# Node Classification in Multirelational Information Networks

Rick Barber, Mirela Spasova

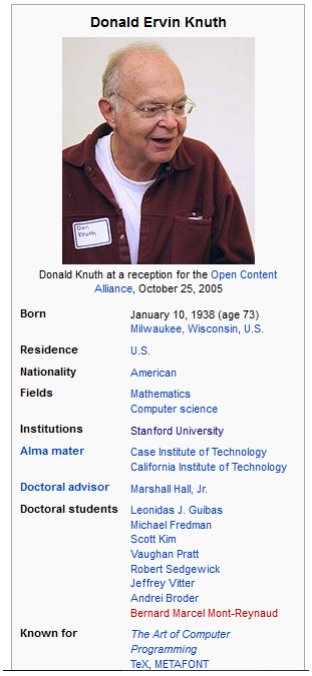
December 15, 2011

## 1 Problem Background

Our goal will be to predict the types of entities in a large multirelational information network. In particular, if we suppose we are given a graph $G = (V, E)$, a hierarchy of types $T$, and an *observed labeling function* $\hat{l}$ which for each node gives a path beginning at the root of $T$, called the *type(s)* of the node. Our goal will be to learn $l$ the *true labeling function* of the nodes

## 2 Data

Wikipedia contains a large amount of unstructured information. Infoboxes, however, are a mechanism used by the site to package many prominent features of a given Wikipedia entity into a structured summary–see below.



**Donald Ervin Knuth**

Donald Knuth at a reception for the Open Content Alliance, October 25, 2005

| | |
|---|---|
| **Born** | January 10, 1938 (age 73)<br>Milwaukee, Wisconsin, U.S. |
| **Residence** | U.S. |
| **Nationality** | American |
| **Fields** | Mathematics<br>Computer science |
| **Institutions** | Stanford University |
| **Alma mater** | Case Institute of Technology<br>California Institute of Technology |
| **Doctoral advisor** | Marshall Hall, Jr. |
| **Doctoral students** | Leonidas J. Guibas<br>Michael Fredman<br>Scott Kim<br>Vaughan Pratt<br>Robert Sedgewick<br>Jeffrey Vitter<br>Andrei Broder<br>Bernard Marcel Mont-Reynaud |
| **Known for** | *The Art of Computer Programming*<br>TeX, METAFONT |

The DBpedia dataset is a post-processed version of Wikipedia's infoboxes, and it will be the data set in our study. DBpedia contains more than 3.6 million entities among which there are almost 8.5 millions directed relations.

DBpedia includes a type hierarchy of 237 types and gives a set of type labels for most nodes in the network. However, a cursory look at the dataset will reveal that the set of types if often far underspecified (Obama is labeled as a Thing, Person but not a Politician or President for instance) and DBpedia's own website claims that they have over a million entities which aren't classified beyond "Thing" which is the root level of the ontology.

# 3 Method

## 3.1 Data generation and model

Some of the parsing machinery was written in joint work with Klemen Simonic and Rick Barber advised by Jure Leskovec in August of this year.

DBpedia's data is available in raw form in three files: the ontology file, the properties file, and the types file.

The ontology file is an XML file describing the hierarchy of DBpedia types. The root node is

Thing since all entities in the dataset are assumed to be Things. We parse the XML, and store the data as a tree structure on which a number of queries can be efficiently performed.

The properties file is a list of relations represented as triples $(s, p, o)$ where $s$ is an infobox entity that we call the subject, $p$ is the name of a relation we call a predicate, and $o$ is either a literal value or another entity which we call the object of the relation.

An example is given below

```
<http://dbpedia.org/resource/Aristotle> <http://dbpedia.org/ontology/influenced>
   <http://dbpedia.org/resource/Ptolemy> .
```

We represent this data as an actual graph. For example, with the above line we will ensure that there is a node corresponding to Aristotle and Ptolemy in our graph and we will add an edge with label "influenced" between these two nodes.

The types file is a list of triples $(e, n, t)$ where $e$ is the name of an entity and $t$ is a type of the entity and $n$ is a constant we ignore.

An example is given below

```
<http://dbpedia.org/resource/Autism>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://dbpedia.org/ontology/Disease> .
<http://dbpedia.org/resource/Autism>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://www.w3.org/2002/07/owl#Thing> .
```

This says that the DBpedia labeling function assigns labels Disease and Thing to the entity Autism. Of course, we can't be certain that the true, onobserved labeling function labels Autism in exactly the same way, but we will try to use the signal of the observed labeling function to learn the true labeling function, which we will detail later. We parse this file and store the type information in dictionaries to query later.

