Clustering Remote RDF Data Using SPARQL Update Queries

Letao Qi, Harris Lin, Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames, Iowa, USA 50011
{leetaoqi, htlin, honavar}@iastate.edu
Overview

• Introduction
  ➢ RDF and SPARQL
  ➢ Background
  ➢ Limitation of Existing Approaches
  ➢ Contributions
  ➢ A Motivating Example

• Clustering Algorithm
  ➢ K-means Clustering Using SPARQL

• Experiment
  ➢ Datasets
  ➢ Design
  ➢ Results

• Conclusion
  ➢ Summary
  ➢ Future Work
Resource Description Framework (RDF)

- Resource Description Framework (RDF)
  - Structured and linked information on the Web in the form of directed labeled graphs
  - Represented as sets of subject-predicate-object triples (RDF triples)

- Example

![RDF Example Diagram]

RDF triple (Letao, hasMajor, Com S)

- An RDF graph is a set of RDF triples.
Resource Description Framework (RDF) Example

**RDF Schema**

![RDF Schema Diagram](image)

**RDF Data (Graph representation)**

- Letao
- Harris
- Photo1
- Photo2
- Tag1
- Tag2
- Tag3

**RDF Data (Triple representation)**

- (Letao, hasFriend, Harris)
- (Letao, hasPhoto, Photo1)
- (Letao, hasPhoto, Photo2)
- (Photo1, hasTag, Tag1)
- (Photo1, hasTag, Tag2)
- (Photo2, hasTag, Tag1)
- (Photo2, hasTag, Tag3)
SPARQL

- SPARQL Query and SPARQL Update
  - offer the means to query and update large amounts of RDF data.
Background

- Linked Open Data cloud: about 300 interlinked datasets containing over 31 billion RDF triples
  - Massive, Remote, Distributed, Inter-linked dataset
- Goal: extract useful knowledge using machine learning techniques

We consider clustering: assigns similar individuals into the same group
  - similarity depends on the features available for each individual as well as the function applied on them
Limitation of Existing Approaches

• Existing approaches to clustering RDF data assume direct access to data, however it is often not desirable to transfer a remote dataset locally because:
  i. High cost in communication
  ii. May not fit in memory
  iii. Data dump may not be available
  iv. Data may be privacy protected

• Problem: How to cluster RDF data without direct access to data?
  I. naive solution - download all dataset
  II. proposed solution - using update queries, and only some data (statistics) are retrieved by the algorithms
Contributions

• We adapted two clustering algorithms:
  ➢ **K-means (talk about this today)**
  ➢ Multi-Resolution Overlapping Communities approach (MROC, an adaption of agglomerative hierarchical clustering algorithm)

• We implemented these algorithms using SPARQL update queries, and experimentally evaluated against a real social network dataset

• We compared our approach against the naive alternative which downloads the dataset and execute the algorithm locally (time and communication cost)
A Motivating Example

- A social network dataset crawled from Flickr
  - A social photo sharing website where each user has a set of photos and each photo is tagged by a set of tags
  - Each user has multiple friends

RDF schema of Flickr dataset
Overview

• Introduction
  ➢ RDF and SPARQL
  ➢ Background
  ➢ Limitation of Existing Approaches
  ➢ Contributions
  ➢ A Motivating Example

• Clustering Algorithm
  ➢ K-means Clustering Using SPARQL

• Experiment
  ➢ Datasets
  ➢ Design
  ➢ Results

• Conclusion
  ➢ Summary
  ➢ Future Work
K-means Clustering Using SPARQL

- The input for k-means is \((U, K)\) such that (i) \(U\) is a data set containing \(N\) users where each user \(u_i\) is described by \(D\) feature values; and (ii) \(K\) is the user-specified number of clusters.

- The output is a set of prototypes (centroids) \(P\) and the assignment of each instance to a prototype.

- Algorithm:
  - Initialization: create \(P\) with \(K\) users randomly selected from \(U\)
  - Repeat E step and M step
    - Expectation: for each user, find the nearest prototype and update its assignment
    - Maximization: update each prototype with the average of users assigned to it
K-means Clustering Using SPARQL (cont.)

- We denote each user as a count vector.

![Diagram of user, photo, and tag relationships]

Based on the dictionary \{Tag1, Tag2, Tag3\},
Count vector representation: \{2, 1, 1\}

Corresponding SPARQL Update Queries

```sparql
insert {graph <:users> {?u ?t ?count}}
where
{
  {select ?u ?t count(?t) as ?count
  where
  ?p <http://flickr/vocab/hasTag> ?t.}}
  group by ?u ?t}
}
```
K-means Clustering Using SPARQL (cont.)

