

# Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics

Davide Barbieri, Daniele Braga, Stefano Ceri, and Emanuele Della Valle,  
*Polytechnic of Milan*

Yi Huang and Volker Tresp, *Siemens*

Achim Rettinger, *Karlsruhe Institute of Technology*

Hendrik Wermser, *Technical University of Munich*

*A combined approach of deductive and inductive reasoning can leverage the clear separation between the evolving (streaming) and static parts of online knowledge at conceptual and technological levels.*

**W**hat are the hottest topics discussed on Twitter? Which topics have my close friends discussed in the last hour? Which movie is my friend most likely to watch next? Which Tuscan red wine should I recommend? With many popular social networks publishing microblogs and feeds, the information

required to answer these queries is becoming available on the Web. This trend is often referred to as the *Twitter phenomenon*. These feeds exist in a stream, a continuous flow of information where recent items are typically more relevant than older ones. However, their interpretation requires rich background knowledge to fulfill meaningful reasoning tasks, beyond standard stream-processing capabilities.

Both the database<sup>1</sup> and data mining communities have studied stream processing. Specialized data stream management systems (DSMSs) are on the market, and DSMS

features are appearing in major database products, such as Oracle and DB2. Online stream mining applies in many contexts, such as in computer network traffic for intrusion detection, in Web searches for online recommendations,<sup>2</sup> and in sensor data for automated real-time decision making. These applications represent a paradigm change in information processing techniques, because data streams are processed on the fly, without being stored, and processing units produce their results without explicit invocation.

DSMSs are designed to process real-time parallel queries over possibly bursty data,

but they cannot perform reasoning tasks as complex as those required to answer our sample queries. Understanding and interpreting the Twitter phenomenon (and many other data streams) requires connecting quick and concise real-world streams to rich background knowledge bases. This combination is crucial for several reasons. It enables

- understanding the topics discussed in the streams with the help of topic taxonomies,
- recommending the most attractive movies to particular user profiles, and
- predicting future behaviors based on the analysis of past behaviors (such as movie attendance).

These few examples show the need for connecting data-stream-processing techniques with both inductive and deductive reasoning methods to support social media analytics. We believe that this is a good example of how the stream reasoning<sup>3</sup> research area enables the merging of data streams and rich background knowledge.

Extending reasoning methods to support changing knowledge is a known challenge for the reasoning community. In deductive reasoning, various methods exist to revise beliefs based on recent information. In inductive reasoning, a body of research in data mining and machine learning already supports online data analysis. However, little work has been done on applying machine learning to streams as rich and structured as those we consider here. We believe that data streams are an ideal model for changes occurring in the real world, as well as a suitable means to delimit the source and the nature of change, clearly separating the static and dynamic parts of knowledge.

This separation, both conceptual and technological, lets us use existing systems for data-stream management and for inductive or deductive reasoning. In our approach, RDF streams, together with an extension of the SPARQL language for continuous queries, are the “glue” that interconnect deductive and inductive reasoning.

### Stream Reasoning

The combination of deductive and inductive stream reasoning extends the notion of pure *stream reasoning*, which involves reasoning in real time on huge and possibly noisy data streams to support numerous concurrent decision processes.

We can characterize stream reasoning with respect to three key concepts in stream processing:<sup>4</sup>

- *Streams*. Data streams are unbounded sequences of time-varying data elements that form a continuous flow of information. Recent updates are more relevant because they describe the current state of a dynamic system. Because RDF is the data interchange format for reasoners, we start with the notion that RDF streams are the fuel for stream reasoning. We define RDF streams as ordered sequences of pairs, made of RDF triples and their timestamps  $T_i$ :

$$\langle \langle \text{subj}_i, \text{pred}_i, \text{obj}_i \rangle, T_i \rangle$$

$$\langle \langle \text{subj}_{i+1}, \text{pred}_{i+1}, \text{obj}_{i+1} \rangle, T_{i+1} \rangle$$

Timestamps are annotations of RDF triples. They are monotonically nondecreasing in the stream ( $T_i \leq T_{i+1}$ ), and adjacent triples can have the same timestamp if they occur at the same time.

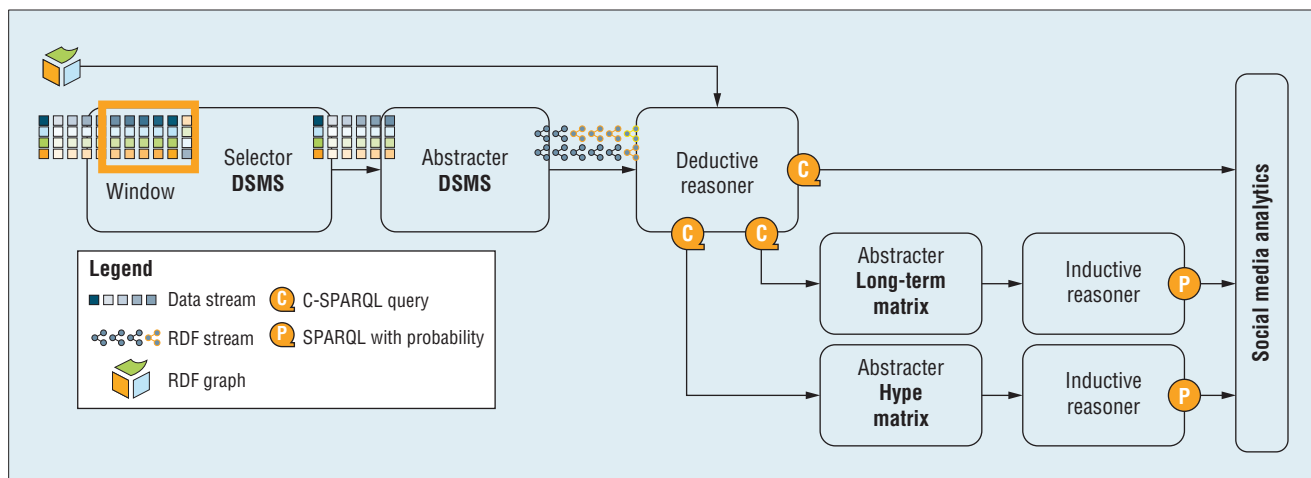
- *Windows*. Traditional reasoning problems assume that we should consider all the available information

