CS2980:
Model-based Semantic Compression in Database
Project Report

Zhang, Minrui
mrzhang@cs.brown.edu

Advisor: Prof. Ugur Çetintemel

May 7, 2014

1 Introduction and Inspiration

We are now in big data era, advances in data collection and management technologies have led to large databases. For example, domains such as Medicine, Biology, Music and experimental sciences in general, are all characterized by large data sequences.

Considering the amount of space the big data required and the amount of IO required to fetch the big data, data compression has become an efficient way to reduce the database size and improve the query performance. Data compression can be lossless or lossy. Lossless compression aims to provide the exact query answer, whereas lossy compression aims to minimize the amount of data that needs to be held. Obviously, lossless compression needs more computation and metadata to preserve the entire information. Also, providing an exact answer to a complex query in large data recording environment can take minutes, or even hours, while users are more prone to searching for certain patterns of behavior that they approximately visualize, rather than for specific values. Thus, lossy compression become popular in analytical use.

Semantic compression is a kind of lossy compression. Since semantic compression account for and exploit both the meanings and dynamic ranges of error individual attributes; and existing data dependencies and correlations among attributes in the table [3], it is very effective for table-data compression. In this project, columns are compressed based on their associations with other columns by using predictive models.

Weka is a comprehensive suite of Java class libraries that implement many machine learning and data mining algorithms. It contains implementations of many algorithms for classification and numeric prediction [7], which can help us to build predictive model for target columns to achieve semantic compression.

Inspired by semantic compression and Weka, this project implemented a system that columns are compressed based on their associations with other columns by using Weka models. The system will predict the result rather than read it directly from database when given a SQL query together with an error tolerance, therefore to achieve higher performance improvements in speed.

2 Related Work

There exists a lot of algorithms to sample or compress databases, no matter basing on rows or columns. The reason why we need compression is that it improves performance by reducing the amount of IO required to fetch data from storage, whether it be disk IO or network IO. For row based compression, the goal is to find minimum number of rows as a sample that can represent the whole table. For column based compression, it typically needs to compress columns basing on the induction from the value of other columns.

Abadi et al. [1] has the lossless column-based compression algorithm using methods such as dictionary encoding and run-length encoding. Such algotirhms can work very well on column-oriented database systems like C-Store [2]. Jagadish et al. [4] presented fascicles which performs row-wise compression, extracting certain patterns of the table and finding groups of rows that have a limited amount of variation. Jagadish et al. [5] also presented itcompress which based on fascicles but achieve better compression ratio by running the row-wise compression algorithms iteratively. VC-dimension [6] is an algorithm to estimate query result selectivities based on the query complexity. A lot of row-wise compression technologies using VC-dimension to extract representative samples from the original dataset.

There are also technologies combine both row-wise and colum-wise compression. For example, Spartan [3] uses Bayesian networks to train the dataset and find the columns can be compressed, then clustering the resulting rows to achieve better compression. This project is also inspired by the way Spartan compress the dataset, since it is transparent to any nowadays DBMS, thus the query engine can choose to run queries on compressed dataset under a certain error threshold to achieve higher performance improvements.

3 Project Overview

3.1 Generating Weka Models

In this project, Weka is the main tool to help us to do the training and classifying on the original data and produce the metadata which we call them models. Models are basically the outcomes of iteratively studying the relationship between each column and the rest columns, which contain all the necessary information of the data for the use to do the prediction for one column using some of the rest columns. The machine learning algorithms we are using for data classification called "Classifiers". After the classification, we will have different models based on different classifiers.

Trees are the main classifiers we are using to produce Weka models for numeric prediction. Trees used for numeric prediction resemble normal decision trees, with the exception of storing at each leaf a class value that represents the average value of instances that reach the leaf (regression tree), or a linear regression model that predicts the class value of instances reaching the leaf (model tree). In our project, we are using both regression tree (REPTree) and model tree (M5P) to produce Weka models.

