A query language for constraint-based clustering

Antoine ADAM ANTOINE.ADAM@CS.KULEUVEN.BE
KU Leuven, Department of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium

Hendrik Blockeel HENDRIK.BLOCKEEL@CS.KULEUVEN.BE
KU Leuven, Department of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium
Leiden Institute of Advanced Computer Science, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Keywords: query language, inductive database, constraint-based clustering

Abstract

Clustering is a widely used data mining task and a lot of constraint-based clustering methods have been developped. Our work focus on the problem of integrating constraint-based clustering in an inductive database system. We propose a new extension of SQL for constraint-based clustering. We present a concrete application in the context of microbiology.

1. Introduction

Data mining provides many methods to discover patterns or learn models from data. Even for a given type of pattern, say, association rules, multiple systems are usually available, each with their strengths and weaknesses in terms of efficiency and flexibility. As a result, it may not be easy, even for specialist users, to solve a particular data mining problem in the best possible way. Inductive database (IDB) systems (Imieliński & Mannila, 1996) can address this problem to some extent.

The idea behind IDBs is to seamlessly incorporate data mining in databases. In an IDB system, patterns are first-class citizens and can be manipulated at the same level than the data using an inductive query language. An interesting characteristic of such a query language is that it is declarative. This means the user specifies the task to execute (e.g., "find all association rules with confidence at least c, support at least s, and one of A, B, C in the head"), but not the algorithm or method that should be run. This is an advantage over data mining tools such as Weka (Hall et al., 2009), where the user must select a particular algorithm, and then accept the limitations that this choice entails. In a declarative specification, it is also easier to specify constraints on the patterns to look for. This links inductive databases to constraint-based data mining, as argued by Džeroski (2011).

Much research on inductive databases has focused on one specific data-mining task, namely association rule mining. Certain other tasks, among which clustering, have not received much attention. Further, for those IDB systems that do cover different types of tasks, the formulation of a problem can be difficult for non-specialists.

In this work, we focus on the problem of constraint-based clustering or semi-supervised clustering. Various methods have been developed in this field, introducing various kinds of constraints that act at different levels. It can go from global-level constraints that act on the resulting clusters, for example specifying the number of clusters or a minimum size of the cluster, to instance-level constraints, for example must-link and cannot-link constraints studied by Wagstaff (2002), also called equivalence constraints, that state if two instances must be or cannot be in the same cluster. We propose a query language, an extension of SQL, that allows the user to query for clusterings and formulate constraints in an easy way.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 presents the query language we propose; Section 4 shows an application of the query language in a microbiology context; Section 5 concludes and discusses future work.

Appearing in *Proceedings of BENELEARN 2013*. Copyright 2013 by the author(s)/owner(s).

2. Related work

The concept of inductive database has been introduced by Imielinski and Manilla (1996), followed then by De Raedt (2002b). Boulicaut and Masson (2005) and more recently Romei and Turini (2011) discussed some requirements of inductive database languages and compare some existing systems.

Various inductive database systems have been developped for different types of databases: SQL-based systems (MINE RULE (Meo et al., 1998), DMQL (Han et al., 1996), MSQL (Imieliński & Virmani, 1999), SPQL/ConQueSt (Bonchi et al., 2006), Mining Views (Blockeel et al., 2012), SiQL/SINDBAD (Wicker et al., 2008), ATLaS (Wang & Zaniolo, 2003)), DMX (Microsoft, 2012); XML-based systems (XMINE RULE (Braga et al., 2003), KDDML (Romei et al., 2006)); and logic-based systems (RDM (De Raedt, 2002a), LDL++ (Giannotti et al., 2004)). Many of these systems only deal with association rule mining (MINE RULE, DMQL, MSQL, XMine). The SINDBAD system, along with the SiQL language, supports the whole data mining process, from pre-processing to post-processing. It includes clustering but only using the k-Medoids algorithm. The ATLaS system extends SQL with user-defined aggregates which can be used for clustering. DMX also includes clustering with KMeans and EM algorithms. The Mining Views framework does not extend the query language but uses virtual mining views to do different data mining tasks. It has the advantage to integrate the data mining process in the database management system. There are also applied system specific to one domain, like Molfea (Helma et al., 2002), for mining frequent molecular structures. Even if some of these language can be extended to include equivalence constraints for clustering, we propose a new language to express such constraints in an easy and declarative way that fits the SQL relational model.