when trying to solve a problem. Stream reasoning, instead, restricts processing to a certain window of concern, focusing on a subset of recent statements in the stream, while ignoring previous statements. However, the cumulative effect of past windows (processed in the past) and present windows can be taken into account.

- *Continuous processing*. Traditional reasoning approaches have well-defined beginnings and endings for reasoning tasks indicating when tasks are presented to the reasoner and when results are delivered, respectively. Stream reasoning moves from this processing model to a continuous model, where tasks are registered and continuously evaluated against flowing data.

We are pursuing our stream reasoning vision within the LarKC project ([www.larkc.eu](http://www.larkc.eu)),<sup>5</sup> with the goal of developing a platform for reasoning on massive heterogeneous information such as social media data. The platform has a pluggable architecture to exploit techniques and heuristics from diverse areas such as databases, machine learning, and the Semantic Web.

In previous research,<sup>6,7</sup> we specified a general, flexible architecture for reasoning over data streams and rich background knowledge, within the LarKC conceptual architecture,<sup>5</sup> to leverage existing DSMS and SPARQL engines. We introduced continuous SPARQL (C-SPARQL) as a SPARQL extension for expressing continuous queries over RDF graphs and RDF streams.<sup>6,8</sup> We elaborated on the deductive reasoning support to C-SPARQL, proposing an efficient incremental technique that exploits the transient nature of streams for maintaining the materialization of their ontological entailments.<sup>7</sup>



**Figure 1.** Architecture of a simple stream reasoned as a set of specialized plug-ins within the LarKC platform. We applied the reasoner to social media analysis.

The challenges of inductive stream reasoning are

- the large amount of information that must be processed in a given time window,
- the structured multirelational nature of the data,
- the sparsity of the typically high-dimensional data, and
- the fact that the data is often incomplete.

Researchers have described a machine learning approach suitable for this challenging data situation called the Statistical Unit Node Set (SUNS) learning approach.<sup>9</sup> Later work then extended the approach for online inductive reasoning.<sup>10</sup>

Figure 1 shows the architecture of a simple stream reasoner that consists of a set of specialized plug-ins within the LarKC platform. A selection plug-in extracts the relevant data in each input stream by exploiting the DSMS's window-processing ability. The window content is fed into a second plug-in that abstracts from fine-grained data streams into aggregated events and produces RDF streams as output. A SPARQL engine that can operate under different entailment regimes constitutes the deductive reasoner plug-in. C-SPARQL queries

are directly registered in the deductive reasoner. The results can be used immediately or by two more sub-workflows, both consisting of an abstracter and an inductive reasoner. As we explained earlier, the inferences of the two inductive reasoners can be queried using an extended version of SPARQL that supports probabilities.<sup>9</sup>

We can extend this simple setup by arbitrarily combining and iterating the deductive and inductive reasoners. For example, it might be helpful to feed the findings of the inductive reasoner back to the deductive reasoner to deduce further knowledge.

### Experimental Data

We based our experiments on Glue (<http://getglue.com>), a social network that lets users connect to each other and share Web navigation experiences. In addition, Glue uses semantic recognition techniques to identify books, movies, and other similar topics and publishes them in the form of data streams. Users can observe the streams and receive recommendations on interesting findings from their friends.

Both the social network data and the real-time streams are accessible via Web APIs. Our experiments built on adapters<sup>11</sup> that export Glue data as RDF streams. Figure 2 gives UML

descriptions of the entities and relationships in the experiments.

Users have online names, and they know and follow other users using well-known Semantic Web vocabularies,<sup>12</sup> such as the Friend of a Friend (FOAF) vocabulary for user names and the *knows* relationship, and the Semantically Interlinked Online Communities (SIOC) for the *follows* relationship. *Objects* represent real-world entities (such as movies or books) with a name and category. *Resources* represent information sources that describe the actual objects, such as webpages about a particular movie or book. For vocabularies, we used `rdfs:label` for the names and `skos:subject` to link an object to its category, by means of the subject attribute. Moreover, we used categories identifiers from the YAGO knowledge base.

The information we have described so far is static background knowledge—that is, in the experiments we assumed that the background knowledge is stable in a period comparable with the size of a window. Of course, we allow updates to this information that do not interfere with window processing. We also have streaming information, namely the notifications of the users' behaviors with respect to resources (and, transitively, to objects).

