Unsupervised Entity Resolution on Multi-type Graphs

Lab Report: Distributed Big Data Analytics, WS2017-2018

Masters of Science, Universität Bonn, Germany

Xhulja Shahini, Ardit Meti

S6xhshah@uni-bonn.de
S6armeti@uni-bonn.de

Supervisors:
Dr.Hajira Jabeen, Gezim Sejdiu

Abstract

Entity resolution is the task of identifying all mentions that represent the same real-world entity within a knowledge base or across multiple knowledge bases[1]. We address the problem of performing entity resolution on RDF graphs containing multiple types of nodes, using the links between instances of different types to improve the accuracy. For example, in a graph of products and manufacturers the goal is to resolve all the products and all the manufacturers. We formulate this problem as a multi-type graph summarization problem, which involves clustering the nodes in each type that refer to the same entity into one super node and creating weighted links among super nodes that summarize the inter-cluster links in the original graph. Experiments show that the proposed approach outperforms several state-of-the-art generic entity resolution approaches, especially in data sets with missing values and one-to-many, many-to-many relations.

1. Problem Definition

1.1 Introduction

The increasing number of entities created online raises the problem of integrating and relating entities from different sources. In this work, we focus on the entity resolution problem. The idea of this project is to automatically group elements that correspond to the same entity.

[1] http://usc-isi-i2.github.io/papers/zhu16-iswc.pdf

We model the observed RDF graph as a multi-type graph and formulate the collective entity resolution as a multi-type graph summarization problem. We thus propose a unified, multi-type graph co-summarization based entity resolution framework (CoSum), which jointly condenses a set of similar vertices in the observation into a super node in the summary graph so that each super node (hidden entity) is coherent, reveals how entities of different types are related with each other.

1.2 Challenges

There are two main challenges in tacking the entity resolution problem:

1. The poor quality of data and ambiguity

2. The heterogeneous nature of relationships

The first challenge consists on different or incorrect spelling of data, missing values, or having information that can be interpreted in various ways. Having such noise in the content and context makes it very difficult to properly calculate the pair-wise distance between entities.
The second challenge is due to the one-to-many and many-to-many relation between entities. This heterogeneous nature of relations makes it challenging to determine which kind of relationships suits these entities better and thus is more difficult to perform collective entity resolution.
To solve the challenges mentioned above we model the observed relations between different types of mentions as a multi-type graph and reduce the entity resolution to a graph summarization problem.
We validate the proposed approach on real-world networks from both an E-commerce and a citation domain. The results show that the proposed approach outperforms other state-of-the-art approaches.