## 3.2 Classifiers and training/test data

For our first task, we have trained binary one vs all classifiers for various entity types and for various subsets of the data.

We use logistic regression throughout (the L_2 regularized version for liblinear, to be exact) in order to easily intepret the results of our prediction probabilistically, which will be important in the final classifier.

For instance, DBpedia has a Comedian type which corresponds to entities in the dataset who are comedians; this will be our running example.

#### 3.2.1 Protocol 1: uniform sampling of negative examples

Here we randomly sample $m + p$ comedians and $m + p$ non comedians from anywhere in the type hierarchy and retain the labels of $m$ for training purposes while hiding the labels for $p$ for testing purposes.

We used this protocol as a proof of concept to get us started and it served to show that we could differentiate types on DBpedia.

#### 3.2.2 Protocol 2: sampling from neighbor types

Here we sample positive examples as before, but for negative examples, we from types which are sibling types of Comedian in the hierarchy. We follow this procedure because our final, holistic classifier will need to be able to differentiate between neighbor types.

#### 3.2.3 The holistic classifier

We trained a classifier for each of the 237 types following protocol 2 above.

Our procedure for making a prediction on a new input $x$ is as follows:

```
1  predict(x, type tree)
2    lastPredict = Thing
3    candidates = Thing.children
4    type.add(lastPredict)
5    while(lastPredict not null and not leaf)
6      lastPredict = \arg\max_{c \in candidates} g(\theta_c^T x)
7    if g(\theta_{lastPredict}^T x) < .5
8      lastPredict = null
9    else
10     type.add(lastPredict)
11     candidates = lastPredict.children
12
13   return type
```

What this says is that we let all of the classifiers in the subtree of the last predicted type vote on whether they think $x$ is of their type or not, and we take the most confident yes vote to make a *subprediction* and proceed all over again with its children as candidates.

The returned type variable will indeed be a path to the root in the type tree, which is to say it follows our definition of a labeling function.

### 3.3 Features

Our feature set consists of features of three distinct types. We have a set of features called the node local features which correspond to network features like in degree, out degree, total degree, and clustering coefficient. We have another set of features called the edge type features which are a vector of counts of the types of edges leaving and entering the node in question. Referring to our earlier data triplet, Aristotle will have at least one edge of type "influenced" leaving his node while Ptolemy will have at least one edge of type "influenced" entering his node. Finally, we will use the count of types of adjacent entities in the graph as a feature. For instance, from the triplet above we know Aristotle has at least one neighbor of type Mathematician, Person, and Thing namely Ptolemy.

## 4 Results

The results of attempting to fully label 1000 randomly drawn DBpedia entities is below.

| | Precision | Recall | Accuracy |
|---|---|---|---|
| Depth 1 | 0.83 | 1.0 | 0.83 |
| Depth 2 | 0.77 | 0.85 | 0.69 |
| Depth 3 | 0.47 | 0.53 | 0.33 |
| Depth 4 | 0.71 | 1.0 | 0.71 |

We seperate our results according to the depth in the type tree at which the subprediction was made.

At a given sublevel suppose the true label is $l$. If our subprediction at this level is $l$ then we record a true positive for class $l$. If our subprediction at this level is $s \neq l$ then we record a false negative for class $l$ and a false positive for $s$. If our classifier prematurely stopped making predictions, we count the subsequent true types as false negatives, and if we make too specific a prediction these are counted as false positives for the labels actually predicted.

But since our DBpedia data is known to be incompletely specified in terms of types our results should be read as having better precision than is indicated because we can't actually count the DBpedia data as being the full truth. Indeed, as we noted before, the Barack Obama entity is specified to have type Thing and Person in the DBpedia network, but our classifier predicts Obama as being a Thing, Person, Politician, President. By our evaluation standards this prediction would reduce the depth 3 and 4 precision even though we see that our prediction is actually more credible

than the "correct" prediction according to the data.

We output these spurious predictions to a file, and many but not all of them were of this nature.

# 5 Summary

Even though our training data was not of perfect quality, we were able to perform reasonably well for a 237 class classification problem. Given that DBpedia's types were originally annotated fully by crowd sourcing, our results could represent significant time savings over a purely manual effort. Furthermore, our results were obtained with a static dataset of a predetermined quality. We belive that if instead of tasking DBpedia's contributors with randomly assigning types to the entities in their network, they answered yes or no classification questions corresponding to a learning algorithm's predictions, the training data and classifier quality would iteratively improve to nearly converging with what should be the true labels of the data.