3. Query language

In this section, we present our query language. Before presenting the actual query, we will discuss the principles that guided our choice in the design of the query language.

3.1. Language properties

The goal of the project is to build a system that allows someone who does not necessarily know data mining to apply data mining algorithms on his data. For this the user should be able to ask what he wants in an easy and declarative way. We list different characteristics that we want our language to have and that guided the design of the query language.

- First, the language should be *concise*. This means queries for simple problem should be short. For instance, asking for a simple clustering without any constraints or specific parameters should be very easy.
- Second, the language should be *intuitive*, and this for both formulating and understanding a query. This means that given a question, it should be easy to formulate it in a query, but also that a written query can be easily understandable.
- Finally, the language should be *expressive*. This means that the language should allow formulating complex queries. This implies that the language should allow various constraints that can be combined. In the idea of an iterative knowledge discovery process, this also includes the closure principle. This principle says that we should be able to query the result of a query. This allows composing queries in a complex way. For instance, it is possible to first execute a clustering using some parameters, then select the instances of one cluster and do another clustering on these instances using other parameters.

3.2. Language syntax

We choose to build an extension of SQL as it is widely used and intuitive. In the logic of inductive database, data and patterns should be considered at the same level. This implies that a query for patterns should follow the same logic as a query for data. For this reason, we designed our clustering query based on the SELECT query of SQL.

For a better understanding, we will show query examples on the following imaginary dataset : we have a table named *points* with 4 attributes *id*, *x*, *y*, *valid*. *id* is an integer, *x* and *y* are real numbers, *valid* is 0 or 1 or null if unknown.

3.2.1. CLUSTER statement

We introduce the new CLUSTER statement in Figure 1. Let us explain the different parts :

data The data we want to cluster. It is a table whose columns are the attributes and lines are the instances to cluster. It can be a table present in the database or the result of a SQL select query.

attributes This part differs from the SELECT query. In the select query, it indicates which column to

```
<statement> ::= "CLUSTER" <attributes>
  "FROM" <data>
  ["WITH" <constraints>]

<attributes> ::= "*" | <attr>
<attr> ::= <attributename>[, attr]

<constraints> ::= <c> ["AND" <constraints>]
<c> ::= <nbclusterconstraint>
      | <linkconstraint>

<nbclusterconstraint> ::=
  "NumberOfClusters = n"

<linkconstraint> ::=
  ["SOFT"]["MUST"|"CANNOT"] LINK
  <cdata>
  ["BY "<attributename>]
```

Figure 1. CLUSTER statement

select and return in the result. In a cluster query, this indicates which columns to use for the clustering task. This way, you can ignore some attributes that are not relevant for the clustering task but still have them in the result of the query, like the Ids. As in the SELECT query, a * means that all columns are used for the clustering.

constraints : the constraints you can add to the clustering. They will constrain the result but also the method used to solve the query. The constraints available are :

- the number of clusters : to specify the number of cluster wanted.
- link constraints: instance-level constraints to specify if some instances must be or cannot be in the same cluster.

Here is an example of a query to cluster all the valid points according to x and y.

```
CLUSTER x,y
    FROM (SELECT * FROM points
          WHERE valid=1)
```

3.2.2. Link constraints

Let us now look at the syntax of the link constraints.

cdata The constrained data following LINK are the instances involved in the link constraint. The set of the constrained instances must be a subset of the data selected in the CLUSTER query. Subsequently, they could in principle be selected by a query: SELECT * FROM (data) WHERE (conditions). However, to be more concise and to avoid repeating the (data) part, we only put the conditions that would follow the WHERE in such a query.

MUST/CANNOT LINK The idea of this constraint was first to incorporate must-link and cannot-link constraints as formulatied by Wagstaff (2002). These constraints are pair-wise constraints which means one concerns only two instances. It can be easily understood that if one wants to specify a lot of constraints, the size of one query can quickly increase too much. To specify quickly a large number of constraints, we allow the data to be composed of more than two instances. For MUST LINK, it is supposed that all instances in *cdata* must be in the same cluster. For CANNOT LINK, it is supposed that all instances in *cdata* have to be each in a different cluster. This allows specifying small group of instances that should be clustered together in a more concise way than specifying all pair-wise constraints. Bar-Hillel et al. (2006) studied this idea of making small groups of instances, that they call chunklets. In the following example, the problem is to cluster all the points in 2 cluster, knowing that the points 3, 4 and 6 must be in the same cluster and the points 1 and 3 must be in different clusters. The query formulating this problem is as follows:

```
CLUSTER x, y
  FROM (SELECT * FROM points)
  WITH MUST LINK (id IN (3, 4, 6))
  AND CANNOT LINK (id IN (3, 1))
```