REPTree generates regression trees based on information gain as the splitting principle, using reduces-error pruning, and sorts values for numeric attributes once. M5P generates M5 model trees, combining a conventional decision tree with the incorporation of linear regression functions at the leaves.

```
REPTree
============

P14p9 < 0.11
|   P11p4 < 0.02
|   |   P15p1 < 0.72 : 0.64 (10/0.01) [5/0.01]
|   |   P15p1 >= 0.72 : 0.53 (21/0.01) [14/0]
|   P11p4 >= 0.02
|   |   P14p9 < 0.08
|   |   |   H2p2 < 0.33 : 0.5 (918/0) [474/0]
|   |   |   H2p2 >= 0.33 : 0.53 (59/0) [21/0]
|   |   P14p9 >= 0.08 : 0.49 (730/0) [351/0]
```

Figure 1: REPTree

Figure 1 and Figure 2 are the result models after using REPTree and M5P to classify the house16H data(price of the house in the region based on demographic composition and a state of housing market in the region) respectively. From the figures we can learn that the models contain the information of the relation between one column and rest columns: 1) What is the dependent columns of this column; 2) How to predict this column by given its dependent columns. One column has one model. In this project, models will be stored in Mysql database in a relation of "column name", "model" and "dependent columns' name", which made the job easier to get the its model and dependencies when given a column name. A column's model and its dependencies are the key factors to predict the column. After we got one column's model and its dependent columns' data according to the dependent columns' name, we can do the prediction for this column.

While saving models to the database, we also store the metadata information about the given table we call it "headers" which contains the information that can tell us, for example how many rows or columns in the table, or whether a column contains numeric data or nominal data.

```
M5 pruned model tree:
(using smoothed linear models)

P14p9 <= 0.131 :
|   P14p9 <= 0.078 :
|   |   P1 <= 364.5 : LM1 (277/95.333%)
|   |   P1 >  364.5 :
|   |   |   P14p9 <= 0.042 : LM2 (240/17.045%)
|   |   |   P14p9 >  0.042 :
|   |   |   |   H8p2 <= 0.022 :
|   |   |   |   |   H2p2 <= 0.085 : LM3 (420/22.732%)
|   |   |   |   |   H2p2 >  0.085 :
|   |   |   |   |   |   price <= 67550 : LM4 (112/26.373%)
|   |   |   |   |   |   price >  67550 : LM5 (64/51.154%)
|   |   |   |   H8p2 >  0.022 : LM6 (248/17.46%)
```

```
LM num: 1
P11p4 =
        -0.0002 * P1
        - 0.4205 * P5p1
        - 0.3003 * P6p2
        + 0.0489 * P14p9
        - 0.3472 * P15p1
        - 0.0277 * P15p3
        + 0.0048 * P16p2
        - 0.0321 * P18p2
        - 0.7882 * P27p4
        + 0.0814 * H2p2
        + 0.3132 * H8p2
        - 0.0657 * H13p1
        - 0.0522 * H18pA
        + 0 * price
        + 0.7022
```

Figure 2: M5P

3.2 Prediction Using Weka Models

As we can see, it is better to use column-oriented database to read a column's dependent columns' data. Therefore, this project built on top of a mock column-oriented database on our hard drive that using one ARFF file to simulate one column. The predicted outputs also be store as ARFF files on the hard drive. In this project, all predictions are completed by a family of overloaded functions called "decompress".

3.2.1 Predicting an Entire Column

When predicting an entire column, we need the name of the table and the name of the column we want to predict. First, we use the table name to get the headers of the table. Then we use the table name and the column name to get the column's model and its dependent columns' name from the database. After that, we create an empty Weka instance. Weka instance is a Weka defined container to store a row of data of a table. Instance is one of the important argument for doing the prediction.

Since we have got the dependent columns' name, we can read the dependent column data according to the name from our mock column-oriented database. After that, we will fill the empty instance with the dependent columns' data according to their original indices in the table. The indices' information can be achieved from the headers.

Once we filled a instance, we can predict the column's value at the corresponding row with the help of the instance and the column's model. We will call different Weka functions to do the prediction based on which algorithms we used to generate the model, in our case, REPtree or M5P. Basically, the functions take the instance and the column's model as the arguments. We iteratively predict all the attributes in the column. The information from the headers helps us to control the iteration.

