More about Cleaning RFID Data

Jun Yang
CPS 296.1, Spring 2007
Sensor Data Processing
With contents from N. Khoussainova

Announcements (Apr. 3)

- Project milestone 2
  - I want to know your progress by April 20!
  - Let me know if you need any more motes soon
- Given the schedule, we will not have a second round of student presentations
  - Grading scale will be adjusted accordingly
- Reading for this Thursday: query evaluation on probabilistic databases
  - A review is due if you did not submit one for today

More on cleaning RFID data

Need for data cleaning

- Applications frequently rely on devices such as
  - RFID readers
  - Light/motion/temperature sensors
- But these devices are unreliable
  - Readings can be missing, duplicated, or simply erroneous
- Example: RFID-enabled library

Key observations in Khoussainova et al.

- Integrity constraints naturally arise in data
  - User/application rely on these constraints, and can specify them explicitly
- Constraint violations $\rightarrow$ input data errors
  - Correct errors so data conforms to constraints
- Also: no single way to correct errors $\rightarrow$ present cleansed data with uncertainty

Constraint taxonomy

<table>
<thead>
<tr>
<th></th>
<th>Stateless</th>
<th>Stateful</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusion</td>
<td>A book in the catalog must appear in at least one location</td>
<td>A returned book must have been previously checked-out</td>
</tr>
<tr>
<td>Exclusion</td>
<td>A book can appear in at most one location</td>
<td>A checked-out book cannot appear in the library</td>
</tr>
</tbody>
</table>
System architecture

Constraint language

\[
\text{FORALL} \quad \text{INPUT}_1 \text{ as } I_1, \ldots, \text{INPUT}_n \text{ as } I_n, \\
\text{WHERE} \quad \text{EXPR}_1 \\
\text{CHECK} \quad \text{EXPR}_2 \\
\text{CONFIDENCE} \quad c
\]

- A checked-out book can not appear in the library

\[
\text{FORALL} \quad \text{CHECKOUTS} \ C \\
\text{WHERE} \quad C.t < \text{now}() \\
\text{CHECK} \quad \text{COUNT} \text{(SELECT} \quad * \\
\text{FROM} \quad \text{SIGHTINGS} \ S \\
\text{WHERE} \quad S.bid = C.bid \\
\text{AND} \quad S.t = \text{now}() \text{)} <= 0
\]

Probabilistic output

- A tuple \((i, j)\) means that book \(i\) is on shelf \(j\)
- Assign this tuple a probability \(p_{i,j} \in [0, 1]\)
  representing the probability that \(i\) is indeed on \(j\)
- Tuple probabilities offer a natural way to "soften" beliefs to make integrity constraints hold
- They are easy to interpret
  - But is this representation “complete”?
Using integrity constraints

- Integrity constraints → constraints involving probabilities
- Handle inclusions constraints first
  - If violated, generate fix-up tuples and associated constraints on probabilities
- Handle exclusion constraints next
  - Identify conflicting groups of tuples and generate associated constraints on probabilities
- Solve for probabilities to maximize entropy
  - Intuitively, to make the answer least biased
- Any implicit assumption here?

More details

- Fix-up tuple: tuple(s) that would eliminate an instance of an inclusion constraint violation had it (they) existed
  - E.g.: if there is no reading for book b1 at all, then one of the possible fix-up tuple would be (b1, s1)
- Conflicting group: a group of tuples that together violate an exclusion constraint
  - E.g.: if there are two readings for b1 at the same time, say (b1, s1) and (b1, s2), then these two tuples form a conflicting group

Example of fix-up tuples

- Inclusion constraint: each book in the catalog must appear on at least one location
- No sighting for book b!
- Generate fix-up tuples: (b, s1), (b, s2), (b, s3), …
- Generate constraint: \( p_{b,s_1} + p_{b,s_2} + p_{b,s_3} + \cdots \geq 1 \)

- Is this constraint as restrictive as the original one?
A simple example of conflicting group

- Exclusion constraint: a book known to be checked out cannot be on any shelf
- Book b detected on shelves s1 and s2!
- Identify conflicting group: (b, s1), (b, s2)
- Generate constraint: \( p_{b, s1} + p_{b, s2} \leq 0 \)
  - Effectively discards (b, s1), (b, s2)

A more complex example

- Exclusion constraint: shelf s1’s maximum capacity is 10 books
- More than 10 books (b1, b2, …, b15) seen on s1!
- Identify conflicting group: (b1, s1), (b2, s1), …, (b15, s1)
- Generate constraint:
  \[
  p_{b1, s1} + p_{b2, s1} + \cdots + p_{b15, s1} \leq 10
  \]
  - Note: some of the probabilities may be involved in other constraints too, which can “help” satisfying this one
  - Is this constraint as restrictive as the original one?

“Soft” constraints

- Inclusion constraint: if book b is on shelf s at the previous timestep, it is likely to remain the same shelf in the current timestep (with 0.3 “confidence”)
- No sighting for book b on s!
- Generate fix-up tuple: (b, s)
- Generate constraint: \( p_{b, s} \geq 1 \times 0.3 \)
  - Confidence lowers lower bound (and raises upper bound)
Assigning probabilities

- Probability assignments should
  - Satisfy the constraints generated
  - Be as uniform (least biased) as possible ← principle of maximum entropy

- Maximize: $-\sum (p_k \log p_k)$ subject to all constraints

Maximum entropy (MaxEnt) primer

- A method for picking a probability distribution subject to a known set of constraints
  - E.g., the expected number of books on shelf $s$ is 8; the probability of book $b$ on shelf $s$ is at least 0.6; etc.
  - And nothing else is known!

