Multi-Modal Data Exploration with a Learned Relational Embedding

Raul Castro Fernandez, Samuel Madden

Introduction

Answering questions using unfamiliar relational databases is hard and time-consuming. Daily, analysts spend a lot of time understanding if a database contains the answer to the question they try to answer. They do that by engaging in *data exploration* sessions, in which they aim to understand the schema, contents of the database, and they devise the specific SQL query they must submit to obtain their answer [2]. During such process, they typically have only access to: i) a SQL interface; ii) the domain knowledge they possess; and iii) some documentation in the form of table descriptions and email exchanges with colleagues. Such resources are insufficcient to understand the database efficiently and require a big manual and time-consuming effort. As a result, exploring databases becomes a productivity killer.

Data exploration sessions are so common today that they constitute an entirely new database workload. However, while researchers and practitioners have learned how to support OLTP and OLAP workloads efficiently, i.e., by using row- and column-stores respectively, the same is not true for exploration workloads. The reason is that data exploration queries are often vague and ill-defined, but today's databases are designed to support precisely defined queries [1]. To illustrate the point, consider the following example of a data exploration session:

Suppose an analyst trying to justify new legislation for education in the US. The analyst gets access to a newly published public database that contains information about US federal spending and wants to use it to figure out how much money is spent in education per state. When facing the 500+ tables, it would be useful to have a classification of the relations based on the kind of content they possess so the analyst can quickly tell apart the relevant tables from others concerning health, infrastructure, etc. When narrowing down the search to a few relations of interest, it would be useful to have access to a meaningful summary of the relations, to confirm whether the tables are interesting or not. If the analyst is deciding whether a couple of tables are related to each other, it would be useful to understand a bit better how the rows in each table relate to each other–before making a huge effort in trying to combine them. At any point, if the analyst has a specific keyword to use to adjust the search, it would be good to understand which tables, columns and rows contain related information to the keyword.

Unfortunately, supporting the *exploration strategies* of the example above would require a lot of engineering effort, and in other cases it is hard to even formulate the problem to be solved. We would need to define a similarity metric between relations (or use an existing one such as [6]) and then implement it to solve the clustering problem. We would also need to implement a different technique to draw sample rows from tables to create useful summaries, using perhaps one of the techniques from [5]. To understand whether two tables are related to each other, we'd need to spend time and effort figuring out their join keys, before even knowing if they are useful at all. Finally, if a requirement changes, or the analyst wishes to perform a different exploration strategy, more effort must be spent upfront to support the new requirement. The fundamental problem is that the implementations of the relational model are not designed for exploration workloads, where we need the ability to query data for arbitrary relationships more freely. We propose to represent relational data in an alternative format that is more appropriate to support varied exploratory strategies such as the one in the example.

In particular, we want to represent relational data in a vector space in which cells, attributes, rows and tables are all vectors, and whose distance in the space determines how related they are to each other. Related may mean they appear in the same row, the same table, or that they share relationships across tables–whether they join or not. Suppose we have the federal spending database of the example above in a vector space with these characteristics. Clustering tables means clustering their vectors, summarizing relations means selecting a diverse set of rows that are far from each other, and figuring out if two tables are related to each other consists of finding pairs of rows that are close to each other in the space. This vector space representation is more appropriate because many exploration strategies can be expressed as vector operations that rely on distances between sets of points in the space.

In this short paper we: i) overview a research agenda around the challenge of building such vector space; ii) we summarize applications of such vector space beyond multi-modal database exploration. The main challenge of building the relational embedding is to come up with a vector space which represents relational data well and serves as a building block to implement various exploration strategies easily and efficiently. A principled way of building the vector space is to represent relational data in some multi-dimensional tensor—which would be very sparse—where dimensions correspond to the different attributes and values in the data, and then factorize the tensor to obtain a dense embedding that would contain information about how the variables are related to each other. This approach is so far only theoretical because we do not know how to precisely represent the relational data in a tensor form, or how to factorize it in such a way that the embedding represents the relationships of the original data. Instead, we propose to learn the embedding directly from the data, which we call *relational embedding*. For that, we build on top of modern language modelling techniques to build a high quality embedding. This process introduces new problems that we explain in detail next.

A Research Agenda for Relational Embeddings

Building a relational embedding consists of dictionary encode tokens from the relational table into vectors in such a way that the distance

between vectors representing related tokens is closer than between the vectors of unrelated tokens. Creating a relational embedding involves many challenges that we explore below:

Encoding strategy: A relation represents two types of relationships. The first type of relationship, called here *row-relationship*, is between the primary key and each of the other attributes of the relation. This relationship indicates that the tokens are related according to the specific attribute name. The second type of relationship, *column-relationship*, is between the values of a given attribute. This relationship indicates that the token are of the same type, which is indicated by the attribute. A relational embedding consists of two vector dictionaries: one to represent the tokens based on their row-relationship, and a second one to represent the tokens based on their column-relationship. Each token will have two vector representations.