Let's take a look at how well our prediction goes. In the project, we have a tool to measure how much the predicted value deviated from the original value.

```
1: 0.10141763646248088
2: 0.43036044735802476
3: 0.27927611229965155
4: 0.19256499424561416
5: 0.09672963116182567
6: 1.0296629651310087
7: 0.1329096077108434
8: 0.04421730137254894
9: 0.4166374615507246
10: 0.26653445556187766
11: 0.00262106351181097

Statistics:

0.020935744382022472 [0% - 1%]
0.07957338483146068 [1% - 5%]
0.09607619382022473 [5% - 10%]
0.25768086286516854 [10% - 25%]
0.27036516853932585 [25% - 50%]
0.18600772471910113 [50% - 100%]
0.08936095505617977 [100% - infinity]
```

Figure 3: Error Percent for Column "price" in Table "house16H"

The Figure 3 shows the first 11 rows' error percent and the statistics of the entire column for the column "price" in the table "house16H". We can see that different rows have different percent of deviation from the original value. Take the example of row 11, the result is very accurate that it only deviates about 0.26% from the original value. Whereas, for row 6, it deviates from the original value more than 100%. From the statistics, we can find that, for example, more than 2% of rows of predicted value deviate from the original value within 1%, and more than 25% of rows of predicted value deviate from the original value between 10% and 25%.

The Figure 4 shows the some of rows' error percent and the statistics of the entire column for the column "HRHHID2" in the table "cps"(Current Population Survey). We can easily tell that the results are much more accurate. For row 32 and 33, the predicted results have even no deviation from the original ones. Therefore, from the statistics, we can see that more the 95% of rows of predicted value deviate from the original value within 1%.

The reason why the second prediction is more accurate is because the column "HRHHID2" has stronger correlation with other columns. Although all the data types of the columns in the "cps" are numeric, the value of each row in a column

are in a finite range. Whereas, for the table "house16H", all the values are arbitrary double values. Thus, it is easier to study the relations between the columns in table "cps" when generating the models. This also give us a hint that in the future work, when dealing with nominal values like "ID", or boolean values "0" or "1", or enum values represented by numbers, we can first transfer them into numeric value then do the prediction in order to achieving better accuracy.

This kind of prediction supports generating values for an entire column that has been removed from a dataset. Therefore, the database only need to store the dependent columns that will lead to the decreased total size of the data we need to store in the database, and on the other hand, will save the time for reading the entire column from the database.

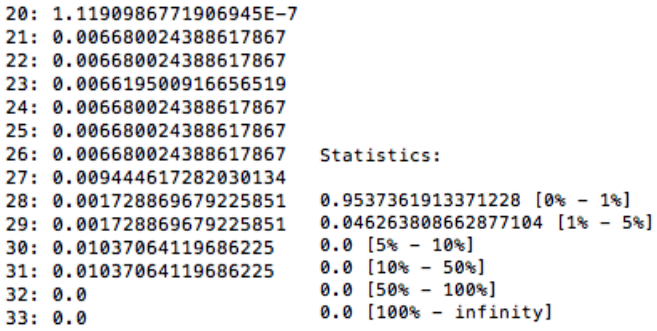

Figure 4: Error Percent for Column "HRHHID2" in Table "cps"

3.2.2 Predicting an Attribute with Error Tolerance

Since we can calculate how much the predicted value has deviated from the original one, we can let users run their queries with error tolerance. If the predicted result within the error tolerance, we return the predicted result to the user, otherwise run the normal SQL.

This kind of prediction is more practical in dealing with queries with WHERE clause and the columns in WHERE conditions are the subset of the dependent columns. In this way, we can read less data from the database, or even nothing, since we can take users' inputs to predict the result they want when all the columns in WHERE conditions are the dependent columns.

We need store a hash-like structure, so that we can tell whether the result within the error tolerance when given WHERE conditions. This process which completed by hash store API, is almost as same as the previous section when predicting the entire column with one difference that we also need to read original data of the column which we want to predict, so that we can calculate how much the predicted data deviated from the original one. First, we maintain 4 hash sets which represent 4 different error tolerance: 1%, 1% to 5%, 5% to 10% and 10% to 25%. Then, during the iteration, for each row, we maintain a hash map to hold the dependent columns' name and corresponding value. After we predicted the attribute the user want and calculated how much percent deviation from the original one, we store the hash map into corresponding hash set, if the error percentage within the error tolerance that represented by one of the hash set. The hash store process runs during offline session.

