Discovering Facts with Boolean Tensor Tucker Decomposition

Dóra Erdős*
Boston University
Boston, MA 02215, USA
edori@cs.bu.edu

Pauli Miettinen
Max-Planck-Institut für Informatik
Saarbrücken, Germany
pmiettin@mpi-inf.mpg.de

ABSTRACT

Open Information Extraction (Open IE) has gained increasing research interest in recent years. The first step in Open IE is to extract raw subject–predicate–object triples from the data. These raw triples are rarely usable per se, and need additional post-processing. To that end, we proposed the use of Boolean Tucker tensor decomposition to simultaneously find the entity and relation synonyms and the facts connecting them from the raw triples. Our method represents the synonym sets and facts using (sparse) binary matrices and tensor that can be efficiently stored and manipulated.

We consider the presentation of the problem as a Boolean tensor decomposition as one of this paper's main contributions. To study the validity of this approach, we use a recent algorithm for scalable Boolean Tucker decomposition. We validate the results with empirical evaluation on a new semi-synthetic data set, generated to faithfully reproduce real-world data features, as well as with real-world data from existing Open IE extractor. We show that our method obtains high precision while the low recall can easily be remedied by considering the original data together with the decomposition.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition*

Keywords

Open Information Extraction; Tensor decomposition; Boolean tensor decomposition; Entity disambiguation; Tucker3

1. INTRODUCTION

`Elizabeth Windsor`	`ruler of`	`England`
`Elizabeth`	`is queen`	`United Kingdom`
`Elizabeth II`	`is queen`	`England`
`Beckham`	`was born in`	`United Kingdom`
`David B.`	`plays for`	`LA Galaxy`
`D. Beckham`	`plays soccer for`	`England`
`Elizabeth Windsor`	`birth`	`England`
`Elizabeth (queen)`	`isRulerOf`	`United Kingdom`
`Beckham (soccer player)`	`isBornIn`	`United Kingdom`
`Beckham (soccer player)`	`playsFor`	`LA Galaxy`
`Beckham (soccer player)`	`playsFor`	`United Kingdom`
`Elizabeth (queen)`	`isBornIn`	`United Kingdom`

Table 1: Top: Surface term triples. Bottom: Facts.

A typical first task in Open Information Extraction (Open IE) is to extract *surface triples* of form *noun phrase–verbal phrase–noun phrase* (e.g. using TextRunner [1]). The next step, then, is to resolve the synonyms and find the facts present in the data. In this paper, we propose the use of *Boolean Tucker tensor decomposition* for this task. We should emphasize already that in this short paper we do not consider the semantics of the words: we do not try to map the entities into known entities, nor do we try to fit them into ontologies, for example. Rather, we are only interested on the combinatorial structure of the data, and what can be inferred from it; using the knowledge about the surface terms is left for future work.

To explain what is Boolean Tucker tensor decomposition, and how it can handle the task of finding the facts, let us start with an example data in Table 1. The top part of it shows seven surface triples that can be expressed using a 6-by-6-by-3 binary tensor (3-way array) where (i, j, k) element is 1 if, for example, i corresponds to `Elizabeth`, j corresponds to `is queen`, and k corresponds to `England`. Call this data tensor $\boldsymbol{\mathcal{X}}$. Our aim is to find which of the surface terms are likely to refer to the same entity or relation, and what are the underlying facts (entity–relation–entity triples). For this paper, an entity is just a set of surface terms referring to it, as is a relation—we do not aim at finding the semantic meaning. We also allow the surface terms to belong to many entities, naturally handling the polysemous surface terms without having to store each instance of each term separately (see [10]). The entities, relations, and facts of our example data are shown in the bottom half of Table 1.

As we expressed our original data using a binary tensor, we can also express the facts of the data using a smaller (2-by-3-by-2 in the example) binary tensor, call it $\boldsymbol{\mathcal{G}}$. The

*The work was done while D.E. was visiting MPI-INF. Part of her work was supported by NSF grants CNS:1017529 and III:1218437.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2507846.

entities and relations can be expressed using binary matrices, one for subject entities, one for relations, and one for object entities, called $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, respectively. Matrix $\boldsymbol{A}$, for example, has as many rows as our original data has surface forms for subject entities (6 in our example), and as many columns as we have subject entities (the dimensionality of the core tensor in the first mode; 2 in our example). A column of $\boldsymbol{A}$ defines the set of surface terms associated with this entity by setting the corresponding rows to 1. In our example, the column of $\boldsymbol{A}$ corresponding to `Elizabeth (queen)` would have 1 in rows corresponding to `Elizabeth Windsor`, `Elizabeth`, and `Elizabeth II`.