BY The BY word can be used with MUST LINK and CANNOT LINK to add link constraints between instances using an attribute of the data. With MUST LINK, must-link constraints will be created between each pair of instances that have the same value for the specified attribute. With CANNOT LINK, cannot-link constraints will be created between each pair of instances that have different values for the specified attribute. To avoid repetition, it is also possible to just say LINK (data) BY attribute. This will create must-link and cannot-link constraints according to the specified attribute as if it was MUST LINK (data) BY attribute AND CANNOT LINK (data) BY attribute. This allows for more concise queries for instance in the case of partially labeled data. However, if no BY statement is present after the LINK, it is considered as MUST LINK. In the next example, some of the points are labelled as valid, some as invalid, and the rest is unknown. The problem is to cluster all the points in 2 clusters with the valid ones in one cluster and the unvalid ones in another cluster.

```
CLUSTER x, y
  FROM (SELECT * FROM points)
  WITH LINK (valid=0 OR valid=1)
    BY valid
```

SOFT The link constraints are considered hard constraints by default. However, one may be also interested in soft constraints. For instance, let us suppose there is some label that makes a partition of the data. Using this partition as a bias, one can be interested in clustering the data using other attributes do that instances having the same label are more likely to be in the same cluster. This can be achieved by adding a SOFT LINK constraint using this label. In the following example, all points are known as valid or invalid. The problem is to cluster the points according to x and y with a preference that the valid instances are in the same cluster and the unvalid ones are in another one.

```
CLUSTER x, y
  FROM (SELECT * FROM points)
  WITH SOFT LINK BY valid
```

3.3. Result of a query

One of the principles of database and querying language is the closure principle. It says that the result of a query should be queryable. This allows an iterative process for exploring the data progressively. As we are querying tables, the result should also be a table. As a first solution, we adopt one solution of SINDBAD (Wicker et al., 2008). The result of the query is the input table (data) where a column *cluster* has been added that contains for each instance its cluster assignement as a strictly positive integer. Figure 2 shows the different elements of the query CLUSTER x, y FROM (SELECT * FROM points).

This is a very simple way of giving the result of a clustering algorithm. However, it only works for partitioning clustering. It does not allow presenting the result of hierarchical, density-based, overlapping or other kinds of clustering. Such methods can still be used to build the clusters but the result should be a partition of the data. It is an issue we want to addres in future work.

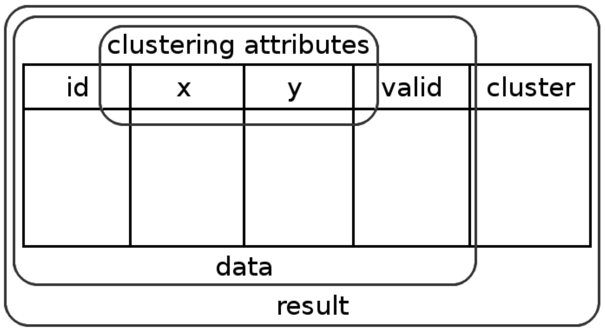

Figure 2. The cluster query viewed as table.

3.4. Query execution

Once a query has been formulated, it has to be executed. The problem is then to choose what method to use. Different methods or algorithms may not lead to the same result. Besides, we have to take into account the different constraints we can add. To do so, we decided to execute different methods depending on the constraints specified. The choice of the method is done as follows:

- By default, the distance used is a Euclidian distance. If soft link constraints are specified, a metric is learned and the resulting Mahalanobis distance is used as in (Bar-Hillel et al., 2006).
- If there are hard link-constraints, the CopKmeans algorithm (Wagstaff, 2002) is executed. For now, the number of clusters has to be specified. If it is not, it is set to the minimum number of clusters so that the link constraints can be respected. If this number is 1, it is set to 2.
- If there are no link constraints, the EM algorithm is used and if the number of clusters is not specified, cross-validation is used to determine it. We used for this the weka implementation of EM.

Currently, only these algorithms are implemented as we have focused our work on the formulation of the query but not on its execution. One of the next goals in our future work is to improve this execution. The challenge will be to combine different kind of constraints.

4. Application

In this section, we present how the query language is used in a concrete application. Our language is domain independent. We here show an example of how it can be integrated in software specific to a domain, in this case microbiology.

