Inducing a Decision Tree with Discriminative Paths to Classify Entities in a Knowledge Graph

Gilles Vandewiele, Bram Steenwinckel, Femke Ongenae, and Filip De Turck

IDLab, Ghent University – imec, Technologiepark-Zwijnaarde 126, Ghent, Belgium
{firstname}.{lastname}@ugent.be

Abstract. Deep-learning based techniques are increasingly being used for different machine learning tasks on knowledge graphs. While it has been shown empirically that these techniques often achieve better predictive performances than their classical counterparts, where features are extracted from the graph, they lack interpretability. Interpretability is a vital aspect in critical domains such as the health and financial sector. In this paper, we present a technique that builds a decision tree of class-specific substructures in order to classify different entities within the knowledge graph. We show how our proposed technique is competitive to current state-of-the-art deep-learning techniques on four benchmark datasets, while being fully interpretable.

Keywords: Knowledge Graphs · White-Box Machine Learning · Semantic Data Mining

1 Introduction

Graphs are data structures that are useful to represent ubiquitous phenomena such as social networks, biological protein reactions and recommendation systems. One of their strengths lies in the fact that they add an extra dimension to the data by explicitly modeling interactions, through the form of edges, between individual units (i.e. nodes) [6]. These graphs can also be used to represent knowledge bases, which are repositories of domain or expert knowledge. Today, these graphs are increasingly being leveraged for various machine learning tasks [23]. One of these tasks is to classify nodes into one out of a set of discrete classes, which is the focus of this study.

Different types of approaches can be identified in order to classify nodes in a knowledge graph. A first group of approaches are classical ones. Here, information about the structure of the graph is explicitly encoded into a feature vector, which can then be fed to a machine learning model [9]. Examples of such features are indications of the presence of specific local neighbourhood structures [10] and graph statistics [1]. The disadvantage of this type of approach is that it is not agnostic: they need to be tailored specifically for the task at hand and application domain at hand. Another popular classical approach, which is more task-agnostic, is applying kernel methods [20], which measure similarity between two knowledge bases, either directly on their graph representation [11,21,22] or based on description logics [4]. Unfortunately, using pairwise similarity measures as features is often a lot less interpretable than using human-understandable variables. A second type of approach, which have been gaining immensely in popularity, is representation learning. The goal is to create a mapping from the graph-based structures onto low-dimensional numerical vectors that can be used for downstream machine learning tasks [5]. These vectors can be created through tensor factorization [13], or by applying unsupervised deep-learning techniques, such as Word2Vec [12] on walks extracted from the graph [14,2,16]. Representation learning can be seen as completely task-agnostic, it is even the case that representations can be re-used for multiple task. Moreover, these techniques often tend to achieve higher performances than, for example, their kernel or classical feature-based counterparts. The disadvantage of these approaches is that by mapping an entity into a low-dimensional latent representation, all interpretability is lost. A final and very recent approach are graph networks, which are adaptations of neural networks that can directly work on graph-based data [3,8,18]. Again, this technique can be seen as a black box, making it very hard or even impossible to extract any insights from the model.

In this paper, we present a technique that can classify unseen entities or nodes from the KG, given some already labeled entities. It does this by building a decision tree consisting of useful substructures, extracted from the neighborhood of the labeled entities, which are very discriminative for a certain class. The technique is domain-agnostic, while resulting in a white-box model that uses interpretable features to classify new unseen nodes.

2 Methodology

In this section, we first explain some fundamental concepts, followed by an elaboration of the different steps of our algorithm.

2.1 Entity Classification: Problem Definition

Given a multi-relational directed knowledge graph $\mathbb{G} = (\mathbb{V},\ \mathbb{E},\ \ell)$, constructed from a knowledge base of triples, where $\mathbb{V}$ are the vertices or entities in our graph, $\mathbb{E}$ the edges or predicates and ℓ a labeling function that maps each vertex or edge on its corresponding label. Moreover, we are provided with a list of entities $\boldsymbol{V}$ with a corresponding vector of discrete labels $\boldsymbol{y}$. Our goal is to construct a model or hypothesis $h(.)$ based on $\boldsymbol{V}$ and $\boldsymbol{y}$ that minimizes a loss function $\mathcal{L}(.)$, and which generalizes well to unseen vertices:

$$\arg\min_{h} \mathcal{L}(\boldsymbol{y}, h(\boldsymbol{V}))$$

2.2 Neighborhoods, Walks and Wildcards

We characterize each instance $v \in \boldsymbol{V}$, by its neighborhood $\boldsymbol{\mathcal{N}}(v)$ of a certain depth $\boldsymbol{d}$. The neighborhood of v is a graph that contains all vertices that can be reached with $\boldsymbol{d}$ or less steps from the instance v. It can easily be extracted by performing a breadth-first-traversal. As done by de Vries et al. [22], we first transform our graph to remove its multi-relational aspect. To do this, each `(subject, predicate, object)` triple from the original knowledge base can be represented by three labeled nodes and 2 unlabeled edges (`subject` $\rightarrow$ `predicate` and `predicate` $\rightarrow$ `object`). This transformation reduces the complexity of the further elaborated procedures, without a loss of correctness, since a distinction between entities and predicates is no longer needed.