Our goal is to find these three matrices and the smaller tensor. But not just any three matrices and a tensor: we want them to agree with the knowledge we have in the surface triples. To this end, we need to explain how we represent the original data using the three matrices and the small tensor. We will call the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ *factor matrices*, the tensor $\boldsymbol{\mathcal{G}}$ *core tensor*, and the factor matrices and the core tensor together are the *decomposition*. So, assume we are given the decomposition corresponding to the bottom half of Table 1, and we want to know if `Elizabeth II is ruler of United Kingdom` is true. We can do this easily: from matrix $\boldsymbol{A}$ we know that `Elizabeth II` appears in column p (corresponding to the entity `Elizabeth (queen)`), and similarly, `ruler of` appears in column q of $\boldsymbol{B}$, and `United Kingdom` in column r of $\boldsymbol{C}$. Further, in the core $\boldsymbol{\mathcal{G}}$, we have $g_{pqr} = 1$, meaning that this fact is true. If some surface form appears in many columns of a factor matrix, we consider the fact to be true if *any* of its meanings yields a true fact. Hence, in order to compute if a surface triple corresponds to a fact, we find the index triple (i, j, k) corresponding to the surface terms, and compute $\bigvee_{p,q,r} g_{pqr} a_{ip} b_{jq} c_{kr}$. Conversely, we can define a good decomposition to be such that it agrees (approximately) with the given data. This leads us to the definition of *Boolean Tucker decomposition* [4]:

Definition 1.1 (Boolean Tucker decomposition). *Given an I-by-J-by-K binary tensor $\boldsymbol{\mathcal{X}} = (x_{ijk})$ and three integers P, Q, and R, the (P, Q, R)* Boolean Tucker decomposition *of $\boldsymbol{\mathcal{X}}$ is a tuple $(\boldsymbol{\mathcal{G}}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$, where $\boldsymbol{\mathcal{G}}$ is a P-by-Q-by-R binary* core tensor *and $\boldsymbol{A}$ (I-by-P), $\boldsymbol{B}$ (J-by-Q), and $\boldsymbol{C}$ (K-by-R) are binary* factor matrices *such that they minimize*

$$\sum_{ijk} \left| x_{ijk} - \bigvee_{p=1}^{P} \bigvee_{q=1}^{Q} \bigvee_{r=1}^{R} g_{pqr}\, a_{ip} b_{jq} c_{kr} \right| . \qquad (1)$$

For a schematic view of Tucker decomposition, see Figure 1. The Boolean Tucker decomposition is like the normal Tucker decomposition [3], except that all tensors and matrices are expected to be binary and the addition is replaced with the logical *or* ($1 + 1 = 1$).

2. THE ALGORITHM

Most existing algorithms for tensor Tucker decomposition (or *Tucker3 decomposition*, to be precise) work with real-valued data and typically require the factor matrices to be column orthogonal [3], yielding dense core tensor. Therefore, they are not suitable for our application (and typically cannot scale for big core tensors). Recently, [6, 7] proposed an algorithm called Rescal for Tucker2 decomposition of data similar to ours. The difference between Tucker3 and Tucker2 is the latter does not reduce dimensionality in one of the

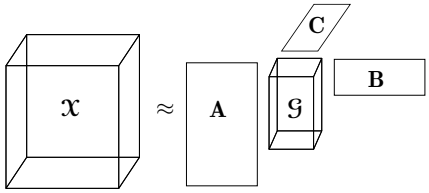

Figure 1: Tucker tensor decomposition.

modes; in case of Rescal, that mode is the relations mode (meaning that Rescal will not do relation disambiguation). In addition, Rescal requires the two factor matrices corresponding to entities to be the same, which is something we do not do in this work. Finally, Rescal will return real-valued factor matrix and core tensor, requiring further work to identify synonyms and their relations.

