ENFrame = $\frac{\textbf{Programs + Queries}}{\textbf{Probabilistic Data}}$

Highlights on published work (with a bit of vision)

Dan Olteanu
University of Oxford & LogicBlox Inc.

Sebastiaan J. van Schaik
University of Oxford

ABSTRACT

This paper overviews ENFrame, a framework for processing probabilistic data. In addition to relational query processing supported by existing probabilistic database management systems, ENFrame allows programming with loops, assignments, list comprehension, and aggregates to encode complex tasks such as clustering and classification of data retrieved via queries from probabilistic databases.

We explain the design choices behind ENFrame, some distilled from the wealth of work on probabilistic databases and some new. We also highlight a few challenges lying ahead.

1. MOTIVATION AND SCOPE

Probabilistic data management has gone a long, fruitful way in the last decade [8]: We currently have a good understanding of the space of possible relational and hierarchical data models and its implication on query tractability; the community already delivered several open-source systems that exploit the first-order structure of database queries for scalable inference, e.g., MystiQ, Trio, and MayBMS to name very few, and applications in the space of web data management, e.g., SPROUT^2 [3] with more to come in near future from Google Knowledge World. Significantly less effort has been spent on supporting complex data processing beyond mere querying, such as general-purpose programming or even data mining tasks.

There is a growing need for computing platforms that would allow users to build applications feeding on uncertain data without worrying about the underlying uncertain nature of such data or the computationally-hard inference task that comes along with it. For tasks that only need to query probabilistic data, existing systems do offer a viable solution. For more complex tasks, however, successful development requires a high level of expertise in probabilistic databases and this hinders the adoption of the existing technology as well as communication between potential users and experts. A similar observation has been recently made in the areas of machine learning [1] and programming languages [4], with the goal of developing a probabilistic programming framework that allows models to be expressed concisely, yet with random variables and probabilities as first-class citizens. This follows a line of work on, e.g., marrying first-order logic with Bayes nets in BLOG [6] and probabilistic extensions of logic programming [5].

The thesis of this paper is that one can build powerful and useful systems for processing probabilistic data that *leverage* aforementioned existing work by the data management community. ENFrame, our recent probabilistic data processing framework [9], fits this vision.

Following the design of the MayBMS/SPROUT probabilistic database system, ENFrame adheres to a sound semantics (the possible-worlds semantics) for its whole processing pipeline; it exploits where possible the structure of queries for scalable processing; and it uses a rich language of probabilistic events to express input correlations, to trace the correlations introduced during task computation, and to enable sensitivity analysis, incremental maintenance, and knowledge compilation-based approaches for approximate inference. A key property inherited from database systems principles is that it separates between the physical and logical representations of probabilistic data [7]: Whereas the physical representation may be that of, e.g., Bayes nets for some sources and (variants of) probabilistic c-tables for others, the users are exposed to a unified relational view of the underlying data. Finally, the users are oblivious to the probabilistic nature of the data: They program as if the input data is plain relational, with no uncertainty or layout intricacy; it is the ENFrame's job to make sense of the underlying data formats, probabilities, and input correlations, thus relieving the users from requiring expert knowledge on inference, query tractability, and probabilistic models and interpretation of their programs.

ENFrame then adds a programming language with rich constructs that allow to express complex tasks such as clustering or nearest neighbour classification intermixed with structured querying: User programs are written in a fragment of Python that so far supports bounded-range loops, if-then-else statements, list comprehension, aggregates, and query calls to external database engines. This addition has non-trivial effects on the other aspects of the system. Firstly, the event language needed to support such tasks is powerful. While propositional formulas over Boolean random variables are expressive enough to capture program computation traces, they can be extremely large and expensive to manage. ENFrame's event language extends algebraic formalisms based on semirings and semi-modules for probabilistic data [2] that can succinctly encode program events (they can be exponentially more succinct than equivalent propositional formulas or Bayesian nets). Secondly, ENFrame needs to scale up inference of networks of highly interconnected events. It does so using distributed approximate inference algorithms that work for a bulk of events at a time.

```
1  (O, n) = loadData()          # list and number of objects
2  (k, iter) = loadParams()     # number of clusters and iterations
3  M = init()                   # initialise medoids
4
5  for it in range(0,iter):     # clustering iterations
6   InCl = [None] * k           # assignment phase
7   for i in range(0,k):
8    InCl[i] = [None] * n
9    for l in range(0,n):
10    InCl[i][l] = reduce_and(
11      [(dist(O[l],M[i]) <= dist(O[l],M[j])) for j in range(0,k)])
12  InCl = breakTies2(InCl)     # each object in exactly one cluster
13
14  DistSum = [None] * k        # update phase
15  for i in range(0,k):
16   DistSum[i] = [None] * n
17   for l in range(0,n):
18    DistSum[i][l] = reduce_sum(
19      [dist(O[l],O[p]) for p in range(0,n) if InCl[i][p]])
20
21  Centre = [None] * k
22  for i in range(0,k):
23   Centre[i] = [None] * n
24   for l in range(0,n):
25    Centre[i][l] = reduce_and(
26      [DistSum[i][l] <= DistSum[i][p] for p in range(0,n)])
27  Centre = breakTies1(Centre)  # enforce one medoid per cluster
28
29  M = [None] * k
30  for i in range(0,k):
31   M[i] = reduce_sum([O[l] for l in range(0,n) if Centre[i][l]])
```

```
$\forall i$ in $0..n-1 : O^i \equiv \Phi(o_i) \otimes \vec{o}_i$
$M^0_{-1} \equiv \Phi(o_{\pi(0)}) \otimes \vec{o}_{\pi(0)}; \ldots ; M^{k-1}_{-1} \equiv \Phi(o_{\pi(k-1)}) \otimes \vec{o}_{\pi(k-1)}$

$\forall \mathrm{it}$ in $0..\mathrm{iter}-1$ :
  $\forall i$ in $0..k-1$ :
    $\forall l$ in $0..n-1$ :
      $\mathrm{InCl}^{i,l}_{\mathrm{it}} \equiv \bigwedge_{j=0}^{k-1} \left[\mathrm{dist}(O^l, M^i_{\mathrm{it}-1}) \leq \mathrm{dist}(O^l, M^j_{\mathrm{it}-1})\right]$

  // Encoding of breakTies2 omitted

  $\forall i$ in $0..k-1$ :
    $\forall l$ in $0..n-1$ :
      $\mathrm{DistSum}^{i,l}_{\mathrm{it}} \equiv \sum_{p=0}^{n-1} \mathrm{InCl}^{i,p}_{\mathrm{it}} \wedge \top \otimes \mathrm{dist}(O^l, O^p)$

  $\forall i$ in $0..k-1$ :
    $\forall l$ in $0..n-1$ :
      $\mathrm{Centre}^{i,l}_{\mathrm{it}} \equiv \bigwedge_{p=0}^{n-1} \left[\mathrm{DistSum}^{i,l}_{\mathrm{it}} \leq \mathrm{DistSum}^{i,p}_{\mathrm{it}}\right]$

  // Encoding of breakTies1 omitted

  $\forall i$ in $0..k-1$ :
    $M^i_{\mathrm{it}} = \sum_{l=0}^{n-1} \mathrm{Centre}^{i,l}_{\mathrm{it}} \wedge O^l$
```