The project also implemented a hash lookup API, which take a hash map and an error tolerance as inputs and return TRUE or FALSE to denote whether the predicted value will be within the error tolerance or not. The columns' name and

corresponding value after the WHERE clause have been parsed into a hash map. If one of the hash set that the error tolerance it represents smaller than the given error tolerance that contains the given hash map, it means the predicted result will be satisfying the error tolerance, and it will return TRUE, otherwise it will return FALSE. The system will then do the prediction if the hash lookup API returns TRUE, otherwise do the normal SQL query. The hash lookup process and predicting process run during the online session.

3.3 Experiments and Analyses

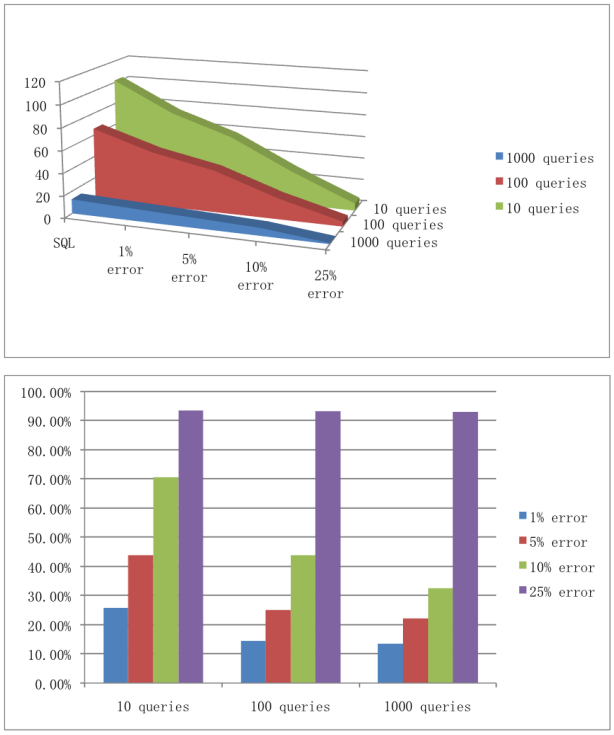

Figure 5: Column "b1x", Dataset "bank32H"

The experiments in this project aims to study and analyze the performance when running user queries with different error tolerances. The experiments can also show how the quantity of queries will affect the performance. The experiments run on different datasets, some of them have strong relationships between columns, some are not. All datasets contain numeric columns only.

The test queries in the form like "SELECT [column name] FROM [table name] WHERE [column A = A', column B = B', ...]". During the offline session, the column that the user want to select has been processed by hash store API, so that we already know how much percent each attribute in the column has deviated from the original value. This helps us to control the composition of the test queries, in other words, we can decide how many test queries that produce predicted result that has error percentage are smaller than the user giving tolerance and how many are bigger. When running the test, the query will first reach the hash lookup API to see whether the predicted result is smaller than the error tolerance which given by the user. If it is, do the prediction and return the predicted value, otherwise run the normal SQL.

The error tolerances are from no error tolerance(normal SQL) to 25% error tolerance. The quantity of queries per test are from 10 queries to 1000 queries.

The running time is average running time per query, and the time unit is milliseconds. From Figure 5 to Figure 8, for each figure, the first chart shows how much time spent when running different number of queries under different error tolerance; the second chart shows the improvement over the baseline(no error tolerance) as a percentage for different error levels.

From the first chart of Figure 5 to Figure 8, we can easily tell that the more error we allowed, the less average running time we will get. The more error allowed means the more results will be predicted rather than directly read from the database. This is the trade off we want to see, as we talked about earlier, we would like to allow some error to achieve higher performance improvements in speed.