Boolean Tucker decompositions are studied much less. The only algorithm we are aware of, from [4], does not scale to the data sizes we need to handle. Therefore, in order to compute the Boolean Tucker decomposition, we will use a scalable algorithm we developed.[1] As the development of that algorithm is not the purpose of this paper, here we will only give a short overview how the algorithm works (see the technical report in footnote 1 for more information).

Our algorithm works in three parts. The first part tries to identify dense sub-tensors (called *blocks*) using random walk: we consider every subject–relation–object triple a node in the graph, and add an edge between two nodes if they differ in at most one location. The random walk part makes multiple sets of short walks in this graph, counting the frequency of visits in each node. Each walk within one set of walks starts from nodes already visited during previous walks in that set (except the first walk, which starts from a random node). After each set of random walks, the algorithm identifies the most frequently visited nodes, and considers the smallest sub-tensor that contains all frequent nodes as a potential block. This sub-tensor might also contain zeros, and therefore the algorithm checks if the density of the block is high enough before accepting it. All nodes visited by the set of random walks are removed from the graph, and the random walk phase ends when the graph becomes empty.

After the random walk phase, the block merging phase starts to go over the found blocks. As the random walks are short, they cannot find big blocks by themselves, instead returning multiple overlapping blocks. The task of this phase is to merge two partially overlapping blocks if the resulting block's density is still high enough. This block merging phase ends when no more merges can be done.

After the block merging phase, we have a special case of a decomposition, known as (Boolean) CP decomposition [3]: the core tensor is hyper-diagonal (i.e. $g_{ijk} = 1$ if and only if $i = j = k$), and each factor matrix has the same number of columns (the columns of the factor matrices define the blocks). But this means that the factor matrices can have multiple (almost) identical columns: if, for example, a set of synonyms corresponding to a subject has multiple relations to multiple objects, the subject's synonym factor will be re-

[1] A paper explaining this algorithm in detail can be found at `http://www.mpi-inf.mpg.de/~pmiettin/btf/`

peated for each of them. Therefore, the task of the last part is to identify these repeated instances and merge them. This task is made harder by the fact that factors corresponding to the same entity or relation are rarely exactly the same. Therefore, for this last part, we employ the Minimum Description Length (MDL) principle [8]: we merge two factors and the corresponding 1s in the core tensor (representing the facts) if doing so reduces the description length of our data. Hence, the MDL principle is also used to decide the final dimensions of the core tensor, P, Q, and R.

Let us briefly give some intuition behind using the MDL principle here (full details are found on the aforementioned technical report). We use the two-part MDL, where the description length of the data $L(D)$ is computed as $L(D) = L(\mathcal{M}) + L(D \mid \mathcal{M})$, where $\mathcal{M}$ is our model of the data and $L(D \mid \mathcal{M})$ is the description length of the data given the model. Our model is the Boolean Tucker decomposition of the data, while the second part involves an *error tensor*; a binary tensor of same size as the data that contains 1 when the decomposition disagrees with the data. When we merge two factors in some of the factor matrices, $L(\mathcal{M})$ is reduced as the factor matrix and the core are made smaller. On the other hand, the error part $L(D \mid \mathcal{M})$ is usually increased. Our aim, then, is to find merges that reduce $L(\mathcal{M})$ more than they increase $L(D \mid \mathcal{M})$.

3. EXPERIMENTAL EVALUATION

To test the applicability of Boolean Tucker decomposition for the task of fact finding we test our algorithm on both real-world and semi-synthetic data.

The semi-synthetic dataset.

In order to get a quantitative evaluation of our algorithm we generate a dataset that on one hand models the statistical properties of real word textual data. On the other hand, the correct decomposition of the data tensor is also known and can be used as ground truth in our evaluations. We call this data YPSS and is publicly available from our website[2].

Data source. The YPSS dataset is generated from a combination of data obtained from the Yago[3] [9] and Patty[4] [5] ontologies. Yago is a semantic knowledge base and Patty is a collection of semantically-typed relational patterns.

Data generation. The data generation process goes over the facts (entity–relation–entity triples) in the Patty data. For each fact, Patty also reports the frequency of the fact. For each fact, we generate as many surface triples as is the frequency of the fact. To generate the surface forms, we replace the entities and the relation in the fact with randomly selected surface form. This surface form is selected with respect to probability $\Pr(s \mid c)$, where s is the surface form and c is the clean entity or relation in the fact.

