International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-3 Issue-12 September-2013

Horizontal Aggregation in SQL to Prepare Dataset for Generation of Decision Tree using C4.5 Algorithm in WEKA

Mayur N. Agrawal[1], Ankush M. Mahajan[2], C.D. Badgujar[3], Hemant P. Mande[4,] Gireesh Dixit[5]

Abstract

In this paper we are taking a Database Table, a database is an organized collection of data. We are performing a Horizontal Aggregation on that particular Database Table; there are three methods from which we are performing Horizontal Aggregation. These three methods give us the output as the dataset of that particular database table. A Dataset is a collection of data, usually presented in tabular form. By providing these dataset as an input to the C 4.5 algorithm in WEKA, we are generating Decision tree for that database table. Three methods for Horizontal aggregation are SPJ, PIVOT and CASE. C4.5 is the Decision tree Generation Algorithm which generates Decision tree.

Keywords

Aggregation, PIVOT, SPJ, CASE, Dataset

1. Introduction

A Dataset are the most important part For data Mining and Preparing data set time consuming task as it requires many complex SQL queries, joining tables and aggregating columns. Current Aggregations have limitations to prepare data sets because in Current Aggregation they return one column per aggregated group proposed method is to generate SQL code to return aggregated columns in a horizontal tabular layout, returning a set of numbers instead of one number per row. Horizontal Aggregation build data sets with a horizontal denormalized which is the standard layout required by most data mining algorithms. Three methods are there for Horizontal Aggregation i.e SPJ, CASE, PIVOT. One is discussing in detail (SPJ).

2. Horizontal Aggregation

For data mining analysis datasets are required and preparing datasets in data mining is the most time consuming task. We proposed an abstract, but minimal, extension to SQL standard aggregate functions to compute horizontal aggregations which just requires specifying subgrouping columns inside the aggregation function call, we proposed three query evaluation methods.

i) SPJ relies on standard relational operators.
ii) CASE relies on the SQL CASE construct.
iii) PIVOT uses a built-in operator in a commercial DBMS that is not widely available.

The SPJ method is based on select, project and joins (SPJ) queries. The CASE method is our most important contribution. It is in general the most efficient evaluation method and it has wide applicability since it can be programmed combining GROUP-BY and CASE statements. We have explained it is not possible to evaluate horizontal aggregations using standard SQL without either joins or ”case” constructs using standard SQL operators.

Our proposed horizontal aggregations can be used as a database method to automatically generate efficient SQL queries with three sets of parameters: grouping columns, subgrouping columns and aggregated column. The fact that the output horizontal columns are not available when the query is parsed makes its evaluation through standard SQL mechanisms infeasible. Our experiments with large tables show our proposed horizontal aggregations evaluated with the CASE method have similar performance to the built-in PIVOT operator. We believe this is remarkable since our proposal is based on generating SQL code and not on internally modifying the query optimizer. Both CASE and PIVOT evaluation methods are significantly faster than the SPJ method.

3. SPJ Method

The SPJ method is based on relational operators only. The basic idea is to create one table with a vertical aggregation for each result column, and then join all those tables to produce FH. We aggregate from F into d projected tables with the Select- Project-Join-Aggregation queries. Each table FI corresponds to one sub grouping combination and has {L1, . . ., Ln} as primary key and an aggregation on A as the only non-key column. It is necessary to introduce an additional table F0 that will be Outer joined with

projected tables to get a complete result set. We propose two basic sub strategies to compute FH.
The first one directly aggregates from F. The second one computes the equivalent vertical aggregation in a temporary table FV grouping by (L1, . . ., Ln), (R1, . . .,Rm). Then horizontal aggregations can be instead computed from FV, which is a compressed version of F, since standard aggregations are distributive. In a horizontal aggregation there are four input parameters to generate SQL code:-

(i) The input table F
(ii) The list of GROUP BY columns L1, …… ,Ln
(iii) The column to aggregate (A) and
(iv) The list of transposing columns R1, … ,Rk.

We extend standard SQL aggregate functions with a transposing BY clause followed by a list of columns to produce a horizontal set of numbers instead of one number.
Syntax for SPJ Method:-
SELECT (L1,L2 …… ,Ln), H(A BY R1, B BY R2.. ,Rm)
FROM F
GROUP BY (L1,L2 …… ,Ln) ;
In SPJ Method there is a use of Left Outer Join, the left outer join is performed in between two tables i.e. left table and Right table. Common fields of both the tables are returned and uncommon fields of left column are also returned. This is the concept of left outer join, which joins the table in this manner. Weather database we are taking as our input Database Table with columns outlook, temperature, humidity and play.