The accesses, likes, and dislikes relationships represent the events occurring when users access resources or express opinions about them. We refer to this vocabulary with the prefix *sd*. Quite straightforwardly, each interaction of a generic user *U* with a resource *R* generates a triple of the form  $\langle U, sd:accesses, R \rangle$ , and selected interactions generate triples of the form  $\langle U, sd:likes, R \rangle$  and  $\langle U, sd:dislikes, R \rangle$ . Figure 3a shows examples of possible triples.

### Stream Reasoning at Work

To demonstrate stream reasoning, we start with an example of social media analysis performed by a C-SPARQL query under simple RDF entailment. Then, we explain how we express complex conditions using the rule profile of OWL2 (OWL2-RL) and we explain how our deductive stream reasoner can efficiently answer C-SPARQL queries under OWL2-RL entailment.

### C-SPARQL under Simple RDF Entailment

Like SPARQL, C-SPARQL can be executed under multiple entailment regimes (see [www.w3.org/TR/sparql11-entailment](http://www.w3.org/TR/sparql11-entailment)). Under simple RDF entailment, C-SPARQL does not require reasoning, but it is already useful. For instance, we can use it to discover causal relationships between different users' actions in Glue (see Figure 3b).

In Figure 3b, lines 1 and 3 tell the C-SPARQL engine to register the stream of interactions Glue generates. They also tell the C-SPARQL engine to observe the stream through a 30-minute window that slides every five minutes. Line 2 tells the engine to generate an RDF stream as output. The basic triple pattern (BTP) at line 5 matches interactions of potential opinion makers with resources. Line 6 matches the opinion makers' followers,

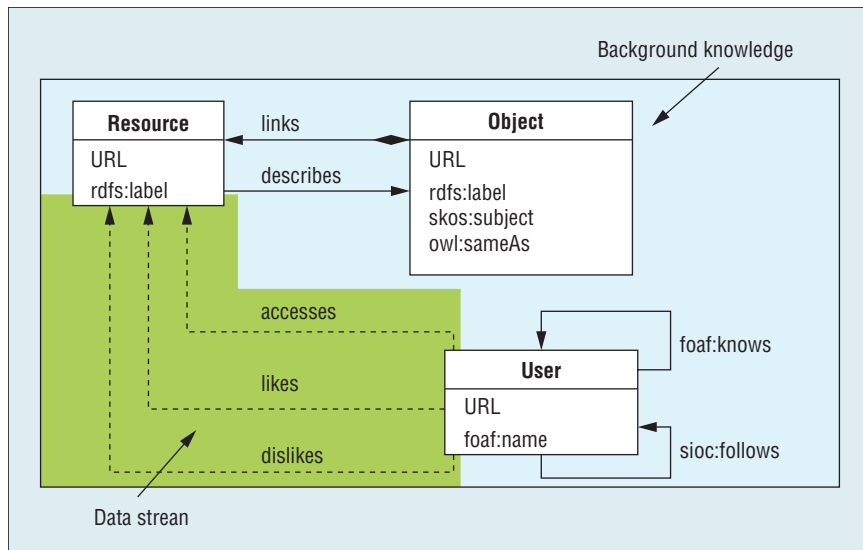


Figure 2. Entities and relationships in our experiments. The UML descriptions show that objects represent real-world entities and resources represent information sources.

```

(<:Giulia, sd:accesses, : Avatar>, 2010-02-12T13:18:05)
(<:John, sd:accesses, : Twilight>, 2010-02-12T13:36:23)
(<:Giulia, sd:likes, : Avatar>, 2010-02-12T13:42:07)

```

(a)

```

1. REGISTER STREAM OpinionMakers COMPUTED EVERY 5m AS
2. CONSTRUCT { ?opinionMaker sd:about ?resource }
3. FROM STREAM <http://streamingsocialdata.org/
  interactions> [RANGE 30m STEP 5m]
4. WHERE {
5.     ?opinionMaker ?opinion ?resource .
6.     ?follower sioc:follows ?opinionMaker.
7.     ?follower ?opinion ?resource.
8.     FILTER ( cs:timestamp(?follower) >
  cs:timestamp(?opinionMaker)
9.         && ?opinion != sd:accesses )
10. }
11. HAVING ( COUNT(DISTINCT ?follower) > 3 )

```

(b)

Figure 3. C-SPARQL samples. (a) The example triples are generated when users interact with resources. (b) The query example identifies users who are opinion makers (that is, who are likely to influence the behavior of their followers).

and line 7 matches their interactions with resources. The FILTER clause uses the custom value testing function *cs:timestamp*, which returns the timestamp of the RDF triple producing the binding. (If the variable gets bound multiple times, the

function returns the most recent timestamp value relative to the query evaluation time.) It checks whether the interactions of the followers occur on the same resource after those of the opinion maker. Timestamps are taken from variables that occur only once in

```

Class( sd:UserOnlyInterestInMovies complete
  intersectionOf(
    sd:User
    restriction(sd:likes allValuesFrom(yago:Movie))
  )
)