For relations, the probabilities $\Pr(s \mid c)$ can be obtained directly from Patty by using the (normalized) support co-occurrence. For entities, Yago unfortunately does not have this information. What Yago has, however, is (an approximation of) $\Pr(c \mid s)$. In addition, we can obtain $\Pr(c)$ for entities c by computing their relative frequency (with multiplicity) in the Patty data, and $\Pr(s)$ for surface terms s by $\Pr(s) = \sum_c \Pr(s \mid c)$. With these, from Bayes' theorem we get $\Pr(s \mid c) = \Pr(c \mid s)\Pr(c)/\Pr(s)$.

Because we generate multiple synonym triplets for every fact in Patty, it may happen that identical surface triplets are generated along the way. Since this phenomenon of multiple appearance is also natural in real data, we report for every surface term triplet its multiplicity.

The final YPSS surface tensor (not counting multiplicities) is of size 222k-by-23.6k-by-225k and contains 63 million surface term triplets.

Ground truth dataset. In the data generation process described above we replace every clean term in the factual data with a number of synonyms of that word. We keep track of the set of synonyms generated for each term and call it the ground truth factor of that clean term. At the end of our generation process we obtain 112k, 25 and 73k ground truth factors corresponding to subject entities, relations, and object entities, respectively.

ReVerb *dataset.*

Besides the YPSS data, we use a second, real-life dataset, in our experiments. This dataset, ReVerb, is the result of a binary assertion extraction of the form `(arg1, rel, arg2)` from the Web, provided as part of the Open Information Extraction project [2][5]. The size of the surface data tensor is 1.4M-by-665k-by-1.6M and it has 14.7 million ones.

3.1 Results

Quantitative results. To assess the quality of the Boolean Tucker decomposition we first report some results on the YPSS dataset. The results we report here are on a sample of the YPSS dataset of size 39.5k-by-8k-by-21k containing 804k surface term triplets.

The tensor that is defined by the Tucker decomposition obtained by our algorithm is sparser tha the original; it only has 266k ones. The YPSS data is generated so that we model missing entries – the term triplets are generated at random and only very few of the possible triplets are in the final dataset. We aim to run our algorithm so that we also finde these "missing" values, hence we force our algorithm to accept every factor that covers as few as 10% of ones in the tensor. The result is 247k false positive hits, thus cells that are predicted one by our algorithm, but the corresponding triplets are not present in our data. We also achieve 20k true positives, which is consistent with our allowed error rate.

But to properly assess our method, we need to study how well the factor matrices correspond to the ground truth, that is, how well we reconstruct the latent structure. Using the ground truth of the YPSS data we can compute the precision and recall of the factor matrices (i.e. synonym sets). Let f_{MDL} be a factor obtained after the MDL phase of or algorithm. Then in our evaluation we first find the ground truth factor which has the highest overlap with f_{MDL}, let this be f_{truth}. The number of true positive terms in f_{MDL} is computed as $|f_{\text{MDL}} \cap f_{\text{truth}}|$ while the number of false positive terms is $|f_{\text{MDL}} \setminus f_{\text{truth}}|$. We obtain an average precision over all found factors of 0.456 for the subject entities, 0.900 for the object entities, and 0.800 for relations. The recall is 0.922, 0.475, and 0.01 for subjects, objects, and relations, respectively. The purity (homogeneity) is 0.46 for subject entities, 0.93 for object entities, and 0.98 for relations while the inverse purity

[2] `http://www.mpi-inf.mpg.de/~pmiettin/btf/`
[3] `http://www.mpi-inf.mpg.de/yago-naga/yago/`
[4] `http://www.mpi-inf.mpg.de/yago-naga/patty/`
[5] `http://reverb.cs.washington.edu/`

(specificity) is 0.79, 0.55, and 0.03 again for subjects, objects, and relations, respectively. The extremely low recall and inverse purity for relations is due to most surface relations appearing only very few times making them "noise" in the MDL sense. We discuss more on this long-tail behaviour below. Why the precision is high but recall low for object entities, but vice versa for subject entities, we do not know.