WEATHER

SR	OUTLOOK	TEMP	HUM	PLAY
1	OUTLOOK	NULL	NULL	PLAY
2	SUNNY	85	85	NO
3	SUNNY	80	90	NO
4	OVERCAST	83	86	YES
5	RAINY	70	96	YES
6	RAINY	68	80	YES
7	RAINY	65	70	NO
8	OVERCAST	64	65	YES
9	SUNNY	72	95	NO
10	SUNNY	69	70	YES
11	RAINY	75	80	YES
12	SUNNY	75	70	YES
13	OVERCAST	72	90	YES
14	OVERCAST	81	75	YES
15	RAINY	71	91	NO

Figure 1:- Database Table

In our project we are taking the Database table of Weather as Shown Above. If we want to apply SPJ method on this Database table then the following query is fired.

Query: -

```
SELECT  F1.temp as tem_Play_yes, F2.temp  as temp_Play_no , F1.hum as hum_Play_yes, F2.hum as hum_Play_no , F1.outlook from
(SELECT outlook, avg (temperature) AS temp, avg (humidity)as hum FROM weather
WHERE play='yes' GROUP BY outlook) F1 left outer join
(SELECT outlook, avg (temperature) AS temp, avg (humidity)as hum FROM weather
WHERE play='No' GROUP BY outlook ) F2 on F1.outlook=F2.outlook ;
```

Now will see the execution of this query in detail by breaking down the query into sub-query. Let's see the first sub-query

SELECT outlook , avg (temperature)AS temp, avg(humidity)as hum FROM weather
WHERE play='yes' GROUP BY outlook

WEATHER

SR NO	OUTLOOK	TEMP	HUM	PLAY
1	OUTLOOK	NULL	NULL	PLAY
2	SUNNY	85	85	NO
3	SUNNY	80	90	NO
4	OVERCAST	83	86	YES
5	RAINY	70	96	YES
6	RAINY	68	80	YES
7	RAINY	65	70	NO
8	OVERCAST	64	65	YES
9	SUNNY	72	95	NO
10	SUNNY	69	70	YES
11	RAINY	75	80	YES
12	SUNNY	75	70	YES
13	OVERCAST	72	90	YES
14	OVERCAST	81	75	YES
15	RAINY	71	91	NO

⇨

F1

OUTLOOK	TEMP	HUM
OVERCAST	75	79
RAINY	71	85
SUNNY	72	70

Figure 2:-Creation of F1 Table

In this part of query selection of Outlook, average of temperature, average of humidity is selected from Weather database when play is yes. Group by outlook means all the value for same outlook having play = yes are Aggregated.
We will see one example, when outlook is overcast see for play when play is yes. Aggregate all the temperature value for this condition looking at this database we have,
(83+64+72+81)/ 4 =75
We get the result as 75 this is displayed in table in Temp column.
In the same way when outlook is overcast, see for play when play= yes. Aggregate all the Humidity value for this condition looking at this database table we have,
(86+65+90+75)/4 =79

We get the result as 79 displayed in the table in Hum column. This contributes a single row for the output generated in figure 2 given below. This procedure is repeated for all outlook. The resulted table that we have got is F1 table which we use later for performing join.

SELECT outlook ,avg(temperature)AS temp, avg(humidity)as hum FROM weather
WHERE play='No' GROUP BY outlook

WEATHER

SR NO	OUTLOOK	TEMP	HUM	PLAY
1	OUTLOOK	NULL	NULL	PLAY
2	SUNNY	85	85	NO
3	SUNNY	80	90	NO
4	OVERCAST	83	86	YES
5	RAINY	70	96	YES
6	RAINY	68	80	YES
7	RAINY	65	70	NO
8	OVERCAST	64	65	YES
9	SUNNY	72	95	NO
10	SUNNY	69	70	YES
11	RAINY	75	80	YES
12	SUNNY	75	70	YES
13	OVERCAST	72	90	YES
14	OVERCAST	81	75	YES
15	RAINY	71	91	NO

⇨

F2

OUTLOOK	TEMP	HUM
RAINY	68	80
SUNNY	79	90

Figure 3:-Creation of F2 Table

In this part of query selection of Outlook, average of temperature, average of humidity is selected from Weather database when play is No. Group by outlook means all the value for same outlook having play = No are Aggregated. We will see one example, when outlook is Rainy see for play when play is No. Aggregate all the temperature value for this condition looking at this database we have,
(65+71)/2 =68
We get the result as 68 this is displayed in table in Temp Column.
In the same way when outlook is Rainy, see for play when play= No. Aggregate all the Humidity value for this condition looking at this database table we have,
(70+91)/ 2 =80. We get the result as 80 displayed in the table in Hum column. This contributes a single row for the output generated in figure 3 given below. This procedure is repeated for all outlooks. The resulted table that we have got is F2 table which we use later for performing join.