```

**(a)**

```

1. REGISTER STREAM MovieOpinionMakers COMPUTED EVERY 5m AS
2. CONSTRUCT { ?opinionMaker sd:about ?resource }
3. FROM STREAM <http://streamingsocialdata.org/interactions> [RANGE 30m STEP 5m]
4. WHERE {
5.     ?opinionMaker a sd:UserOnlyInterestInMovies .
6.     ?opinionMaker ?opinion ?resource .
7.     ?follower sioc:follows ?opinionMaker.
8.     ?follower ?opinion ?resource.
9.     FILTER ( cs:timestamp(?follower) > cs:timestamp(?opinionMaker)
10.            && ?opinion != sd:accesses )
11. }
12. HAVING (COUNT(DISTINCT ?follower) > 3)

```

**(b)**

```

(<:Giulia, sd:likes, :Avatar>,          2010-02-12T13:18:05)
(<:John, sd:likes, :StarWars>,          2010-02-12T13:36:23)
(<:John, sd:likes, :WutheringHeights>,  2010-02-12T13:38:07)
(<:Giulia, sd:likes, :AliceInWonderland>, 2010-02-12T13:42:07)

```

**(c)**

**Figure 4. Example query. (a) A stream reasoner can identify users who are movie opinion makers leveraging the ontological definition (b) within the C-SPARQL query. For instance, (c) if the window on Glue RDF stream contains the triples, Giulia can be a movie opinion maker.**

patterns applied to streaming triples to avoid ambiguity. Also, the query filters out actions of type “accesses” that are normally required before expressing an opinion such as “like” or “dislike.” Finally, the `HAVING` clause distinguishes potential opinion makers from actual opinion makers, checking that at least three followers imitated their behavior.

As an alternative to C-SPARQL, two additional approaches are streaming SPARQL<sup>13</sup> and time-annotated SPARQL.<sup>14</sup> Both languages introduce windows, but only C-SPARQL brings the notion of continuous processing, typical of stream processing, into the language. All other

proposals rely on permanently storing the stream and processing it with one-shot queries. Moreover, only C-SPARQL proposes an extension to SPARQL to support aggregates. This extension permits optimizations that push, whenever possible, aggregate computations as close as possible to the raw data streams.<sup>6</sup>

### C-SPARQL and Deductive Stream Reasoning

Running C-SPARQL queries under expressive OWL reasoning regimes widens the spectrum of analysis that the stream reasoner can perform. For instance, we might define a “movie opinion maker” as an opinion maker

who recently liked only movies. Figure 4a shows the OWL definition of users who like only movies.

This ontological definition can be used in the C-SPARQL query in Figure 4b. For instance, if a window contains the triples in Figure 4c, then Giulia is an instance of `UserOnlyInterestInMovies`, while John is not (he also liked a book).

Evaluating the query in Figure 4 requires reasoning both on the triples in the window and on the background knowledge about objects described in Glue. In particular, the reasoner must check if users match the ontological definition before checking if they are opinion makers. The

deductive stream reasoner must know the ontological definition and combine the RDF stream with relevant background knowledge about movies and books—that is, it must know that *Wuthering Heights* is a book while the other items are movies.

Existing techniques for performing this reasoning task include incremental maintenance of materialized views in logic,<sup>15</sup> graph databases,<sup>16</sup> extensions of the RETE algorithm for incremental rule-based reasoning,<sup>17</sup> and recent attempts to apply incremental reasoning in description logics.<sup>18</sup> All these methods operate incrementally, but none are explicitly dedicated to data stream processing. In a previous work,<sup>7</sup> we proposed a technique for efficiently computing this class of C-SPARQL queries that incrementally maintains a materialization of ontological entailments exploiting the transient nature of streaming data. By adding expiration time information to each RDF triple, we show that it is possible to compute a new complete, correct materialization whenever the window slides by dropping expired statements and entailments and then only adding the deductions that depend on the new triples that entered the window.

### Inductive Stream Reasoning using C-SPARQL

Still wondering about Giulia, we can query which movies Giulia will like the most, even if she has not seen them yet. The answer is built for an ad hoc query; the system uses the last window in the stream to determine such predicted probability (see Figure 5a).

At line 3 in Figure 5a, the construct `WITH PROB` extends SPARQL by letting it query an inducted model. The variable `?prob` assumes the value 1 for the movies she has watched and

```

1. SELECT ?movie ?prob
2. FROM STREAM <http://streamingsocialdata.org/
   interactions> [RANGE 30m STEP 5m]
3. WHERE { :Giulia sd:likes ?movie . WITH PROB ?prob
4.         ?movie a yago_Movie .
5.         FILTER ( ?prob > 0 && ?prob < 1 )
6. } ORDER BY ?prob

(a)

(:WutheringHeightsTvMovie,    0.8347)
(:StarWars,                    0.5693)

(b)

```

**Figure 5. Additional C-SPARQL example. This code shows (a) the C-SPARQL query and (b) its results.**

assumes the estimated probabilities between 0 and 1 for the next movies she would like to watch. The clause `ORDER BY` is used to return movies sorted by decreasing the probabilities. The query answer includes pairs of movie title and predicted likelihood (see Figure 5b).

To run inductive reasoning on semantic data, we use the SUNS learning approach. First, we define the statistical unit, population, sampling procedure, and features. A *statistical unit* is an object of a certain type, such as a user. The *population* is the set of statistical units under consideration. For instance, in the experiments we describe here, we define population as Glue social network users. For training models we sample a subset from the population. Then, based on the sample, the SUNS constructs data matrices by transforming the set of RDF triples related to statistical units into matrices. The rows in the matrix stand for instances of a statistical unit and columns represent their features derived from the associated RDF graph. The binary entries one and zero represent the truth values “true” and “unknown” of the corresponding triples. Suppose that rows are users and columns are movies. A 1 in the  $(i, j)$  entry in the matrix indicates that the  $i$ -th user rates the  $j$ -th movie as liked; otherwise, it is unknown whether that user likes that movie.