Qualitative results. With the ReVerb data, many factors are intuitive collections of synonyms, for example, {`entrepreneur, vendor programmer, industry analyst`} as objects or {`usually end with`, `be usually precede by`, `usually terminate in`, `usually culminate in`} as relations. Also {`poor judgment`, `leadership and vision`, `true leadership`} as subjects form a reasonable set, even if the terms are not synonyms in a strict sense.

On examining our results on YPSS and ReVerb data sets, we found that many surface terms do not belong to any entity or relation. This is to be expected: most surface terms appear only very few times in the data, and this "long tail" is easier to explain as single elements than using factors. In other words, the MDL-optimal decomposition will put these rare surface terms into the error tensors instead of covering them. We do not consider this a problem of our method *per se*, as we can always keep the error tensor available and use it to get information about the long tail.

While the decomposition did not contain all surface terms, for those that it contained, we did find nice sets of synonyms. Using these synonyms, we can correctly answer questions even when the answer is not in the original data. Consider, for example, the simple SPARQL ASK query `clean water`, `be essential to`, `health`. It would return `false` in the original data, as we do not have that triple in our raw data. Yet, doing the same query in the decomposition (following the procedure explained in Section 1) would return `true`, as our core tensor does have a fact relating the entities and relation to which these surface terms are associated.

The decomposition can also be used on answering SELECT queries by (implicitly) considering the tensor that would result should the core and factor matrices be multiplied together. A query with one free variable results in a (row, column, or tube) fiber in this tensor, and the 1s in this fiber correspond to the surface terms in the answer set. If the query has two free variables, the result is a matrix (slice of the tensor), again with locations of 1s explaining defining the answer pairs.

Examining the results for the SELECT queries shows them to behave similarly to the ASK queries: The long tail is not covered by the decomposition, but the decomposition was able to return results that as themselves were not in the data (though their synonyms were). Again, for optimal results, we can always combine the results from the data with the results from the decomposition for increased recall. And as shown above, the precision will still stay high.

4. DISCUSSION

In this paper we show how to model the fact discovery in open information extraction as a Boolean tensor decomposition problem and give an overview of an algorithm to find this decomposition. We believe that this formulation will prove useful for other researchers wanting to get a new angle to the problem.

There are many interesting future research directions we have not studied in this work. One of them is to use the textual similarity of surface terms. A simple idea here is to use the textual similarity as a regressor, benefiting the factors that contain similar surface terms. Similarly approach should work with other kind of auxiliary information, too.

Another idea is to restrict the entity factor matrices to be equivalent. Currently, we distinguish between subject and object entities, and when we decide, say, that two surface subjects are synonyms, we do not distribute that information over to the object entities. Allowing this information flow could potentially speed up the algorithm considerably, and it should alleviate the extreme sparsity of the data.

That extreme sparsity of the data is a clear problem for our approach. We can control the minimum density of the blocks we find, but very sparse blocks are better represented, from MDL's point of view, by their elements, and thus the MDL phase tends to reduce the sparsest blocks even when a human would judge them perfectly satisfiable. The underlying problem is that most 0s in our data do not mean that the corresponding triple does not exist; they mean that the extractor did not see that triple (this problem, of course, is not specific to our algorithm). The methods we mentioned above could help with this problem. Another approach could be to have different types of 0s: 0s that are simply unknown, and 0s that correspond to triples that do not exist.

In this paper we have considered the Boolean tensor decompositions. We believe that our reasons for this restriction are valid, especially what comes to the core tensor. But it could be argued that the factor matrices do not need to be binary; they could rather be non-negative, containing some kind of ranking information about how well the surface term fits into the entity or relation in question.

5. REFERENCES

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI '07*, pages 2670–2676, 2007.

[2] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP '11*, 2011.

[3] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[4] P. Miettinen. Boolean tensor factorizations. In *ICDM '11*, pages 447–456, 2011.

[5] N. Nakashole, G. Weikum, and F. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP '12*, pages 1135–1145, 2012.

[6] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML '11*, pages 809–816, 2011.

[7] M. Nickel, V. Tresp, and H.-P. Kriegel. Factorizing YAGO: Scalable machine learning for linked data. In *WWW '12*, pages 271–280, 2012.

[8] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, Sept. 1978.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge. In *WWW '07*, pages 697–706, 2007.

[10] A. Yates and O. Etzioni. Unsupervised methods for determining object and relation synonyms on the web. *J. Artif. Intell. Res.*, 34:255–296, 2009.