4.1. Cellphinder project

Our work is part of a project called CellPhinder. This project groups researchers from microbioloby and computer science and it is the initial source of motivation for our new query language. Our role in the project is to make data-mining available for the microbiologists. As said earlier, our query language is declarative. Therefore, it can be used by people who do not know clustering algorithms but are interested in the result. We indeed believe it will be easier to learn how to use our language than to learn various clustering methods and to have to put the data into the right format. We also choose to use a query language because it easily allows an iterative exploration of the data by reusing previous queries to build new ones. As the microbiologists are the first user of our system, we choose the patterns and constraints that are useful and of interest to them. This is why we focused on clustering, as it provides two interesting learning patterns: on one hand, the clusters can identify normal behavior in the data; on the other hand, instances far from any clusters or in an isolated cluster are outliers, which is also of interest for the microbiologists.

4.2. The software

We are currently building software to make our language accessible to microbiologists. It will provide an easy access to the data, as well as visualisation tools, for example distribution graphs. The user will have the possibility to build the query step by step using graphic tools but also to directly type the query he wants to execute. The software will show the result of the query in various representations, textual or graphical. At this moment, the software is still being developed. When the microbiologists are able to use the software, they will provide us with direct feedback on the language.

Our software is implemented in Java. The system is coupled with a MySQL database. When a cluster query is parsed, the SQL part is given to the database and the resulting data is given to the clustering engine. This engine uses Weka implementations of KMeans and EM and our custom-made implementation of CopKMeans.

4.3. Query examples

4.3.1. The data structure

The structure of the data we are working on is as follows. An Experiment consists of various lineages. A Lineage is the set of cells that originates from one mother cell. This mother cell grows and then divides in two cells, which grow and each divide in two cells, etc... From a Cell, different parameters are measured at each definite lapse of time: length, width, curvature, perimeter, growthspeed,... A cell then has different states over time. Finally, Fluorescence spots are also measured for each state. Fluorescence is made by proteins inside the cell. The fluorescence can be diffuse or localised. Consequently, we have a database made of five tables: experiment, lineage, cell, stateovertime, fluorescence. These tables have one-to-many relation from one to the next (an experiment has many lineage, that have many cells...). We also have a relation inside the cell table between parent/children cells.

4.3.2. Example 1

Assume data from a new experiment, experiment number 5, is available. The different id parameters in the following examples are not relevant for the example, they are only here to have an exact, realistic query. First, we want to cluster the cells of the experiment.

```
CLUSTER LifeTime, LagTime, LengthMean
  FROM (SELECT c.Id, c.LifeTime, c.LagTime,
               AVG(s.Length) AS LengthMean
        FROM stateovertime s, cell c,
             lineage l
        WHERE l.ExperimentId=5
          AND c.LineageId = l.Id
          AND s.CellId = c.Id
          GROUP BY c.id)
```

4.3.3. Example 2

By looking at the data, a few lineages have been found that seem to behave similarly. Another one has been found that seems to behave really differently than the others. It can be interesting to know if there are other lineages like this in the data. Must-link constraints can be added between the "normal" lineages and cannot-link constraints between the special and the others.

```
CLUSTER LifeTimeMean, LagTimeMean
  FROM (SELECT l.Id,
          AVG(c.LifeTime) AS LifeTimeMean,
          AVG(c.LagTime) AS LagTimeMean,
        FROM cell c, lineage l
        WHERE c.LineageId = l.Id
        GROUP BY l.id)
    WITH MUST LINK (Id IN (20, 21,
                           22, 23, 25))
      AND CANNOT LINK (Id IN (20, 24))
```

4.3.4. Example 3

The whole dataset is divided between mutants, which are cells that have been modified, and wildtypes, which are unmodified cells. This suggests certain similarity between cells that are of the same type and it may be interesting to use this background knowledge to do clustering. However, it is already a full partition of the data thus hard constraints are not useful. Therefore, soft constraints can be used. In the lineage table, there is a *Mutant* attribute that is 0 is the cells of the lineage are wildtypes and 1 if they are mutants.

```
  CLUSTER LifeTime, LagTime,
      LengthMean, WidthMean
    FROM (SELECT c.Id, c.LifeTime,
            c.LagTime, l.Mutant
            AVG(s.Length) AS LengthMean
            AVG(s.Width) AS WidthMean
          FROM stateovertime s, cell c,
              lineage l
          WHERE l.ExperimentId=5
            AND c.LineageId = l.Id
            AND s.CellId = c.Id
          GROUP BY c.id)
    WITH SOFT LINK BY Mutant
```

5. Conclusions & Future work

In our work, we have considered must-link and cannot-link constraints which are instance level constraints. The next step will be to include more constraints in our language: feature-level constraints (so that some features can be specified more important than others), global-level constraints (minimum cluster size, balanced clusters). This will raise the problem of solving the query. Indeed, there exist algorithms to solve clustering with these different type of constraints separately. However, the problem occurs when combining different types of constraints in one query.