After the transformation, we perform a multivariate analysis of the data matrices. Multivariate prediction methods are especially suited for challenging data situations: large scale, multirelational, high dimensional, and highly sparse. The multivariate modeling problem can be solved via singular value decomposition (SVD), nonnegative matrix factorization (NNMF),<sup>19</sup> and latent Dirichlet allocation (LDA).<sup>20</sup> All three approaches estimate unknown matrix entries via a low-rank matrix approximation. NNMF is a decomposition under the constraints that all terms in the factoring matrices are nonnegative, while LDA is based on a Bayesian treatment of a generative topic model. (Recently, we developed a regularized SVD that is rather insensitive on the rank used in the matrix factorization step.) After matrix completion, the 0 entries are replaced with certainty values representing the likelihood that the corresponding triples are true. We have investigated the performance of these methods in offline and online settings, following different sampling strategies.<sup>10</sup> In this context, online setting means that the trained model is applied to predict relationships between entities at query time, including the new entities unseen in the training data set.

```

1. CONSTRUCT {?opinionMaker sd:about ?resource}
2. FROM <http://streamingsocialdata.org/interactions>
3. WHERE {?opinionMaker ?opinion [:about ?resource ;
4.           dc:created ?dateOpinionM .]
5.       ?opinionMaker a sd:UserOnlyInterestInMovies .
6.       ?follower sioc:follows ?opinionMaker .
7.       ?follower ?opinion      [:about ?resource ;
8.           dc:created ?dateOpinionF .]
9.       FILTER (?opinion != sd:accesses) &&
10.          ?dateOpinionM > "2010-02-12T13:00:00Z"^^xsd:dateTime &&
11.          ?dateOpinionM < "2010-02-12T13:30:00Z"^^xsd:dateTime &&
12.          ?dateOpinionF > "2010-02-12T13:00:00Z"^^xsd:dateTime &&
13.          ?dateOpinionF < "2010-02-12T13:30:00Z"^^xsd:dateTime &&
14.          ?dateOpinionF > ?dateOpinionM)
15. ]
16. HAVING (COUNT(DISTINCT ?follower) > 3)

```

Figure 6. The SPARQL query equivalent to the C-SPARQL query shown in Figure 4b. Notably, the C-SPARQL syntax is more handy and terse.

In our example, the user is the main entity of interest (and the reasoner’s statistical unit). Each user is involved in a number of relationships, such as interests in movies, books, and other items; the friendship relationships; and the follows relationship. All data is referred to users and is described by RDF triples, expressing that users “relate” to objects. C-SPARQL continuously delivers new windows of (aggregated) features to the inductive reasoning, and the results of C-SPARQL are transformed into a data matrix, which becomes the input for the inductive reasoner. At predefined time intervals, a learning module applies a multivariate analysis to the data matrices. A second learning module, called the hype model, monitors rapid changes. Two data matrices—a *hype matrix* and a *long-term matrix*—contain short-term trends and long-term information, respectively. The hype matrix is simply populated with the current window content, whereas the long-term matrix is continuously updated and evolves over time.

### Evaluation

To evaluate our approach, we first used a stress test to show its scalability and then evaluated its applicability

to a real case. We show that each architectural component separately applies orthogonal optimizations, yielding an efficient solution when one system’s output is fed as input to the next system.

### Performance and Scalability Evaluation

As in earlier work,<sup>11</sup> we compared a C-SPARQL query’s execution time in our deductive stream reasoner to the execution time of an equivalent SPARQL query on ARQ (<http://jena.sourceforge.net/ARQ>) with inference support. We ran the tests for the query on a Pentium Core 2 Quad 2.0 GHz with a 2-GBbyte RAM.

A little change to the schema to represent interactions allows writing an equivalent SPARQL query (see Figure 6). The code in bold adds two BTPs (lines 4 and 8) that match the creation date of the interaction and four filter conditions (lines 10 through 13) that select the same time interval of the C-SPARQL query. Notably, the C-SPARQL syntax is more handy and terse.

We registered the C-SPARQL query in our engine, fed RDF triples into our engine at a rate of 200 triples per second (t/s), and measured the time

required to compute the answer. Using ARQ, we executed the equivalent SPARQL query six times against repositories containing a growing number of triples and again measured the time required to compute each answer.

Figure 7 shows the results. By comparing the linear regressions of the two experiments—Linear(SPARQL) and Linear(C-SPARQL 200 t/s)—we see that the C-SPARQL window-based selection performs significantly better than the FILTER-based selection of SPARQL in Jena.

### Evaluation on a Real Case Scenario

To prove the effectiveness of stream reasoning for social media analytics, we evaluated the accuracy of top-*N* movie recommendations. First, we compared diverse inductive reasoning approaches with common recommendation methods, some of which were realized by deductive stream reasoning. Second, we examined the performance of the combination of both inductive and deductive streaming reasoning.