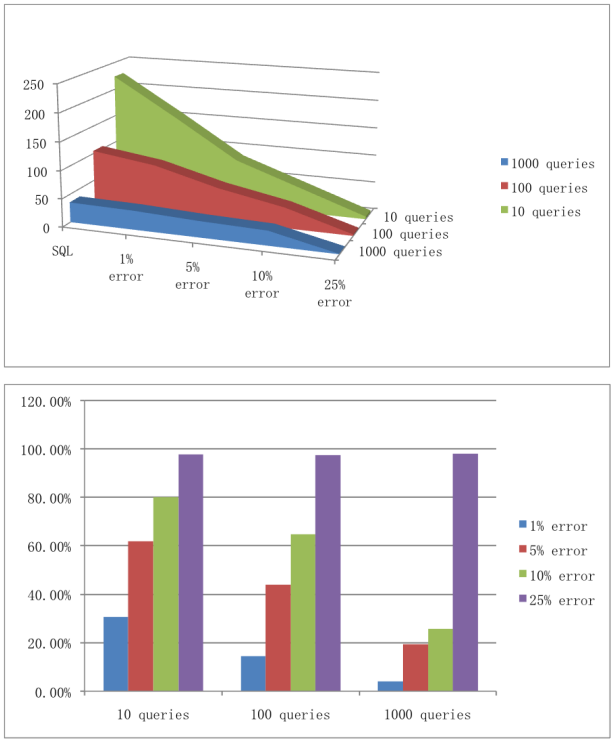

Figure 6: Column "PEERNHRO", Dataset "cps"

On the other hand, we can see that the more queries we run, the less average running time we will get. This gives us confidence that this kind of semantic compression use in database will be more useful in big data circumstance with enormous I/O requests.

Compare Figure 6 to Figure 5, we can see the average running time for selecting column "PEERNHRO" in dataset "cps" is longer than selecting column "b1x" in dataset "bank32H". That is because column "PEERNHRO" in dataset "cps" has stronger connect with other columns, thus the column owns more dependent columns.

The second chart of Figure 5 to Figure 8 show that the more error we allowed, the higher percent of improvements in average time cost will be gained. For error bound 25%, the percentage of improvements in speed is around 90%. Especially for column "PEERNHRO" in dataset "cps", the improvements are almost reach 100% when allowed 25% error.

The great improvements on dataset "cps" also gives us a hint when dealing with numeric attributes have a finite range of values. First, this kind of

datasets have strong correlations between columns that lead to great performance gains under model-based semantic compression system. Second, the dependent columns in this kind of datasets will produce more same hash entries, thus the size of the hashes become smaller and lead to a significant speedup.

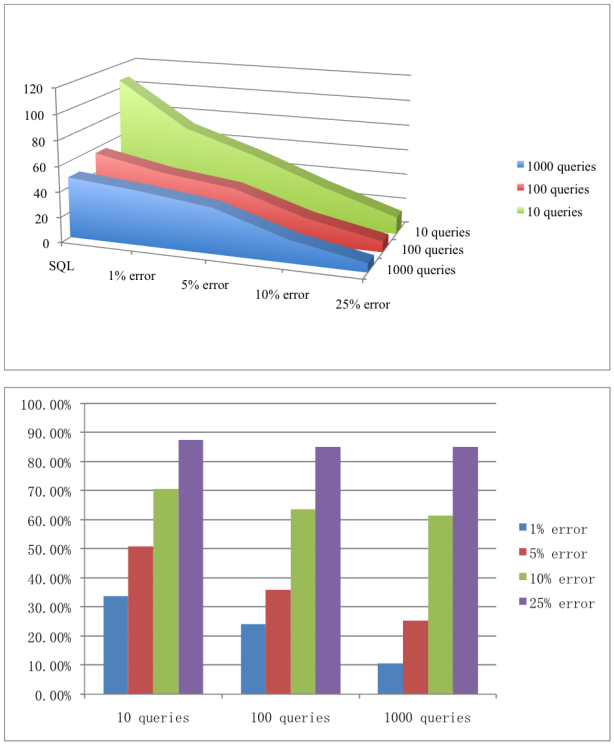

Figure 7: Column "H18pA", Dataset "house8L"

4 Conclusion

This project implemented a system supports Weka model based semantic compression on top of a mock column-oriented database. The system generate Weka models by using different machine learning algorithms to iteratively study the relations between columns. The system can predict certain attributes with the help of Weka models, and the predicted results are very accurate when the given dataset shows strong relationships between columns. The system also allows user to do the query with a certain error tolerance. The various experiments under different circumstance prove that queries with error tolerance can achieve significant improvements over the normal queries in speed when using model-based semantic compression in database system.