- Examples of MaxEnt distributions
  - Given no info: uniform
  - Given mean and standard deviation: Gaussian
  - Given mean $1/\lambda$ and support $[0, \infty)$: exponential rate $\lambda$

- Efficient off-the-shelf MaxEnt solvers exist

Qualitative evaluation: constraints

Maximize $-\sum (p_k \log p_k)$ subject to:

- (Max capacity)
- ($b34$ is shelved)
- ($b4,b5,b6$ are checked out)
- ($b34$ can only be at one location)

Constraint 1
Constraint 2
Constraint 3
Constraint 4
Qualitative evaluation: results

- Query: What fraction of books are misshelved?
- In reality, 5 books out of 20 books are misshelved (0.25)
- RFID antennas reported that 10 books out of 33 are misshelved (0.303)
- MaxEnt-based approach reports that 5.4994 books out of 20.802 are misshelved (0.264)

Solution performance

- Take-away point:
  really fast—feasible in a stream setting

Discussion

- Use constraints to help data cleaning!
  - Declarative specification of constraints
  - Can leverage historical data and external information
- Result captures uncertainty, but
  - Some dependencies are lost → need a more powerful representation?
- MaxEnt is computationally efficient, but
  - After translation, some constraints are not as tight as they can be
  - Statistical prior knowledge is difficult to incorporate → need a more Bayesian framework?
    - Soft constraints cannot fully accomplish it
An alternative approach

- Framework: Bayesian
  - Allows principled incorporation of statistical prior knowledge, known constraints, and observations
- Inference procedure: Monte Carlo sampling
  - Supports complex distributions and constraints
- Representation: a weighted collection of samples
  - Captures the distribution of all possible “true states”
  - No independence assumption

Modeling the application

- Indicator variables
  - \( x_{ij} \): 1 if book \( i \) is really on the shelf \( j \); 0 otherwise
    - True state; unobserved
  - \( y_{ij} \): 1 if book \( i \) is detected by shelf \( j \)'s sensor; 0 otherwise
    - Observed but noisy

Linking observation and reality

- Bayes’ theorem: posterior = (likelihood \( \times \) prior) / (normalizing constant)
  \[ p(\hat{x}|\hat{y}) = \frac{p(\hat{x}, \hat{y})}{p(\hat{y})} \propto p(\hat{y}|\hat{x})p(\hat{x}) \]
- Our goal: derive posterior from likelihood and prior
- Likelihood example: RFID detection error model
- Prior examples:
  - Each book has a designated shelf on which it appears with highest probability
  - Each book is likely to remain on the same shelf
Incorporating constraints

- Draw samples from posterior distribution subject to hard constraints on $x_{ij}$’s
  - Each valid sample must observe all these constraints
  - Think of it as a possible database state
  - Each sample has a weight reflecting how likely it is the true state
- Example constraints:
  - Each book must appear at least one location: $\sum_j x_{ij} \geq 1$
  - Maximum capacity of shelf $j$ is 10: $\sum_i x_{ij} \leq 10$

*Normal “database” constraints; no need to translate to those involving probabilities*

Drawing samples

- Challenges
  - Posterior may be complex and difficult to sample directly
  - Constraints may invalidate most of the samples drawn
    - Tight constraints can easily waste 99% of the samples
- Sequential importance sampling (SIS)
  - Draw one $x_{ij}$ at a time, from its possible range(s) derived from constraints
  - Each $x_{ij}$ drawn updates constraints on remaining values to be drawn
  - Correct the sampling bias in the weight

Sampling scalability

- Deriving the possible range(s) for a variable given constraints can be computationally very expensive
- If the ranges are not as tight as possible, lots of invalid samples can still be generated
- Fortunately, for this RFID cleaning application, constraints are easy enough that there exist a fast algorithm for deriving possible ranges accurately

*Figure showing a graph or diagram related to the sampling process.*
Queries over weighted samples

- General strategy
  - Evaluate query over each weighted sample
  - The collection of weighted answers represent the answer distribution
- How many books are currently located on shelf $s$?
  - Compute count for $s$ in each sample
  - Expected count = weighted average of all counts
- What is the probability that shelf $s$ has more than 8 books?
  - For each sample, return 1 if statement is true; 0 otherwise
  - Probability = weighted average of all returned values
- What is the probability that books $b_1$ and $b_2$ are on the same shelf?
  - Same as above

Discussion

- Principled way of incorporating statistical prior knowledge, known constraints, and observations, all in one framework
- Beyond RFID?
  - RFID data arguably is “easier” to cleanse because it is binary
  - Discrete representations are insufficient for continuous domains
- “Collection of samples” representation can be generalized
- Functionality and generality comes at a cost
  - “Collection of samples” representation in general is less efficient than specialized representations
  - Sampling and evaluating queries over samples may be expensive
- But for some apps, this might be the only approach feasible
  - Future work on improving its efficiency is well justified