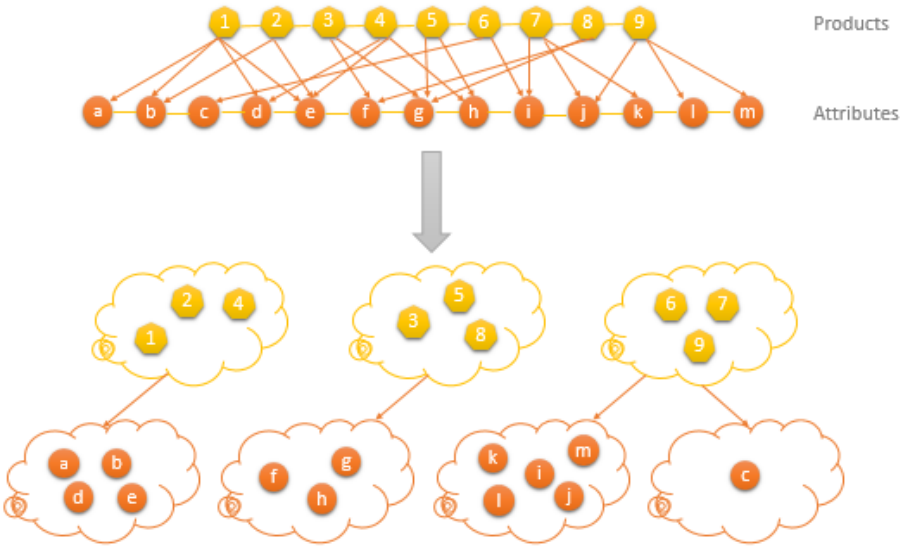

Figure1. Summary Graph out of multi-type graph

2. Approach

To implement the solution we have used Scala programming language with Spark framework. We model the observed RDF graph as a multi-type graph, where vertices represent different types of objects, and edges represent either co-occurrence between two-types of vertices, or similarity between the same-type vertices. The weights of the super edges indicate which type of information is more useful when resolving certain types of entities. Then we formulate the collective entity resolution as a multi-type graph summarization problem. Particularly, the goal is to transform the original k-type graph into another k-type summary graph composed of super nodes and super edges. Each super node is a cluster of original vertices (of the same type) representing a latent entity, while super edges encode potentially valuable relations between those entities.

The overview of our solution is as follows:

Start with a random summary graph, we first search for an improved summary graph with fewer super nodes, by crossing out one or many super nodes. The second step is to fix the number of super nodes [p1, · · · , pk], and compute the vertex-to-clustering mapping C and super links L. These two procedures are performed alternately, until they reach a locally optimal summary graph.

Algorithm 1 The graph summarization framework for k-partite graphs

Input: A k-type Graph G
Output: A k-type summary graph $\mathcal{S}(G)$
01: Initialize a random k-type summary graph, with number of super nodes $[n_1, \cdots, n_k]$
02: **repeat**
/$*$ vertex allocation optimization (Section 4.3)$*$/
03: $\mathcal{S}(G)$=`Search`(G, $\mathcal{S}(G)$) (see Alg. 2)
/$*$ fix the number of super nodes, and optimize super nodes assignment (Section 4.2)$*$/
04: **do**
05: **for** each t-type vertices
06: update C_t with Eq. (4)
07: **for** each non-empty edge set between t- and t'-type vertices
08: update $L_{tt'}$ with Eq. (5)
09: **while** C and L converge
10: construct the new summary graph $\mathcal{S}(G)$
11: **until** $J(\mathcal{S}(G))$ converges
12: **return** $\mathcal{S}(G)$

Figure2. The algorithm

3. Implementation

We have implemented this algorithm in two different ways in a distributed way, using COO¶ format and locally using Breeze¶ and MKL¶ library.

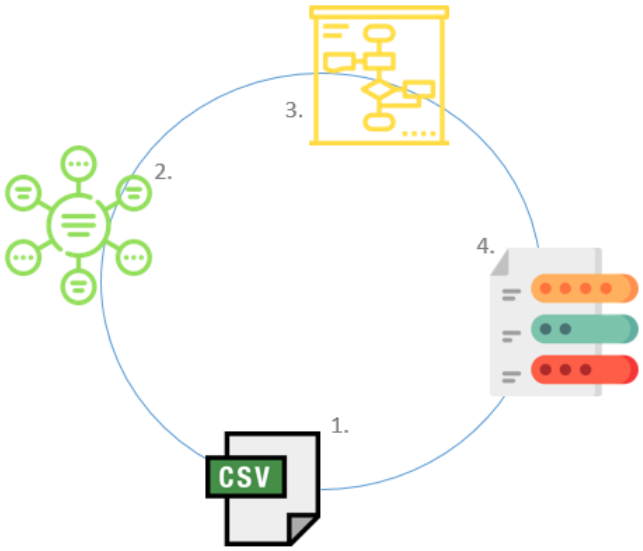

Figure 3

1. We read the input data from an Excel file. Our input data consists of three main files in TDIDF¶ format: Product2Product (containing similarities between the first type of nodes which are product's names) Attribute2 Attribute (containing similarities between second type of nodes which are words) Product2Attributes (a graph containing connection relations between first type and second type of nodes).
2. After reading the file we create the respective RDD Data types. Coordinate Matrices for the distributed solution and Local Matrices¶ for the second implementation.
3. We apply the Summarization Function algorithm to compute the probabilities of original vertices to belong to the super nodes of the summarization graph.
4. We output the final result in an Excel sheet.

3.1 Tools

3.1.1 TFIDF

TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus[2].