As we have got F1 and F2 table So another part of Query is to join F1 and F2 to get us the result aggregated i.e. horizontal aggregation of that weather database table. Resulted table is our table which is aggregated horizontally, shown in figure 4.

F1

Outlook	Temp	Hum
overcast	75	79
Rainy	71	85
Sunny	72	70

F2

Outlook	Temp	Hum
Rainy	68	80
Sunny	79	90

⇩

Temp_Play_Yes	Temp_Play_No	Hum_Play_Yes	Hum_Play_No	Outlook
75	NULL	79	NULL	Overcast
71	68	85	80	Rainy
72	79	70	90	Sunny

Figure 4:-Horizontally Aggregated Table using SPJ method

Left outer join is performed between F1 and F2. Left outer join output the common rows of both table and also the uncommon fields of left table. Here, left table is F1 and F2 is right table so it result contains the common outlook of F1 and F2 i.e. Rainy and Sunny, also the uncommon outlook of F1 i.e. overcast. F1 table is for the condition for Play=”yes” and F2 table is for play=”No”.

Let’s look into resulted output table temp_play_yes column when outlook is overcast value is taken from temp column of F1 table i.e. 75, in the same way rainy and sunny outlook are carried out. Now look at Temp_play_No column when outlook is overcast value is taken from temperature column of F2 in this case outlook overcast is not present so a NULL value is displayed, for outlook rainy and sunny respective values are taken from Temp_play_no. Same procedure is done for humidity case i.e. hum_play_yes and hum_play_No. Here we have Got the Horizontal Aggregation of our Weather database table.

4. Case Method

The case statement returns a value selected from a set of values based on Boolean expressions. For this method we use the ”case” programming construct available in SQL, where each non key value is given by a function that returns a number based on some conjunction of conditions.

Query For CASE Method: -

SELECT
avg(CASE WHEN play='yes'THEN temperature ELSE null END)as temp_play_yes,
avg(CASE WHEN play='no' THEN temperature ELSE null END)as temp_play_no,
avg(CASE WHEN play='yes'THEN humidity ELSE null END)as hum_play_yes,
avg(CASE WHEN play='no' THEN humidity ELSE null END)as hum_play_no,

outlook FROM weather GROUP BY outlook;

5. Pivot Method

Pivot is complementary data manipulation operators that modify the role of rows and columns in a relational table. In Pivot it transforms a series of rows into a series of fewer rows with additional columns. Pivoting refers to transposing columns data into rows.

Query for PIVOT Method: -

Select [temp_play_yes] as temp_play_yes, [temp_play_no] as temp_play_no , [hum_play_yes] as hum_play_yes, [hum_play_no] as hum_play_no, Table1.outlook from
(SELECT [yes] as temp_play_yes, [no] as temp_play_no , outlook FROM (SELECT outlook, play, temperature FROM weather) t1 PIVOT(avg (temperature) FOR play IN ([yes] , [no])) as t2) Table1 left outer join
(SELECT [yes]as hum_play_yes, [no]as hum_play_no, outlook FROM (SELECT outlook, play, humidity FROM weather) t1 PIVOT(avg (humidity) FOR play IN ([yes] , [no])) as t2) Table2 on Table1.outlook=Table2.outlook;

All the three method of Horizontal Aggregation have there results and depending on their results Datasets are created. Created dataset's attributes are depended upon the attributes of the result all the three Horizontal Aggregation methods. Creation of Datasets is Shown in Next point.

6. Creation of Dataset

In our project let us see how datasets are created using SPJ Practically. Sets given below shows weather Database created in Microsoft SQL Server Management Studio 2008. The Microsoft SQL Server 2008 Database Engine is a service for storing and processing data in either a relational (tabular) format or as XML documents.Datasets are prepared from the output of all these three method, so by applying these three methods on our Weather database we get the Dataset in .arff file format.

ARFF files have two distinct sections. The first section is the **Header** information, and the second section is **Data** information. The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.