Figure 1: k-medoids clustering specified as user program and event program.

2. ENFRAME BY EXAMPLE

Figure 1 shows ENFrame's user program (left) and the translation into an event program (right) for k-medoids clustering. We highlight some aspects here and refer to [9] for a detailed description.

The k-medoids clustering algorithm selects a set of k initial medoids (cluster centres), then iteratively assigns data points to the closest medoid and updates the medoids.

On line 3, the data is loaded – either from a database (if a query is provided), or from an external data source. After selecting the initial medoids (line 3), the assignment (line 6 onward) and update (line 14 onward) phases are specified. An object is assigned to a cluster C_i if distance to C_i's medoid is the smallest (line 11). Similarly, an object is selected as a cluster medoid of C_i if the sum of distances to all other objects in the cluster (line 18) is minimal (line 26).

The event program uses *conditional values* to construct random variables O_i whose outcomes are vectors in the feature space (line 1): O_i is defined as $\vec{o}_i$ when $\Phi(o_i)$ is true or *undefined* otherwise. $\mathrm{InCl}^{i,l}_{\mathrm{it}}$ is a Boolean random variable that mirrors the Boolean variable `InCl[i][l]` in the user program and represents the event that object o_l is assigned to cluster C_i in iteration *it*. $\mathrm{DistSum}^{i,l}_{\mathrm{it}}$ is a real-valued random variable representing all possible sums of distances from candidate medoid o_l to other objects in cluster C_i.

All program variables have a probabilistic interpretation and thus define a probability density function that can be inspected by the user. For instance, the user can define a variable stating that two objects belong to the same cluster, or are likely to co-occur together in clusters. Probability density functions of program variables can be further processed, e.g., by a probabilistic DBMS or subsequent programs.

3. OPEN ENDS

Many aspects of ENFrame as described in this paper are work in progress: The trade-off between the expressiveness of the user language and efficient processing; which additional constructs are needed in the event language to support a library of common data mining tasks; and how to exploit the program structure (in addition to query structure) for efficient inference. A better integration of queries and program constructs is desirable, with relations and program data structures (e.g., multi-dimensional arrays) to be used interchangeably in both programs and queries.

We plan to investigate a functionality-performance trade-off observed when lifting the fine-grained events obtained by exhaustive grounding of the user program to their higher order realisation. For the k-medoids program, all cluster membership events for a given cluster C_i can be lifted to one higher-order event, which expresses the membership of all data points to C_i and can be compiled into C++ code for efficiency reasons. Whereas the fine-grained events lead to large networks and poorer performance of inference, they support more accurately answer explanation, sensitivity analysis, and incremental maintenance.

4. REFERENCES

[1] Defense Advanced Research Projects Agency. Probabilistic programming for advancing machine learning, April 2013. DARPA-BAA-13-31.

[2] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.

[3] R. Fink, A. Hogue, D. Olteanu, and S. Rath. SPROUT^2: A squared query engine for uncertain web data. In *SIGMOD*, 2011.

[4] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *ICSE*, 2014. Future of Software Engineering track (to appear).

[5] L. De Raedt and A. Kimmig. Probabilistic programming concepts, December 2013. arXiv 1312.4328.

[6] B. Milch and et al. BLOG: Probabilistic models with unknown objects. In *IJCAI*, 2005.

[7] D. Olteanu, L. Papageorgiou, and S. van Schaik. Πgora: An integration system for probabilistic data. In *ICDE*, 2013.

[8] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases.* Morgan & Claypool, 2011.

[9] S. van Schaik, D. Olteanu, and R. Fink. ENFrame: A platform for processing probabilistic data. In *EDBT*, 2014.