We define a walk as a sequence of vertices. Due to our previously discussed transformation, this sequence will consist of consecutive vertices and edges from the original graph. The first vertex within this walk is often called the root of the walk, which serves as a placeholder that is replaced by a specific vertex depending on the context. We notate a walk as $\texttt{root} \rightarrow \texttt{e}_0 \rightarrow \texttt{v}_1 \rightarrow \texttt{e}_1 \rightarrow \ldots$.

We introduce a new special type of hop for our walks, which we call a 'wildcard' and notate by an asterisk $*$. The semantics of this wildcard is that any edge or vertex label can be matched on that position in the sequence. This enables the walks to have more expressive power. To illustrate this, imagine that the presence of an entity of a specific type $\mathcal{T}$ is very discriminative for a certain class. It is possible that only the fact that this entity is of that type carries information, while the specifics of the entity itself are unimportant. As such, this could be represented by a walk $\texttt{root} \rightarrow * \rightarrow * \rightarrow \texttt{rdf} : \texttt{type} \rightarrow \mathcal{T}$.

2.3 Inducing a decision tree of discriminative walks

In this study, we will focus on a special type of walk. A walk of length l has a root placeholder, followed by $l-2$ wildcards and ending in a specific vertex v, i.e. $\texttt{root} \rightarrow * \rightarrow ... \rightarrow * \rightarrow \texttt{v}$. As mentioned, the first hop, `root`, is replaced by v whenever we want to search for it in its neighborhood $\boldsymbol{\mathcal{N}}(v)$. Alternatively, we can simply represent these types of walks by a tuple: $w = (v,\ l)$. These type of walks allow to be searched in a neighborhood in constant time, while already possessing a rich amount of expressive power, as we will demonstrate empirically further. When extracting a neighborhood of depth $\boldsymbol{d}$, we keep track of $\boldsymbol{d}$ different sets $\{\boldsymbol{\mathcal{N}}_i(v) \mid 1 \leq i \leq \boldsymbol{d}\}$, where $\boldsymbol{\mathcal{N}}_i(v)$ stores the nodes that can be reached in exactly i hops. Whenever we want to search for a certain walk $w = (v,\ l)$ of that type in a neighborhood, we only need to check whether v appears in $\boldsymbol{\mathcal{N}}_l(v)$, thus avoiding the need to traverse parts of the graph. This allows us to quickly perform a brute-force search on all combinations of vertices v and lengths, up until a specified maximum.

Our goal is to mine a walk $w = (v,\ l)$ that maximizes information gain, which is defined as the (weighted) reduction in entropy obtained by partitioning the data. To calculate this, we partition our set of nodes into two mutually exclusive sets: a set of nodes for which the walk can be found and a set of walks for which the walk cannot be found. Afterwards, we can measure the entropy of the corresponding labels in these two sets in order to calculate the difference.

Often, one walk is not enough to create a perfect separation between the different classes in the feature space, especially when dealing with a multi-class problem. Therefore, we recursively build a decision tree by partitioning our data after mining the most discriminative walk into a set of nodes for which the walk can be found and a set for which the walk cannot be found. While decision trees possess excellent interpretability characteristics, they can be prone to overfitting [17]. Therefore, two hyper-parameters that allow for pre-pruning, which are conditions on which the algorithm halts, are introduced. On the one hand, the algorithm halts when a certain depth (`max_depth`) is reached. On the other hand, the algorithm stops when the amount of samples in a particular node of the decision tree is lower than a specified amount (`min_samples_leaf`).

3 Results

In this section, we will demonstrate the predictive power of our proposed approach by comparing its accuracy on benchmark datasets with accuracy scores for two recent state-of-the-art techniques: RDF2VEC [14] and Relational Graph Convolutional Networks (R-GCN). We extracted four datasets, from varying domains, from a public repository set up by Ristoski et al. [15]. For each of the benchmark datasets, we use the same train-test partitioning as provided by the original repository. For all classifiers, no pre-pruning was applied and trees were thus grown until the training set was perfectly classified. The neighborhood depth, and the maximum depth of the extracted walks was equal to 8, as was done in the study of Ristoski et al [14]. For each dataset, we performed 5 runs. The average accuracy scores achieved on the test set and their corresponding standard deviations are summarized in Table 1. The results for R-GCN and RDF2VEC are taken directly from Schlichtkrull et al. [19]. As can be seen, the performances of all three techniques are competitive to each other. Only on the *AM* dataset, our technique performs slightly worse than both others, while on the *MUTAG* dataset, our technique outperforms the two others, albeit with a higher variance.