@**relation** abc
@**attribute** Team_Play_no real
@**attribute** Team_Play_yes real
@**attribute** Humidity_play_no real
@**attribute** Humidity_play_yes real
@**attribute** Season {sunny, overcast, rainy, outlook}
@**Data**

75, 0, 79, 0	overcast
71, 68, 85, 80	rainy
72, 79, 70, 90	sunny
0, 0, 0, 0	outlook
75, 0, 79, 0	overcast
71, 68, 85, 80	rainy
72, 79, 70, 90	sunny
0, 0, 0, 0	outlook
75, 0, 79, 0	overcast
71, 68, 85, 80	rainy
72, 79, 70, 90	sunny

This is the Dataset created by three methods, this dataset is given as an input to C 4.5 algorithm using WEKA to generate Decision tree. On that dataset Entropy and Information gain Operation are performed and a suitable decision rules are generated. Based on that rule Decision Tree is generated. Entropy of every attribute is calculated. In WEKA algorithm is implemented and linked with java file.

On the basis of these Datasets Generated from the three methods of Horizontal Aggregation we will generate Decision Tree. To generate Decision Tree we Required an Algorithm. So C4.5 Algorithm is applied to these Datasets to generate Decision tree. C4,5 Algorithm is Implemented in WEKA TOOL.

C4.5 is an algorithm which generates Univariate decision tree. It is the extension of Iterative Dichotomiser 3 (ID3) algorithm which is used to find simple decision trees. C4.5 is also known as Statistical Classifier because its decision trees can be used for classification purpose. C4.5 builds decision trees from a set of training data using the concept of entropy and information gain. The training dataset consists of various training samples which are characterized by large number of features and also consists of target classes.

The WEKA (Waikato Environment for Knowledge Analysis) project aims to provide a comprehensive collection of machine learning algorithms and data preprocessing tools to researchers and practitioners. It allows users to quickly try out and compare different machine learning methods on new data sets. Its modular, extensible architecture allows sophisticated data mining processes to be built up from the wide collection of base learning algorithms and tools provided

Algorithm

1. Check for base cases.
2. For each Attribute A calculate
a) Normalized information gain from splitting on Attribute A.
3. Select the best attribute A that has highest information gain.
4. Create a decision node that splits on best of A as root node.
5. Recurs on the sub lists obtained by splitting on best of A, and add those nodes as children node.

Entropy

It is a measure in the information theory, which characterizes the impurity of an arbitrary collection of examples. If the target attribute takes on *c* different values, then the entropy S relative to this c-wise classification is defined as

$$\text{Entropy}(s) = \sum_{t=1}^{c} -pi \log 2\, pi$$

Information Gain

It measures the expected reduction in entropy by partitioning the examples according to this attribute. The information gain, Gain (T, X) of an attribute A, relative to the collection of examples T, is defined as:
Information Gain = Entropy (T) – Entropy (T, X)
Based on these Calculation and Using C4.5 algorithm in WEKA Decision Tree is Generated and it is shown in fig below:-

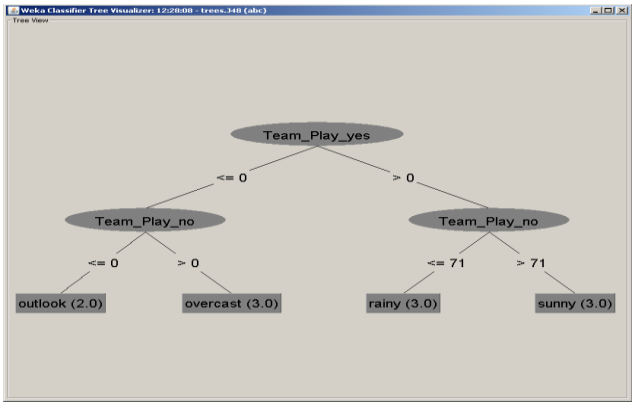

Figure 5: - Generating Decision tree using Dataset and C 4.5 Algorithm.

Decision Rule:

1. If ((Team_Play_yes <= 0) & (Team_Play_no <= 0)) => : outlook (2.0)
2. If ((Team_Play_yes <= 0) & (Team_Play_no > 0)) =>: overcast (3.0)
3. If((Team_Play_yes > 0) & (Team_Play_no <= 71)) => : rainy (3.0)
4. If((Team_Play_yes > 0) & (Team_Play_no > 71)) => : sunny (3.0)

7. Comparing Results

Suppose we have taken any of the dataset and on that dataset apply both algorithm in WEKA. We can see that time required for building model in ID3 is 2.63 sec and time required for building same model in C4.5 or J48 algorithm is 0.27 sec. from these we can say that by using C4.5 algorithm in WEKA time reqired for building model is comparatively very less as time required in ID3 algorithm.