- **Preprocess**: We preprocess the dataset using count aggregation queries, and represent each instance as a vector of feature values of length $D$ into a new graph called $\langle \text{users} \rangle$.

```
Flickr Data

Preprocess

Initialization

\langle \text{users} \rangle

\langle \text{prototypes} \rangle

\langle \text{distances} \rangle

\langle \text{assignments} \rangle

M step

E step

E step

\text{(user, prototype, distance)}

\text{(user, <hasPrototype>, prototype)}

\text{(user, tag, tag-count)}

\text{(prototype, tag, tag-count)}
```
Overview

• Introduction
   RDF and SPARQL
   Background
   Limitation of Existing Approaches
   Contributions
   A Motivating Example

• Clustering Algorithm
   K-means Clustering Using SPARQL

• Experiment
   Datasets
   Design
   Results

• Conclusion
   Summary
   Future Work
Experiment - Dataset

- The dataset is crawled from a photo sharing network Flickr.
- We take 10 snapshots of the datasets from 1k, 2k, ..., and finally 10k users.

RDF schema of Flickr dataset

<table>
<thead>
<tr>
<th>Dataset snap shot</th>
<th>#Users</th>
<th>#Friendship edges</th>
<th>#Distinct photos</th>
<th>#Distinct tags</th>
<th>#Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,000</td>
<td>34,935</td>
<td>275,895</td>
<td>4,760</td>
<td>2,506,625</td>
</tr>
<tr>
<td>2</td>
<td>2,000</td>
<td>62,586</td>
<td>569,848</td>
<td>4,869</td>
<td>5,278,995</td>
</tr>
<tr>
<td>3</td>
<td>3,000</td>
<td>86,878</td>
<td>776,993</td>
<td>4,898</td>
<td>7,005,817</td>
</tr>
<tr>
<td>4</td>
<td>4,000</td>
<td>103,998</td>
<td>966,160</td>
<td>4,925</td>
<td>8,476,583</td>
</tr>
<tr>
<td>5</td>
<td>5,000</td>
<td>115,817</td>
<td>1,163,714</td>
<td>4,956</td>
<td>10,121,011</td>
</tr>
<tr>
<td>6</td>
<td>6,000</td>
<td>124,891</td>
<td>1,337,040</td>
<td>4,974</td>
<td>11,468,325</td>
</tr>
<tr>
<td>7</td>
<td>7,000</td>
<td>132,128</td>
<td>1,499,324</td>
<td>4,985</td>
<td>12,756,597</td>
</tr>
<tr>
<td>8</td>
<td>8,000</td>
<td>137,812</td>
<td>1,630,991</td>
<td>4,989</td>
<td>13,752,406</td>
</tr>
<tr>
<td>9</td>
<td>9,000</td>
<td>142,348</td>
<td>1,725,764</td>
<td>4,992</td>
<td>14,445,663</td>
</tr>
<tr>
<td>10</td>
<td>10,000</td>
<td>144,860</td>
<td>1,825,092</td>
<td>4,992</td>
<td>15,125,463</td>
</tr>
</tbody>
</table>
Experiment – Design (K-means)

- We measure and compare the time and communication complexity taken by several alternatives for each clustering algorithm:
  
  1) K-means
     i. [Local]: Transfer the RDF data dump, preprocess locally then run Weka’s SimpleKMeans function
     ii. [PreQuery+Local]: Use queries to preprocess the tags data into count-vector representation, then transfer the preprocessed count and run Weka locally
     iii. [Query] Use queries to execute the entire algorithm

For all executions we fix K to be 10 clusters, and a maximum of 20 iterations.
Experiment – Results (K-means)

Processing time complexity and communication complexity results for K-Means clustering algorithm.
Experimentation – Results (K-means) (cont.)

- We assume a 2M/s download speed.

*Total time complexity (measured processing time plus estimated data transfer time) results for K-Means*
Overview

- **Introduction**
  - RDF and SPARQL
  - Background
  - Limitation of Existing Approaches
  - Contributions
  - A Motivating Example

- **Clustering Algorithm**
  - K-means Clustering Using SPARQL

- **Experiment**
  - Datasets
  - Design
  - Results

- **Conclusion**
  - Summary
  - Future Work
Summary

• LOD cloud calls for approaches to extract useful knowledge using machine learning techniques (we considered clustering)
  ➢ RDF data are massive and remote, and we want to avoid gathering the datasets onto a centralized location for analysis

• We show how to implement two representative clustering algorithms using SPARQL update queries

• We compare the time and communication complexity of our algorithms with of those that require direct centralized access to the data

• We conduct experiments on a real social network dataset and report our preliminary findings
Future Work

1) Allowing the creation of user defined stored procedures on the RDF data source
2) Considering MapReduce implementation to answer those queries proposed in this work
3) Extending our approach into multiple remote and distributed RDF data sources
The End
Thank you!