Term frequency

In the case of the term frequency tf(*t*,*d*), the simplest choice is to use the *raw count* of a term in a document, i.e. the number of times that term *t* occurs in document *d*. If we denote the raw count by $f_{t,d}$, then the simplest tf scheme is tf(*t*,*d*) = $f_{t,d}$. Other possibilities include

- Boolean "frequencies": tf(*t*,*d*) = 1 if *t* occurs in *d* and 0 otherwise;
- term frequency adjusted for document length : $f_{t,d}$ ÷ (number of words in d)
- logarithmically scaled frequency: tf(*t*,*d*) = log (1 + $f_{t,d}$), (or zero if $f_{t,d}$ is zero);
- augmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the raw frequency of the most occurring term in the document:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

Inverse document frequency

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

With:

- N : total number of documents in the corpus N = |D|
- |{d ∈ D : t ∈ D}| : number of documents where the term t appears (i.e., tf(t , d) ≠ 0). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to 1+|{d ∈ D : t ∈ D}| .

3.1.2 Coordinate Matrix

[2] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

A CoordinateMatrix (COO) is a distributed matrix backed by an RDD of its entries[3]. Each entry is a tuple of (i: Long, j: Long, value: Double), where i is the row index, j is the column index, and value is the entry value. A CoordinateMatrix should be used only when both dimensions of the matrix are huge and the matrix is very sparse.
A CoordinateMatrix can be created from an RDD[MatrixEntry] instance, where MatrixEntry is a wrapper over (Long, Long, Double). A CoordinateMatrix can be converted to an IndexedRowMatrix with sparse rows by calling toIndexedRowMatrix.

3.1.3 Local Matrix

Spark supports local vectors and matrices stored on a single machine, as well as distributed matrices backed by one or more RDDs[4]. Local vectors and local matrices are simple data models that serve as public interfaces. The underlying linear algebra operations are provided by Breeze and jblas. A local vector has integer-type and 0-based indices and double-typed values, stored on a single machine. Spark supports two types of local vectors: dense and sparse. A dense vector is backed by a double array representing its entry values, while a sparse vector is backed by two parallel arrays: indices and values. For example, a vector (1.0, 0.0, 3.0) can be represented in dense format as [1.0, 0.0, 3.0] or in sparse format as (3, [0, 2], [1.0, 3.0]), where 3 is the size of the vector.

3.1.4 Breeze

Breeze is a library for numerical processing. It aims to be generic, clean, and powerful without sacrificing (much) efficiency[5].
Compared to other numerical computing environments, Breeze matrices default to column major ordering, like Matlab, but indexing is 0-based, like Numpy. Breeze has as its core concepts matrices and column vectors. Row vectors are normally stored as matrices with a single row. This allows for greater type safety with the downside that conversion of row vectors to column vectors is performed using a transpose-slice (a.t(::,0)) instead of a simple transpose (a.t).

3.1.5 MKL

Intel Math Kernel Library (Intel® MKL) optimizes code with minimal effort for future generations of Intel processors[6]. It is compatible with your choice of compilers, languages, operating systems, and linking and threading models.

- Features highly optimized, threaded, and vectorized math functions that maximize performance on each processor family

[3] https://stanford.edu/~rezab/papers/linalg.pdf
[4] https://stanford.edu/~rezab/papers/linalg.pdf
[5] https://github.com/scalanlp/breeze/wiki/Breeze-Linear-Algebra
[6] https://software.intel.com/en-us/mkl

- Uses industry-standard C and Fortran APIs for compatibility with popular BLAS, LAPACK, and FFTW functions—no code changes required
- Dispatches optimized code for each processor automatically without the need to branch code
- Provides Priority Support that connects you directly to Intel engineers for confidential answers to technical questions

3.2 Implementation with COO

A distributed matrix has long-typed row and column indices and double-typed values, stored distributively in one or more RDDs. It is very important to choose the right format to store large and distributed matrices. Converting a distributed matrix to a different format may require a global shuffle, which is quite expensive.
In this approach we have used CoordinateMatrix to store the input and compute the calculations. In order to do the computations we had to implement ourselves many Linear Algebraic functions.