5 Future Work

In this project, the learning algorithms are used for generating Weka models are REPtree and M5P. REPtree is good for both numeric attributes and nominal attributes, whereas M5P can only be used for numeric value. Both of them can study the relations between columns very well and produce accurate predictions for numeric value. Although there are many other machine learning algorithms supported by Weka library are not as promising as those two during the test, it is important to research how to choose different algorithms for different types of

datasets.

On the other hand, the system runs on top of numeric datasets only for now. The performance of numeric value prediction is way better than the nominal one. It will gain more practical use if model-based semantic compression database supports nominal value prediction as well as the numeric one. The calculation for how much percent the nominal result has deviated from the original value is also different from the numeric one. Using edit distance to calculate the predicted value error percentage may be a good idea for the nominal attributes.

The system supports queries with or without WHERE clause. It is practical to use the system to search the data from the database. It would be more practical if the system allows more flexible queries even with aggregating operations.

Finally, for the future work, this system should be implemented over a real column-oriented database like C-Store, or a distributed system like Hive. The problem is that the column-oriented database systems are not widely used as the relational ones. Also, it will be a challenge to deal with weak-correlated datasets since it is difficult to study the relationships between columns, which may lead increased time costs and decreased predicting accuracy.

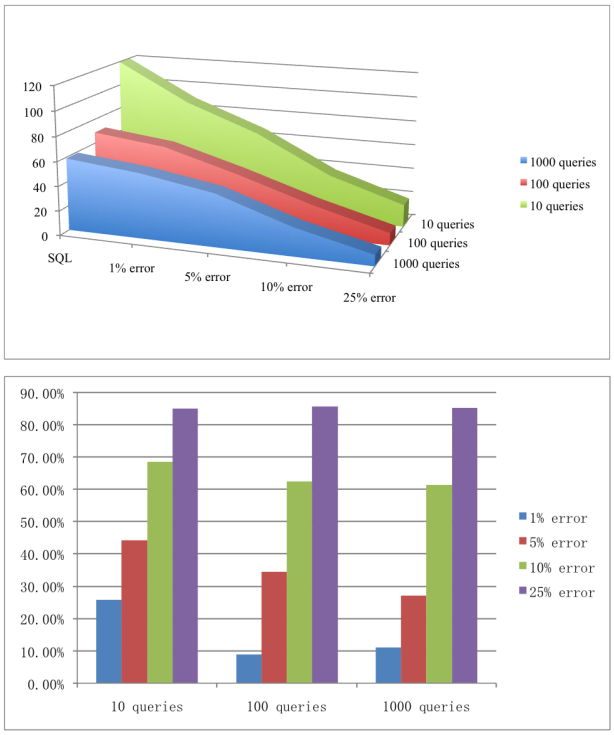

Figure 8: Column "P1", Dataset "house16H"

6 Acknowledgements

I would like to thank Professor Ugur Çetintemel for his help and for being my reading research advisor. He gives me a lot of inspiration on research of big data and distributed systems. He always patiently answered all my concerns and questions, and gives me valuable academic and career advice.

References

[1] Daniel Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM, 2006.

[2] Daniel J Abadi, Peter A Boncz, and Stavros Harizopoulos. Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665, 2009.

[3] Shivnath Babu, Minos Garofalakis, and Rajeev Rastogi. Spartan: A model-based semantic compression system for massive data tables. *ACM SIGMOD Record*, 30(2):283–294, 2001.

[4] HV Jagadish, Jason Madar, and Raymond T Ng. Semantic compression and pattern extraction with fascicles. In *VLDB*, volume 99, pages 7–10, 1999.

[5] HV Jagadish, Raymond T Ng, Beng Chin Ooi, and Anthony KH Tung. Itcompress: An iterative semantic compression algorithm. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 646–657. IEEE, 2004.

[6] Matteo Riondato, Mert Akdere, Ugur Çetintemel, Stanley B Zdonik, and Eli Upfal. The vc-dimension of sql queries and selectivity estimation through sampling. In *Machine Learning and Knowledge Discovery in Databases*, pages 661–676. Springer, 2011.

[7] Ian H Witten, Eibe Frank, Leonard E Trigg, Mark A Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.