The vectors are learned directly from the data using a skipgram model [3]. The Skipgram model, originally proposed for learning a language model, uses the concept of a *context*, which is a window over the text used to determine what pairs of words are positive samples (those within the window) and which pairs are negative samples (pairs formed by one word in the window and one word outside the window). To limit the number of total negative samples—necessary to achieve reasonable training times—a negative sampling strategy is often used.

When learning a relational embedding there are opportunities on deciding how to define the context of the skipgram model as well as the sampling strategy used. The goal is to obtain better vector representations than those given by a baseline skipgram model. One point of the research agenda is exploring different strategies for learning the relational embedding.

Space and Performance Overhead: To learn good quality vectors we must use a high capacity model: vectors must be high-dimensional, e.g., 100 dimensions. Each token from the relational data is mapped to two 100-dimensional vectors (one for row-relationship and one for column-relationship), and this introduces storage and performance overhead. For example, the relational embdding of a small, 100MB database, takes about 2GB in space. That's 20X more space! A research challenge is how to reduce the storage requirements of the embedding. Among the strategies we are using are: i) reducing the precision of the vector components once they've been learned; ii) reducing the dimensionality of the embedding using non-linear techniques [4] that maintain as well as possible the pairwise distances of the vectors in the original embedding. This second point is the most promising: a non-linear reduction to 3 dimensions reduces the relational embedding to around 40MB in the example mentioned above, that's a 50X reduction in storage, which permits querying the embedding with higher performance as well. However, it also introduces a data quality problem which we explore in the next paragraph.

Cleaning Vector Data: When dealing with high-dimensional data, one must pay attention to the problems associated to the curse of dimensionality. Two specific problems are a challenge that makes part of our research agenda: the existence of *hubs*, vectors that appear close to most other vectors, and therefore do not add useful information, and the phenomenon of *distance concentration*, which makes distances between high-dimensional vectors tend to the same magnitude. In addition to these two, because the relational embedding is learned, it is noisy. A research agenda item is how to define dirty data in the context of vector data—for example, hubs—and propose techniques to reduce them, of which those of topological and shape analysis seem promising. This is necessary because, otherwise, the effect of dirty data makes dimensionality reduction techniques very inefficient.

Performance Primitives/Structures: Many of the above requirements demand the computation of pairwise distances between a large number of vectors, e.g., to find hubs, to perform non-linear dimensionality reduction, etc. This operation can be very time consuming. Speeding up the process calls for the design and implementation of index structures geared towards vector data.

API guarantees: Both the multi-modal exploration application outlined in this paper as well as few others mentioned below demand querying vector data with variants of top-k queries. For the applications to be robust to user demands, we must produce rankings that are informative to users. Specific design points are the distance metric to use for measuring distance, the size of the ranking for which the results are likely of high precision. In addition, we must implement API calls efficiently for maximum performance.

Applications of Relational Embeddings

In addition to the multi-modal data exploration that motivates this short paper, we have identified and investigated other applications of relational embeddings which we outline below:

Fabric of data: Relational embeddings can be combined with text embeddings to come up with common intermediate representation to operate on structured and unstructured data at the same time. We call this abstraction the fabric of data; it has the promise of facilitating many data discovery queries than span different data formats, vastly improving users' productivity and enabling new applications.

Generative model of relational data: We can use the relational embedding as the training data of a generative model. Creating a generative model can be useful, for example, to grow the database with new new synthetic data that can then be used for performance benchmarking. The advantages of the model is that it allows to grow data that follows the distribution of each row- and column-relationship, instead of demanding a data distribution beforehand. In addition to the previous application, another promising application of the generative model is filling missing values, detecting outliers and other data quality issues.

Secure data exchange: It is plausible to share different translations of the embedding with different entities based on the security clearances of those individuals. This can be done in such a way that only the person with the right translation can query the embedding and obtain results that are sensical. Exploring how to leverage differential privacy in this context is another way in which the relational embedding may facilitate secure data exchange.

Conclusion

Today's analysts engage in complex data exploration sessions to understand databases and answer the questions necessary to perform their job. Relational embedding is an abstraction that represents data as a vector space, and that has the potential of facilitating a multi-modal exploration strategy of relational data. We have identified an interesting research agenda around the construction and usage of relational embeddings, as well as other applications that are enabled by the new abstraction.

1. REFERENCES

[1] J. M. Hellerstein, M. Stonebraker, and J. Hamilton. Architecture of a database system. *Foundations and Trends® in Databases*, 2007.

[2] S. Idreos, O. Papaemmanouil, et al. Overview of Data Exploration Techniques. In *SIGMOD*, 2015.

[3] T. Mikolov, I. Sutskever, et al. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 2013.

[4] L. van der Maaten and G. E. Hinton. Visualizing Data using t-SNE. 2008.

[5] Y. Wang, A. Meliou, et al. RC-index: Diversifying Answers to Range Queries. *VLDB*, 2018.

[6] X. Yang, C. M. Procopiuc, et al. Summarizing Relational Databases. *VLDB*, 2009.