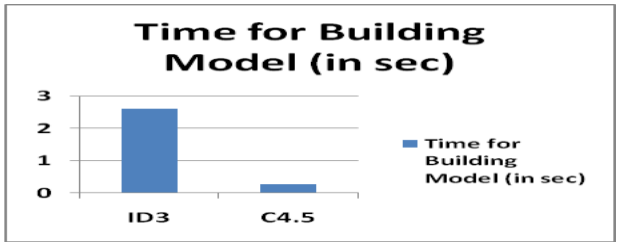

Figure 6 :- Resulted Graph

8. Conclusion

In this project we have seen how Datasets are created using methods of Horizontal Aggregation. Horizontal aggregations represent an extended form of traditional SQL Aggregations, which return a set of values in a horizontal layout, instead of a single value per row. We provide a more efficient, better integrated and more secure solution compared to external data mining tools. With the execution of methods of Horizontal Aggregation Datasets are created and these Datasets are used to generate Decision Tree. Decision Tree is generated using C4.5 algorithm in WEKA. Model built time of C4.5 is less than that of ID3. Memory used for storing C 4.5 Dataset is comparatively less than ID3. So, use of C4.5 algorithm will help us to reduce time required for building model of a particular dataset and also it require less memory to store its Datasets.

9. Future Scope

As we have seen Generation of Decision Tree using C4.5 algorithm, we can approach towards C5.0 which is much more efficient than C4.5. In Future, we can

design C5.0 algorithm to generate Decision Tree. We can find new technique from which we can perform Horizontal Aggregation effectively. Reducing the number of comparisons needed to compute horizontal aggregations may lead to changing the algorithm to parse and evaluate a set of aggregations when they are combined with "case" statements with disjoint conditions.

References

[1] C. Ordonez and Z. Chen. "Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining analysis", IEEE Transactions on Knowledge and Data Engineering (TKDE), 24(4), 2012.

[2] C. Cunningham, G. Graefe, and C.A. Galindo-Legaria. "PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS", In Proc. VLDB Conference, pages 998–1009, 2004.

[3] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. "Data cube: A relational aggregation operator generalizing group-by, cross-tab and subtotal", In ICDE Conference, pages 152–159, 1996.

[4] C. Ordonez. "Horizontal aggregations for building tabular data sets", In Proc. ACM SIGMOD Data Mining and Knowledge Discovery Workshop, pages 35–42, 2004.

[5] C. Ordonez. "Vertical and horizontal percentage aggregations", In Proc. ACM SIGMOD Conference, pages 866–871, 2004.

[6] C. Ordonez. "Integrating K-means clustering with a relational DBMS using SQL", IEEE Transactions on Knowledge and Data Engineering (TKDE), 18(2):188–201, 2006.

[7] C.Ordonez."Data set preprocessing and transformation in a database system", Intelligent Data Analysis (IDA), 15(4), 2011.

[8] C. Ordonez and S. Pitchaimalai. "Bayesian classifiers programmed in SQL", IEEE Transactions on Knowledge and Data Engineering (TKDE), 22(1):139–144, 2010.

[9] Sarawagi, Sunita, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. Vol. 27, no. 2. ACM, 1998.

[10] G. Graefe, U. Fayyad, and S. Chaudhuri. "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases", Proceedings of The Fourth International Conference on Knowledge Discovery and Data Mining, 1998, pages 204-208.

[11] Jens-Peter Dittrich, Donald Kossmann and Alexander "Bridging the Gap between OLAP and SQL", Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005 .

[12] S. Aishwarya, S. Ramadevi. "Multi dimensionalised Aggregation in Horizontal Dataset using Analysis services", Journal IJETAE, ISSN: 2250-2459, Jan 2013.

[13] Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA data mining software: an update." ACM SIGKDD explorations newsletter 11, no. 1 (2009): 10-18.

Mr. Mayur N Agrawal has completed his BE Computer and Currently Doing his M.E in Computer science form North Maharashtra University, Jalgaon. Has 3 International Publications. Recent Publication: - IJESRT Paper name: - Horizontal Aggregation in SQL to prepare dataset for Data Mining Analysis .

Mr. Ankush Mahajan has completed his BE Computer and Currently Doing his M.E in Computer science from North Maharashtra University, Jalgaon.

Mr. Chandrashekhar D. Badgujar has completed his M. Tech in Computer Engineering from NMIMS University, Mumbai in the year 2010. He has 5 years teaching and 2 years industry experience. His research interests include image processing and web mining. He has 10+ publications on his name till date.

Mr. Hemant Mande has completed his BE Computer and Currently Doing his M.E in Computer science form North Maharashtra University, Jalgaon.