring and Obfuscating User Gender Based on Ratings. *Proceedings of the sixth ACM conference on Recommender systems - RecSys '12*, page 195.

[Wu and Madden, 2013] Wu, E. and Madden, S. (2013). Scorpion: Explaining Away Outliers in Aggregate Queries. *Proceedings of the VLDB Endowment*, 6(8):553–564.

[Yager, 1984] Yager, R. R. (1984). General multiple-objective decision functions and linguistically quantified statements. *International Journal of Man-Machine Studies*, 21(5):389–400.

[Yager, 1988] Yager, R. R. (1988). On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190.

[Yager, 1994] Yager, R. R. (1994). Interpreting linguistically quantified propositions. *International Journal of Intelligent Systems*, 9(6):541–569.

[Yager, 1997] Yager, R. R. (1997). A note on a fuzzy measure of typicality. *International Journal of Intelligent Systems*, 12(3):233–249.

[Yang et al., 2009] Yang, X., Procopiuc, C. M., and Srivastava, D. (2009). Summarizing relational databases. *Proceedings of the VLDB Endowment*, 2(1):634–645.

[Yu and Jagadish, 2006] Yu, C. and Jagadish, H. V. (2006). Schema summarization. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 319–330. VLDB Endowment.

[Zadeh, 1965] Zadeh, L. A. (1965). Fuzzy Sets. *Information and Control*, 8(3):338–353.

[Zadeh, 1983] Zadeh, L. a. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computers & Mathematics with Applications*, 9(1):149–184.

[Zadeh, 1987] Zadeh, L. a. (1987). A Computational Theory of DIspostions. *International Journal of Intelligent Systems*, 2(1):39–63.

[Zadrozny and Kacprzyk, 1996] Zadrozny, S. and Kacprzyk, J. (1996). FQUERY for Access. In *Proceedings of the 1996 ACM symposium on Applied Computing - SAC '96*, pages 532–536, New York, New York, USA. ACM Press.

[Zadrozny et al., 2010] Zadrozny, S., Kacprzyk, J., and Wysocki, M. (2010). On a novice-user-focused approach to flexible querying: The case of initially unavailable explicit user preferences. *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA'10*, pages 696–701.

[Zloof, 1975] Zloof, M. M. (1975). Query by Example. *Proceedings of the National Computer Conference and Exposition*, pages 431–438.

[Zloof, 1977] Zloof, M. M. (1977). Query-by-example: A Data Base Language. *IBM Syst. J.*, 16(4):324–343.

# List of Figures


1.1 Sensitivity around the borders, explained. . . . . 3

3.1 Example schema based on XMark, from [Yu and Jagadish, 2006] . . . . 25
3.2 Schema summaries from [Yu and Jagadish, 2006] . . . . . 25
3.3 An example of partition associated with the domain of the attribute hardness, from [Ughetto et al., 2008] . . . . . 26
3.4 A (hierarchical) rewriting of a term based on two adjacent terms in a partition, from [Ughetto et al., 2008] . . . . . 27

4.1 ClusterXplain workflow . . . . . 46
4.2 A partition over the domain of the attribute *year* . . . . . 50
4.3 Projection onto $\mathcal{A}_\pi$ . . . . . 52
4.4 Projection onto $\mathcal{A}_\omega$ . . . . . 53
4.5 Representation of three clusters on the attributes $X$ and $Y$: Example 4.11 61
4.6 Different clustering results . . . . . 64
4.7 Full clusters of second-hand cars over the attributes *price* and *mileage* . 65
4.8 Processing times in milliseconds (crisp approach). . . . . 67
4.9 Processing times in milliseconds (fuzzy approach). . . . . 67

5.1 A representation of the fuzzy quantifier *most* . . . . . 78
5.2 Relational schema of the cinematographic database . . . . . 82
5.3 Survey screenshot. . . . . 101
5.4 Relevance before explanation . . . . . 102
5.5 Relevance after explanation . . . . . 102

6.1 A partition over the domain of the attribute *year* . . . . . 109

## Résumé

Dans ces travaux de thèse nous proposons de tirer parti de la théorie des ensembles flous afin d'améliorer les interactions entre les systèmes de bases de données et les utilisateurs. Les mécanismes coopératifs visent à aider les utilisateurs à mieux interagir avec les SGBD. Ces mécanismes doivent faire preuve de *robustesse* : ils doivent toujours pouvoir proposer des réponses à l'utilisateur. `Empty set (0,00 sec)` est un exemple typique de réponse qu'il serait désirable d'éradiquer. Le caractère informatif des explications de réponses est parfois plus important que les réponses elles-mêmes : ce peut être le cas avec les réponses vides et pléthoriques par exemple, d'où l'intérêt de mécanismes coopératifs *robustes*, capables à la fois de contribuer à l'*explication* ainsi qu'à l'*amélioration* des résultats. Par ailleurs, l'utilisation de termes de la langue naturelle pour décrire les données permet de garantir l'*interprétabilité* des explications fournies. Permettre à l'utilisateur d'utiliser des mots de son propre vocabulaire contribue à la personnalisation des explications et améliore l'interprétabilité.

Nous proposons de nous intéresser aux explications dans le contexte des réponses coopératives sous trois angles : 1) dans le cas d'un ensemble pléthorique de résultats ; 2) dans le contexte des systèmes de recommandation ; 3) dans le cas d'une recherche à partir d'exemples. Ces axes définissent des approches coopératives où l'intérêt des explications est de permettre à l'utilisateur de comprendre comment sont calculés les résultats proposés dans un effort de transparence. Le caractère informatif des explications apporte une valeur ajoutée aux résultats bruts, et forme une réponse coopérative.

## Abstract

In this thesis, we are interested in how we can leverage fuzzy logic to improve the interactions between relational database systems and humans. Cooperative answering techniques aim to help users harness the potential of DBMSs. These techniques are expected to be *robust* and always provide answers to users. `Empty set (0,00 sec)` is a typical example of answer that one may wish to never obtain. The informative nature of explanations is higher than that of actual answers in several cases, *e.g.* empty answer sets and plethoric answer sets, hence the interest of *robust* cooperative answering techniques capable of both *explaining* and *improving* an answer set. Using terms from natural language to describe data — with labels from fuzzy vocabularies — contributes to the *interpretability* of explanations. Offering to define and refine vocabulary terms increases the personalization experience and improves the interpretability by using the user's own words.

We propose to investigate the use of explanations in a cooperative answering setting using three research axes: 1) in the presence of a plethoric set of answers; 2) in the context of recommendations; 3) in the context of a query/answering problem. These axes define cooperative techniques where the interest of explanations is to enable users to understand how results are computed in an effort of transparency. The informativeness of the explanations brings an added value to the direct results, and that in itself represents a cooperative answer.