4 Conclusion and Future Work

In this study, we proposed to build a decision tree in a recursive fashion. The advantage of this approach is that it is fully interpretable and the model, or at least the path to a certain prediction, can easily be visualised to a domain expert. While our implementation is a first proof-of-concept, results are already very promising as the technique is competitive to current state-of-the-art on four

Dataset	R-GCN	RDF2VEC	Walk Tree
AIFB	**95.83 ± 0.62**	88.88 ± 0.00	89.44 ± 2.08
BGS	83.10 ± 0.80	**87.24 ± 0.89**	86.90 ± 1.38
MUTAG	73.23 ± 0.48	67.20 ± 1.24	**73.82 ± 5.61**
AM	**89.29 ± 0.35**	88.33 ± 0.61	86.77 ± 0.59

Table 1. The accuracy scores of Relational Graph Convolutional Networks (R-GCN), RDF2VEC and our proposed approach on four benchmark datasets.

tested benchmark datasets. It even outperforms current techniques on one of the four datasets. Nevertheless, potential improvements can still be made. Examples of possible extensions are a post-pruning phase and experimenting with other splitting criteria than information gain. Moreover, ensemble techniques can be used to construct a collection of different decision trees in order to reduce the model variance. One straight-forward ensembling technique is bagging, where we induce multiple decision tree on subsets of training instances, which are sampled from the original training set through bootstrapping. One other alternative, that has proven great success in the domain of timeseries classification [7], is to step away from mining these discriminative substructures one by one in a recursive fashion, but instead mine a large collection of substructures and then create feature vectors based on the presence or absence of these substructures. Finally, it should be noted that the concept of a walk is not a necessity in the proposed approach, as only the last hop on that walk determines whether or not it can be found in a neighborhood. Nevertheless, using a walk as representation is more intuitive and easily allows for further extension upon this data structure, such as heuristically filling in wildcards on the walk to try and improve the information gain or plugging in more complex data structures than single vertices (e.g. trees and subgraphs).

Code availability An implementation of the algorithm is available on Github[1].

Acknowledgements Gilles Vandewiele (1S31417N) and Bram Steenwinckel (1SA0219N) are both funded by a strategic base research grant of Fonds Wetenschappelijk Onderzoek (FWO). Femke Ongenae is funded by a Bijzonder OnderzoeksFonds (BOF) grant of Ghent University.

References

1. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. Social Network Data Analytics (2011)
2. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global rdf vector space embeddings. In: International Semantic Web Conference. pp. 190–207. Springer (2017)
3. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems. pp. 3844–3852 (2016)

[1] `https://github.com/IBCNServices/KGPTree`

4. Fanizzi, N., dAmato, C., Esposito, F.: Statistical learning for inductive query answering on owl ontologies. In: International Semantic Web Conference. pp. 195–212. Springer (2008)
5. Goyal, P., Ferrara, E.: Graph Embedding Techniques, Applications, and Performance: A Survey. Knowledge-Based Systems (2017)
6. Hamilton, W.L., Ying, R., Leskovec, J.: Representation Learning on Graphs: Methods and Applications. Preprint of article to appear in the IEEE Data Engineering Bulletin (2017)
7. Hills, J., Lines, J., Baranauskas, E., Mapp, J., Bagnall, A.: Classification of time series by shapelet transformation. Data Mining and Knowledge Discovery **28**(4), 851–881 (2014)
8. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
9. Latouche, P., Rossi, F.: Graphs in machine learning: an introduction. ESANN pp. 207–218 (apr 2015)
10. Liben-Nowell, D., Kleinberg, J.: The Link Prediction Problem for Social Networks. Proceedings of the Twelfth Annual ACM International Conference on Information and Knowledge Management (CIKM) (November 2003), 556–559 (2003)
11. Lösch, U., Bloehdorn, S., Rettinger, A.: Graph kernels for rdf data. In: Extended Semantic Web Conference. pp. 134–148. Springer (2012)
12. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
13. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE **104**(1), 11–33 (2015)
14. Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
15. Ristoski, P., de Vries, G.K.D., Paulheim, H.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: International Semantic Web Conference. pp. 186–194. Springer (2016)
16. Saeed, M.R., Prasanna, V.K.: Extracting entity-specific substructures for rdf graph embedding. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). pp. 378–385. IEEE (2018)
17. Schaffer, C.: When does overfitting decrease prediction accuracy in induced decision trees and rule sets? In: European Working Session on Learning. pp. 192–205. Springer (1991)
18. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling Relational Data with Graph Convolutional Networks. ESWC (2017)
19. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European Semantic Web Conference. pp. 593–607. Springer (2018)
20. Vishwanathan, S., Schraudolph, N., Kondor, R., Borgwardt, K.: Graph Kernels. Journal of Machine Learning Research **11**(Apr), 1201–1242 (2010)
21. de Vries, G.K.: A fast approximation of the weisfeiler-lehman graph kernel for rdf data. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 606–621. Springer (2013)
22. de Vries, G.K.D., de Rooij, S.: Substructure counting graph kernels for machine learning from rdf data. Web Semantics: Science, Services and Agents on the World Wide Web **35**, 71–84 (2015)
23. Wilcke, X., Bloem, P., De Boer, V.: The Knowledge Graph as the Default Data Model for Machine Learning. Data Science **1**, 1–0 (2017)