Advantages: We can use matrices without size limitations.
Disadvantages**:** RDD's are immutable. New RDDs have to be created each time that we want to update one single index. Linear Algebra functions are not supported and converting to data structures which support them is quite expensive.

3.3 Implementation with Breeze

The algorithm used in this implementation is exactly same as the one used in the first implementation. The reason why we decided to do a second implementation was that Breeze uses DenseMatrix, which is mutable and you can update them efficiently. Also it has many libraries that support different Linear Algebra functions like creation of identity matrices, element-wise multiplication, division, etc.
In order to speed up the processing time of the matrices conversions we had to use MKL library.

Advantages: Breeze uses DenseVector which is mutable. It supports Linear Algebra functions.
Disadvantages**:** Takes a long time to compute big-sized matrices multiplication. Another library MKL is needed to speed up the processing of matrices.

4. Evaluation

4.1 Complexity Analysis

We have analysed the time complexity of our graph summarization algorithm for each basic operator with both dense and sparse matrices (COO) representation.

	Dense	Sparse
C_t	$O(n_t^2 p_t + n_t \sum_{t'>t}(n_t p_t + p_t p_t))$	$O((nz)_t p_t + \sum_{t'>t}(m_{tt'} + n_t q_{tt'}))$
$L_{tt'}$	$O(n_t p_t^2 + p_t(n_t n_{t'} + p_{t'} n_{t'} + p_t p_{t'}))$	$O(n_t p_t^2 + p_t(m_{tt'} + q_{tt'} + p_t n_{t'}))$

Figure4. Complexity analysis

Here nz is the number of non-zero entries in the matrix of similarities.

4.2 Evaluation using F-Measure

To evaluate our algorithm we decided to use F-measure metric. We have done the calculations for different input size, but only for the second algorithm using Breeze. Due to high space and time complexity we found it unreasonable to evaluate the first algorithm without doing first some necessary improvements, which are part of our future work.

Precision ≈ 0.75 Recall ≈ 0.5 F-measure = 0.6

Map	ID	Title	Description	Manufacturer	Price
0	b000jz4hqo	clickart 950 000 - premier image pack (dvd-rom)		broderbund	0
1	b0006zf55o	ca international - arcserve lap/desktop oem	oem arcserve backup v11.1 ...	computer associates	0
2	b00004tkvy	noah's ark activity center (jewel case ages 3-8)		victory multimedia	0
3	b000g80lqo	peachtree by sage premium accounting for nonprofits 2007	peachtree premium ...	sage software	599.99
4	b0006se5bq	singing coach unlimited	singing coach unlimited ...	carry-a-tune technologies	99.99
5	http://www.google.com/base/feeds/snippets/18441480711193821750	clickart 950000 - premier image pack (dvd-rom)	collection of images & fonts...		48.95
6	b00021xhzw	adobe after effects professional 6.5	upgrade only; installation of after effects...	Adobe	499.99
7	http://www.google.com/base/feeds/snippets/18441188461196475272	sage (ptree) - vernfp2007rt - premium	peachtree premium ...		590.35
8	http://www.google.com/base/feeds/snippets/18428750969726461849	singing coach unlimited - electronic learning products	real-time pitch recognition...		82.5
9	http://www.google.com/base/feeds/snippets/18430621475529168165	adobe software 22070152 after effects 6.5	adobe after effects...		507

Table 1: Input example with 10 entries

3	*0*	*1*
7	*0*	*1*
4	*1*	*1*
6	*1*	0.271327
8	*1*	*1*
9	*1*	0.278596
0	*2*	*1*
5	*2*	*1*
1	*3*	*1*
2	*4*	*1*
6	*4*	0.728673
9	*4*	0.721404

Table 2: Output example with 10 entries

Explanatory Note!

- First column indicates the ID's of the products
- Second column indicates the ID's of super nodes
- Third column indicates the probabilities of each product to belong in that super node

5. Project Timeline

Week 1:

- Read the paper
- Understood the problem
- Divided the tasks

Week 2:

- Studied Scala/ Spark
- Discussions about the right way to approach the solution with the mentors
- Analysed the Algorithm efficiency and improvements that could be done

Week 3:

- Explored the features of Spark
- Started implementing the algorithm using distributed RDD
-

Week 4:

- Dealing with problems in implementation.
- Finished the first implementation algorithm

Week 5:

- Came up with the idea of implementing it locally.
- Studied Breeze library and explored its functionalities.

