Model-based DBs

1. USE CASES

As we have discussed, models can be used to support different use cases. Since each use case may have slightly different requirements, we next list all use cases that we want to support:

- *Explain data to end users.* In databases holding sensor data, users are often interested not in the actual data values (i.e., in the sensor readings) themselves but instead in higher-level statistical properties of the corresponding signals. For instance, in a database storing temperature readings inside an office building, one might be interested not in the actual temperature at a given time but instead in figuring out how the inside temperature changes with the time of the day or how the inside temperature correlates with the outside temperature. In other words, one might be interested in the *definition* of a model that captures the essence of the data values. Therefore, models can first of all be used as outputs of user queries.

- *Compress data.* However, models can be also useful as internal components of a database management system (DBMS) for facilitating various optimizations. An example of such an optimization is data compression. Having a model that correctly predicts the data most of the time, allows one to represent the data by keeping just the model and the data values that were not correctly predicted by the model (i.e., the exceptions), instead of all data values. For instance, assuming that the inside temperature can be predicted from the outside temperature through a function, one does not need to store both the inside and outside temperature values. Instead one can represent the same information by keeping just the outside temperature and the function definition, thus enabling data compression. Data compression is extremely important as it can used depending on the setting to achieve various efficiencies such as for instance reducing storage and network bandwidth requirements or increasing query processing performance.

- *Predict data values.* Last but not least, models can be used to predict values for points for which we do not have any data. For instance, a model can be used to predict the temperature at a place for which we do not have any readings. The output of this prediction can be used either by users (e.g., a person interested in this temperature) or by the DBMS itself to allow joining two mis-aligned data sets (e.g., joining two data sets containing inside and output temperatures at different places and at different points in time).

2. MODELS

This section is an attempt to formalize the notion of models. It is largely based on Yoav's write-up with the following changes: (a) generalization of the model definition to data sets that do not include a time dimension but other (potentially multiple) continuous dimensions, such as space, (b) introduction of the notions of encoding and decoding together with a discussion explaining where the data may be materialized, and (c) introduction of the notion of model class, with the intend to capture general classes of parameterized models (such as ARMA), whose instantiation leads to concrete models.

Raw Data. The raw data is represented by a relation R with schema S. Each row in R corresponds to a single measurement. The relation's schema S contains among other a set of special attributes, called the *identifying attributes*, that correspond to a set of continuous dimensions that uniquely identify each measurement. [1] For instance, the set of identifying attributes for a relation containing temporal measurements may be {time}, while the same set for a relation containing spatiotemporal measurements may be {time, latitude, longitude}.

Model Instance. A model instance (or in short model) $M = (Enc, Dec)$ is a pair, consisting of an encoder Enc and

[1]For our discussion, we assume that R is in a form where its identifying attributes form a continuous key. Note, that due to this restriction we may not be able to use directly relations already stored in the DBMS as input but may have to convert them first to relations satisfying the key requirement. For example, a relation containing the GPS paths for multiple users may have to be converted first to multiple relations; each containing the paths for a single user.

a decoder Dec, as described below.

Encoder. The encoder Enc is a program that, given R, generates a finite representation E of the data in R modulo some lost information that should ideally correspond to the noise of the signal captured by R. Intuitively it can be thought of as the program that compresses the data. We denote the result of applying Enc on R as $Enc(R)$ and call it the encoded data.

Decoder. The decoder Dec is a program that, given encoded data E and a binary sequence b, generates a relation over schema S that is continuous on the set of identifying attributes. We denote the result of applying Dec on E and b as $Dec(E, b)$.

Model accuracy. We say that model M is accurate for R if there is no program that can distinguish between the original relation R and $Dec(Enc(R), b)$ restricted to the combinations of identifying attribute values present in R, where b is a random binary sequence.

Model class. A model instance may be very efficient in representing a very particular type of signal (e.g., the temperature readings from a particular sensor). However, in our framework we strive for reusable models that can be employed in many different settings. This is captured by the concept of model class:

Let $\bar{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a list of variables and A_1, A_2, ..., A_n the domains of the respective variables. Then a model class with parameters $\bar{\lambda}$ is a function from $A_1 \times A_2 \times \ldots A_n$ to the set of all possible model instances.

For instance, ARMA is a model class, while a particular instantiation of the parameters of the ARMA model is an instance of that model. Our framework will include implementations of various model classes that can be tailored to specific data sets, producing corresponding model instances.

Data Materialization. Depending on the exact scenario, the data can be materialized (i.e., actually stored in the system) either as raw data or as encoded data. For instance, if the model is simply used to predict data values, one can simply store the raw data and apply the model on the fly. On the other hand, if the model is used in order to compress the data, one would most likely choose to store the encoded data instead.

3. QUERY LANGUAGE

For the following discussion let M be the output of a decoder, S its schema and $\{c_1, c_2, \ldots, c_n\}$ its set of identifying attributes. By definition, M is an infinite relation and as such its contents cannot be directly presented to the users. Instead we allow users to query M through *safe* queries (i.e., queries that are guaranteed to produce a finite output). We next list several classes of safe queries that can be evaluated over M:

- *Point queries*: These queries ask about a single tuple with particular values for all its identifying attributes. In other words they ask about a particular data point in the space $(c_1, c_2, \ldots, c_n)$. These queries can be written in SQL as:

  SELECT ...
  FROM M
  WHERE ($c_1 = v_1$ ... AND $c_n = v_n$)

- *Multi-point queries*: These form a generalization of point queries by allowing one to retrieve tuples for a set of points in the space $(c_1, c_2, \ldots, c_n)$. In SQL such queries can be written as:

  SELECT ...
  FROM M
  WHERE ($c_1 = v_1^1$ ... AND $c_n = v_n^1$) OR
  ($c_1 = v_1^2$ ... AND $c_n = v_n^2$) OR
  ...
  ($c_1 = v_1^m$ ... AND $c_n = v_n^m$)

- *Grid queries*: These queries retrieve the tuples of M for all points in the space $(c_1, c_2, \ldots, c_n)$ that fall on a grid, specified by an interval defined for each axis. They are similar to multi-point queries in that they retrieve tuples for a set of points. However, in the case of multi-point queries, these points are given in the query definition (and are thus data independent), while in the case of grid queries, they are generated from M's output according to the grid definition. Such a query can be written in a SQL-like notation through the introduction of a new "GRID" keyword as follows:

  SELECT ...
  FROM M
  WHERE ...
  GRID $int_1, int_2, \ldots, int_n$

  where int_i is the interval used to create the grid for attribute c_i, where $i \in [1, n]$

- *Aggregate queries:* These queries return values of aggregates functions that are well defined over a continuous data set. Examples of such functions are MIN, MAX and AVG.[2] The SQL representation of such queries is shown below:

  SELECT aggr($\bar{x}$)
  FROM M
  WHERE ...

- *Reconstruction queries:* If a model is used to compress data, one might want to reconstruct an approximation of the raw data relation. This is supported by the reconstruction query that returns tuples for all points in the space $(c_1, c_2, \ldots, c_n)$ that are also present in the raw data relation R. A proposed syntax for such queries is given below:

  RECONSTRUCT
  FROM M

[2]These functions are only well-defined if the model's output has a min and max value for each defining attribute (e.g., it provides only time points within a period $[t_1, t_2]$). We may want to add this restriction to the decoder's definition.