To gather a data set for the evaluation, we used a predefined C-SPARQL query and then transcoded and stored the output RDF streams into

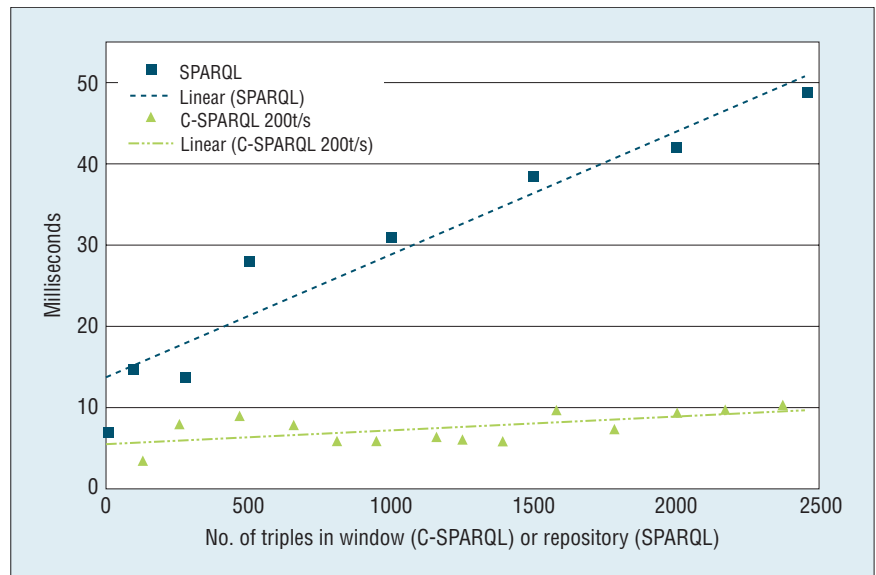
a data matrix. The matrix was continuously updated between 19 February to 22 April 2010. Finally, we selected 245,860 interactions made by 2,457 users. In particular, we examined the interactions of the “liked movies” relationship. The transformed data matrix was extremely sparse with only 0.002 percent non-zero elements. To make statistically significant evaluations, we removed all users with almost no interactions and items that were evaluated less than five times.

After pruning, the resulting subset consisted of 1,455 users and 7,724 features, with a sparsity of 0.02 percent. The item most specified by users was the “liked movies” relation with 2,467 movies. “Liked music,” “liked recording\_artists,” “likedmovie-stars,” “liked tv\_shows,” and “liked video\_games” were specified 1,378, 1,241, 592, 592 and 579 times, respectively. The remaining 18 features were mentioned less than 250 times.

Since they are the most easily adaptable to the dynamic setting, we applied SVD and regularized SVD for movie recommendations. As baseline methods, we first used a global “liked movie” list carried out by a simple registered C-SPARQL query (see Figure 8).

Second, we extracted the “most liked” movies of a person’s friends, also calculated via a corresponding registered C-SPARQL query (not shown). Third, we applied the k-nearest neighbour (kNN) regression, using the same user-based and movie-based similarity measures.<sup>21</sup> We carefully tuned parameters of each method using cross validation.

Figure 9a shows the evaluation results—the percentage of truly liked movies in the top  $N$  recommendations where  $N = 10, 20, 30, 40,$  and  $50$ . First, SVD and regularized SVD outperformed all baseline methods. In



**Figure 7. Stress test results. The window-based selection of C-SPARQL outperforms the *FILTER*-based selection of SPARQL.**

```

1. REGISTER STREAM MostLiked COMPUTED EVERY 1d AS
2. SELECT ?movie (COUNT(?user) AS ?noOfUser)
3. FROM STREAM <http://streamingsocialdata.org/
  interactions> [RANGE XX STEP XX]
4. WHERE {?movie a yago_Movie .
5.         ?user sd:likes ?movie .}
6. GROUP BY ?movie
7. ORDER BY DESC(?nrOfUser)

```

**Figure 8. A simple registered C-SPARQL query. This query returns a global “liked movie” list.**

particular, the regularized SVD performed much better than any other method and was robust and insensitive to its parameters, as expected. Second, both kNN lines are above the baselines, meaning that the users and the items collected share some common regularity. For example, users who like the same actors are likely to watch movies that feature them. Taking such additional features into account significantly improves the accuracy of movie recommendations. Of course, SVD and regularized SVD exploit the data regularity as well. Third, the two baseline methods almost completely overlapped. The reason might be that most users would like to watch movies from the same global list of the most-popular movies

rather than considering their friends’ preferences.

In the second part of our empirical study, we experimented with combining the output of the deductive and the inductive reasoning module. In this scenario, the inductive module models long-term user preferences, which in this experiment, is trained only on data more than 30 days old. It is quite reasonable to assume that the long-term preference model is updated only at larger intervals because the required computations can be quite costly if we avoid subsampling. The deductive reasoning module contributes predictions in the form of “most liked,” which simply aggregates recent recommendations to capture the short-term trends of “hype.”