Week 6:

- Started implementing the second solution using Breeze and Local Matrices.
- Debugging
- Optimized it

Week 7:

- Looking for other additional optimization functionalities
- Started working on the report.

Week 8:

- Final work on the code, adding comments and making it more readable
- Finished the report

6. Future Work and Improvements

The input files that we actually use are TFIDF format, but in the future we are planning to work with raw data (Ex: title, name, manufacturer, price) and we will have to generate the TFIDF format input out of these data, this should be done automatically for various graph raw data.
Our future work includes optimizing the first algorithm that was implemented using CoordinateMatrix in order to make it scalable for even bigger data.
We are planning to add another functionality to make the output easily readable for the user. After Graph summarization we want to create a graph using GraphX library that will show how the final summarized graph will look like.
We will also try to optimize our functions, to decrease the overall running time of the algorithm.

7. Setting up the environment

In order to run the program you need Spark and Scala installed on your machine, or you can simply install Scala IDE for eclipse.

Step 1:

Using a Maven template (forked from SANSA Template Maven spark https://github.com/SANSA-Stack/SANSA-Template-Maven-Spark.git) you can download and generate the project with all its dependencies.

Step 2:

Open Scala IDE and import this project as maven project. Go to:

- File > import
- Select Maven > Existing Maven Projects > Press Next
- Browse to the project directory where you cloned/downloaded the project which contains the .pom file and press Finish

Step 3:

- Configure Scala Compiler to use Scala 2.11 bundle (dynamic)

- Go to project's properties by right-click on project > properties
- At Scala Compiler check `Use Project Settings` and select latest 2.11 bundle (dynamic) for Scala installation
- Press `Apply and close` and run the main Class > src/main/scala/MainProgramC.scala

Note: In order to optimize the matrix operations we used the open-source library netlib-java which is a wrapper for low-level BLAS, LAPACK and ARPACK that performs as fast as the C / Fortran interfaces with a pure JVM fallback.

However the best choice is to use Intel MKL library. In that case here is how to setup MKL as the underlying native engine under Windows:

- Download and install MKL;
- Locate the folder containing the relevant (64-bit or 32-bit) DLLs - e.g. C:\Program Files (x86)\IntelSWTools\compilers_and_libraries\windows\redist\intel64_win\mkl - and add this to your PATH;
- Add -Dcom.github.fommil.netlib.NativeSystemBLAS.natives=mkl_rt.dll to the VM arguments at Run Configurations.

REFERENCES

[1] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, Craig A. Knoblock
Unsupervised Entity Resolution on Multi-type Graphs
http://usc-isi-i2.github.io/papers/zhu16-iswc.pdf

[2] Wikipedia,
Tf–idf
https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] Reza Bosagh Zadeh, Xiangrui Meng, Alexander Ulanov, Burak Yavuz, Li Pu, Shivaram Venkataraman, Evan Sparks, Aaron Staple, Aaron Staple,
Matrix Computations and Optimization in Apache Spark
https://stanford.edu/~rezab/papers/linalg.pdf

[4] Reza Bosagh Zadeh, Xiangrui Meng, Alexander Ulanov, Burak Yavuz, Li Pu, Shivaram Venkataraman, Evan Sparks, Aaron Staple, Aaron Staple,
Matrix Computations and Optimization in Apache Spark
https://stanford.edu/~rezab/papers/linalg.pdf

[5] GitHub,
Breeze Linear Algebra
https://github.com/scalanlp/breeze/wiki/Breeze-Linear-Algebra

[6] Intel, Developer Zone,
Intel Math Kernel Library
https://software.intel.com/en-us/mkl

[7] Lin Hong Seba,
Graph Summarization Algorithm
https://github.com/linhongseba/ClusterandSummaryKPartite/tree/master/KSummary/matlab/Demo_code/Product