Another issue we want to adress is the problem of the representation of the result. Indeed, we can only present partitioning clusterings but it can be interesting to try to include other types of results like overlapping, hierarchical or model-based clusterings.

Finally, we have currently included outlier detection as a post-processing step of clustering in our software but it can be interesting to include it in the language as a real task.

Acknowledgments

For their useful comments on this paper, we would like to thank the reviewers as well as Tias Guns. For their collaboration in the Cellphinder project, we would like to thank Abram Aertsen and Sander Govers from the Laboratory of Food Microbiology of the KULeuven. This work is funded by the KULeuven in the project IDO/10/012: Elaboration of the CellPhInDER platform.

References

Bar-Hillel, A., Hertz, T., Shental, N., & Weinshall, D. (2006). Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, *6*, 937.

Blockeel, H., Calders, T., Fromont, É., Goethals, B., Prado, A., & Robardet, C. (2012). An inductive database system based on virtual mining views. *Data Mining and Knowledge Discovery*, *24*, 247–287.

Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., & Trasarti, R. (2006). Conquest: a constraint-based querying system for exploratory pattern discovery. *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on* (pp. 159–159).

Boulicaut, J.-F., & Masson, C. (2005). Data mining query languages. *Data Mining and Knowledge Discovery Handbook*, 715–726.

Braga, D., Campi, A., Ceri, S., Klemettinen, M., & Lanzi, P. (2003). Discovering interesting information in xml data with association rules. *Proceedings of the 2003 ACM symposium on Applied computing* (pp. 450–454).

De Raedt, L. (2002a). Data mining as constraint logic programming. *Computational Logic: Logic Programming and Beyond*, 113–125.

De Raedt, L. (2002b). A perspective on inductive databases. *ACM SIGKDD Explorations Newsletter*, *4*, 69–77.

Džeroski, S. (2011). Inductive databases and constraint-based data mining. *Formal Concept Analysis*, 1–17.

Giannotti, F., Manco, G., & Turini, F. (2004). Specifying mining algorithms with iterative user-defined aggregates. *Knowledge and Data Engineering, IEEE Transactions on*, *16*, 1232–1246.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*, 10–18.

Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O., et al. (1996). DMQL: A data mining query language for relational databases. *Proc. 1996 SiGMOD* (pp. 27–34).

Helma, C., Kramer, S., & De Raedt, L. (2002). The molecular feature miner MolFea. *Proceedings of the Beilstein-Institut Workshop.*

Imieliński, T., & Mannila, H. (1996). A database perspective on knowledge discovery. *Communications of the ACM*, *39*, 58–64.

Imieliński, T., & Virmani, A. (1999). MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, *3*, 373–408.

Meo, R., Psaila, G., & Ceri, S. (1998). An extension to sql for mining association rules. *Data Mining and Knowledge Discovery*, *2*, 195–224.

Microsoft (2012). Data mining extensions (DMX). `http://msdn.microsoft.com/en-us/library/ms132058.aspx`. Accessed on 30-April-2013.

Romei, A., Ruggieri, S., & Turini, F. (2006). KDDML: a middleware language and system for knowledge discovery in databases. *Data & Knowledge Engineering*, *57*, 179–220.

Romei, A., & Turini, F. (2011). Inductive database languages: requirements and examples. *Knowledge and Information Systems*, *26*, 351–384.

Wagstaff, K. L. (2002). *Intelligent clustering with instance-level constraints.* Doctoral dissertation, Cornell University.

Wang, H., & Zaniolo, C. (2003). Atlas: A native extension of sql for data mining. *Proceedings of the 3rd SIAM International Conference on Data Mining.*

Wicker, J., Richter, L., Kessler, K., & Kramer, S. (2008). SINDBAD and SiQL: An inductive database and query language in the relational model. *Machine Learning and Knowledge Discovery in Databases*, 690–694.