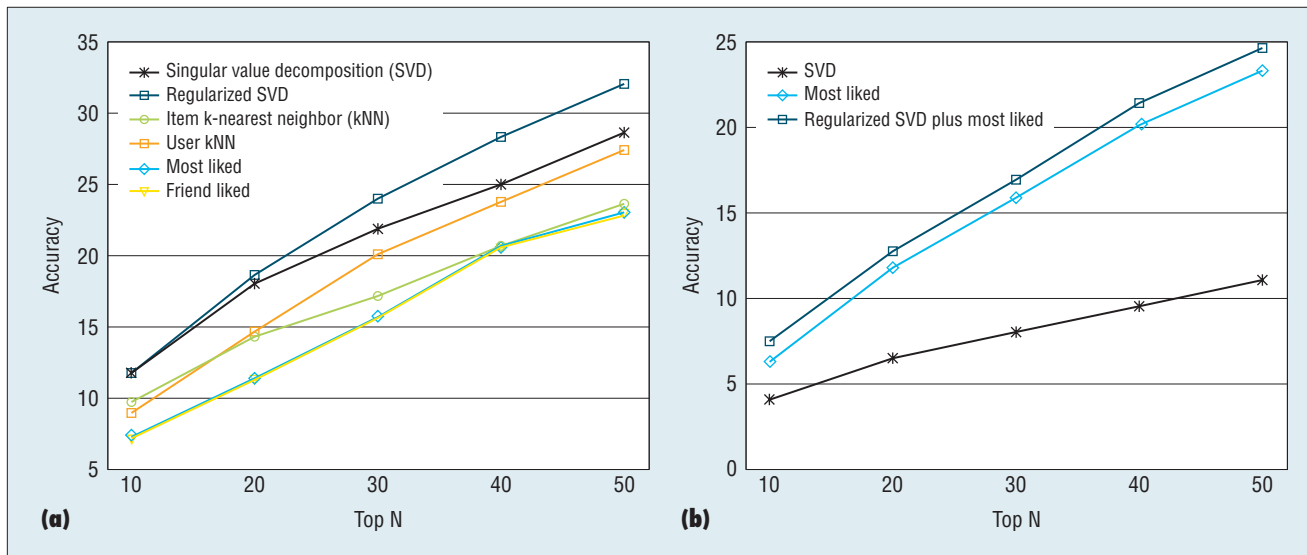


Figure 9. Accuracy of top- $N$  movie recommendations. (a) Inductive stream reasoning and deductive stream reasoning separately. (b) Inductive and deductive stream reasoning combined.

Strictly speaking, the latter is also an inductive process, but the deductive component inherently supports aggregation as well.

The short-term trend could have been predicted by a multivariate analysis similar to the long-term module and combined in a comparable way to the deductive module's output. However, Figure 9b clearly shows that the combination of the long-term inductive model and the "most liked" deductive model outperforms both separated methods. Experiments with more sophisticated hype modules and the exploration of different combination schemes are part of future work.

This article illustrates a sequential integration between two existing stream reasoning environments within the LarKC platform. However, the LarKC pluggable architecture also allows for other forms of integration with reasoners sharing the same RDF resources, freely reacting to RDF streams, and mutually interacting.

Interesting future work would be applying the integrated reasoning framework to other social networks, such as Twitter and Facebook.

Temporal aspects of relational learning are currently finding increasing interest. Ongoing work concerns novel extensions of the SUNS approach to handle time-dependent semantic data sets and the C-SPARQL approach to support more expressive entailment regimes. ■

### Acknowledgments


This work was partially supported by the European project LarKC (FP7-215535).

### References

1. M. Garofalakis et al., *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*, Springer-Verlag, 2007.
2. J.H. Su et al., "Music Recommendation Using Content and Context Information Mining," *IEEE Intelligent Systems*, vol. 25, no. 1, 2010, pp. 16–26.
3. E. Della Valle et al., "It's a Streaming World! Reasoning upon Rapidly Changing Information," *IEEE Intelligent Systems*, vol. 24, no. 6, 2009, pp. 83–89.
4. S. Babu and J. Widom, "Continuous Queries over Data Streams," *Sigmod Record*, vol. 30, no. 3, 2001, pp. 109–120.
5. D. Fensel et al., "Towards LarKC: A Platform for Web-Scale Reasoning," *Proc. IEEE Int'l Conf. Semantic Computing (ICSC 08)*, IEEE Press, 2008, pp. 224–229.
6. D. Barbieri et al., "An Execution Environment for C-SPARQL Queries," *Proc. Int'l Conf. Extending Database Technology (EDBT 2010)*, ACM Press, 2010, pp. 441–452.
7. D. Barbieri et al., "Incremental Reasoning on Streams and Rich Background Knowledge," *Proc. Extended Semantic Web Conf. (ESWC 2010)*, 2010.
8. D.F. Barbieri et al., "C-SPARQL: SPARQL for Continuous Querying," *Proc. 18th Int'l World Wide Web Conf. (WWW 09)*, ACM Press, 2009, pp. 1061–1062.
9. V. Tresp et al., "Materializing and Querying Learned Knowledge," *Proc. 1st ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web (IRMLes 2009)*, 2009; doi=10.1.1.149.6151.
10. Y. Huang et al., "Multivariate Prediction for Learning on the Semantic Web," *Proc. 20th Int'l Conf. Inductive Logic Programming (ILP 2010)*, preprint, 2010, doi:10.1109/MIS.2010.111.
11. D.F. Barbieri et al., "Continuous Queries and Real-Time Analysis of Social Semantic Data with C-SPARQL," *Proc. Social Data on the Web Workshop*, 2009.

## THE AUTHORS

12. U. Bojars et al., "Interlinking the Social Web with Semantics," *IEEE Intelligent Systems*, vol. 23, no. 3, 2008, pp. 29–40.
13. A. Bolles et al., "Streaming SPARQL: Extending SPARQL to Process Data Streams," *Proc. European Semantic Web Conf. (ESWC 2008)*, Springer-Verlag, 2008, pp. 448–462.
14. A. Rodriguez et al., "Semantic Management of Streaming Data," *Proc. Intl. Workshop on Semantic Sensor Networks (SSN)*, 2009; <http://cet.ncsa.uiuc.edu/publications/SemanticSN2009streaming.pdf>.
15. R. Volz et al., "Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases," *J. Data Semantics II*, vol. 3360, 2005, pp. 1–34.
16. Y. Zhuge and H. Garcia-Molina, "Graph Structured Views and their Incremental Maintenance," *Proc. 14th Int'l Conf. Data Eng.*, IEEE Press, 1998, pp. 116–125.
17. F. Fabret et al., "An Adaptive Algorithm for Incremental Evaluation of Production Rules in Databases," *Proc. 19th Int'l Conf. Very Large Data Bases (VLDB 1993)*, Morgan Kaufmann, 1993, pp. 455–466.
18. B. Cuenca-Grau et al., "History Matters: Incremental Ontology Reasoning using Modules," *Proc. 6th Int'l Semantic Web Conf. (ISWC 2007)*, Springer-Verlag, 2007, pp. 183–196.
19. D.D. Lee and H.S. Seung, "Learning the Parts of Objects by Non-negative Matrix Factorization," *Nature*, vol. 401, 21 Oct. 1999, pp. 788–791.
20. D.M. Blei et al., "Latent Dirichlet Allocation," *J. Machine Learning Research*, vol. 3, Jan. 2003, pp. 993–1022.
21. G. Linden et al., "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *IEEE Internet Computing*, vol. 7, no. 1, 2003, pp. 76–80.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

**Davide Barbieri** is a PhD student in the Department of Electronics and Information (DEI) at the Polytechnic of Milan, Italy. His research interests address issues related to stream processing, especially C-SPARQL, which will be the main topic of his doctoral dissertation. Barbieri has a MSc in computer science from Polytechnic of Milan. Contact him at [davide.barbieri@polimi.it](mailto:davide.barbieri@polimi.it).

**Daniele Braga** is an assistant professor in the Department of Electronics and Information (DEI) at the Polytechnic of Milan. His research interests include issues related to the manipulation of semistructured data (visual languages and advanced processing for XML data), service integration (with specific reference to complex queries over heterogeneous Web data sources), and stream reasoning (C-SPARQL and reasoning over streaming data). Braga has a PhD in computer science from the Polytechnic of Milan. Contact him at [daniele.braga@polimi.it](mailto:daniele.braga@polimi.it).

**Stefano Ceri** is a professor of database systems in the Department of Electronics and Information (DEI) at the Polytechnic of Milan. His research work covers extending database technologies to incorporate new features (distribution, object orientation, rules, and streaming data) as well as the engineering of Web-based applications and complex search systems. Ceri has a PhD in electronics from Polytechnic of Milan. Contact him at [stefano.ceri@polimi.it](mailto:stefano.ceri@polimi.it).

**Emanuele Della Valle** is an assistant professor of software project management in the Department of Electronics and Information (DEI) at the Polytechnic of Milan. His research interests include scalable processing of information at the semantic level and stream reasoning. Della Valle has a MSc in computer science from Polytechnic of Milan. Contact him at [emanuele.dellavalle@polimi.it](mailto:emanuele.dellavalle@polimi.it).

**Yi Huang** is a staff scientist at Siemens Corporate Research and Technology and is finishing his PhD at Ludwig Maximilian University of Munich, Germany. His research interests focus on statistical machine learning, text mining, information retrieval, and the Semantic Web. Huang has a Diploma in computer science from Ludwig Maximilian University of Munich. Contact him at [yihuang@siemens.com](mailto:yihuang@siemens.com).

**Volker Tresp** is the head of a research team in machine learning at Siemens Corporate Research and Technology. His research interests include machine learning, data mining, information extraction, and the Semantic Web. Tresp has a PhD from Yale University. Contact him at [volker.tresp@siemens.com](mailto:volker.tresp@siemens.com).

**Achim Rettinger** is a project manager and assistant professor at the Institute of Applied Informatics and Formal Description Methods (AIFB) at the Karlsruhe Institute of Technology (KIT, Germany). His research interests include combining machine learning, knowledge discovery, and human computer systems with semantic technologies. Rettinger has a PhD in computer science from the Technische Universität München. Contact him at [rettinger@kit.edu](mailto:rettinger@kit.edu).

**Hendrik Wermser** is pursuing his master's at the Technical University of Munich. His research interests are in information retrieval, recommender systems, and machine learning. Wermser has a BS in computer science from the Technical University of Munich. Contact him at [wermser@cs.tum.edu](mailto:wermser@cs.tum.edu).

**Intelligent  
IEEE  
Systems**  
**THE #1 ARTIFICIAL  
INTELLIGENCE  
MAGAZINE!**

*IEEE Intelligent Systems* delivers the latest peer-reviewed research on all aspects of artificial intelligence, focusing on practical, fielded applications. Contributors include leading experts in

- Intelligent Agents • The Semantic Web
- Natural Language Processing
- Robotics • Machine Learning

Visit us on the Web at  
[www.computer.org/intelligent](http://www.computer.org/intelligent)