Helping Novices Avoid the Hazards of Data: Leveraging Ontologies to Improve Model Generalization Automatically with Online Data Sources

Sasin Janpuangtong and **Dylan A. Shell**
Department of Computer Science and Engineering
Texas A&M University
College Station, TX 77843
sasin324@tamu.edu and dshell@cse.tamu.edu

Abstract

The infrastructure and tools necessary for large-scale data analytics, formerly the exclusive purview of experts, are increasingly available. Whereas a knowledgeable data-miner or domain expert can rightly be expected to exercise caution when required (e.g., around fallacious conclusions supposedly supported by the data), the non-expert may benefit from some judicious assistance. This paper describes an end-to-end learning framework that allows a novice to create models from data easily by helping structure the model building process and capturing extended aspects of domain knowledge. By treating the whole modeling process interactively and exploiting high-level knowledge in the form of an ontology, the framework is able to aid the user in a number of ways, including in helping to avoid pitfalls such as data dredging. Prudence must be exercised to avoid these hazards as certain conclusions may only be supported if, for example, there is extra knowledge which gives reason to trust a narrower set of hypotheses. This paper adopts the solution of using higher-level knowledge to allow this sort of domain knowledge to be used automatically, selecting relevant input attributes, and thence constraining the hypothesis space. We describe how the framework automatically exploits structured knowledge in an ontology to identify relevant concepts, and how a data extraction component can make use of online data sources to find measurements of those concepts so that their relevance can be evaluated. To validate our approach, models of four different problem domains were built using our implementation of the framework. Prediction error on unseen examples of these models show that our framework, making use of the ontology, helps to improve model generalization.

Introduction

A variety of well-established supervised learning methods produce a model from a set of examples. Despite the maturity of these algorithms, decisions that result from models are unlikely to be correct if data have been used indiscriminately. This is part of the so-called data dredging problem [Smith and Shah, 2002]. Figure 1 shows a bemusing example: US spending per annum on science, space, and technology is highly correlated with suicides by hanging, strangulation, and suffocation. Logically, we know that this correlation does not imply causation (*i.e.*, higher spending on technology cannot cause more suicides by hanging or vice versa). Unfortunately, without the capacity to distinguish real and spurious correlations, learning methods are prone to picking up such correlations in producing a model [Tukey, 1977]. The onus to be judicious ultimately falls on the person building the model. Otherwise, data dredging may lead to specious models, which may overfit, generalize poorly, and give fallacious conclusions.

The increased availability of data and the existence of easy-to-use procedures for regression and classification in commodity software allows inexperienced users to search for correlations amongst a large set of variables with scant regard for their meaning. Indeed, data dredging has been democratized and anyone may use seemingly sophisticated tools to arrive at unsound conclusions. This motivates the present work in developing a software framework to treat the whole modeling process rather than merely the model fitting stage. This framework should allow a non-expert user to (i) easily or even automatically create a model from existing data, (ii) avoid the pitfalls of data dredging, and (iii) build an accurate model, which can correctly predict unseen data.

Our work is inspired by the modeling process employed by Nelson and Sprecher [2008] to build a model of nuclear power use in a given country to help understand civil nu-

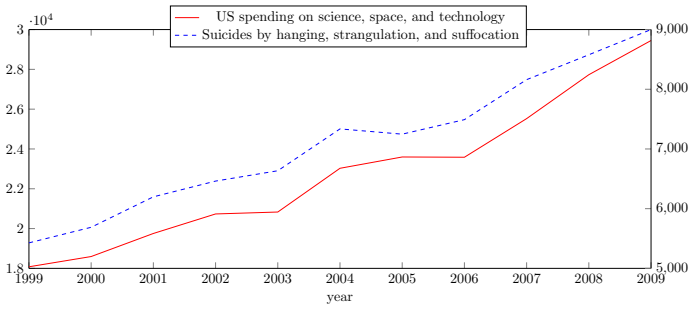

Figure 1: A plot showing correlation between data of 2 attributes: US spending on science, and suicides by hanging. Data of these attributes are highly correlated with correlation = 0.992 (Source: `http://www.tylervigen.com`).

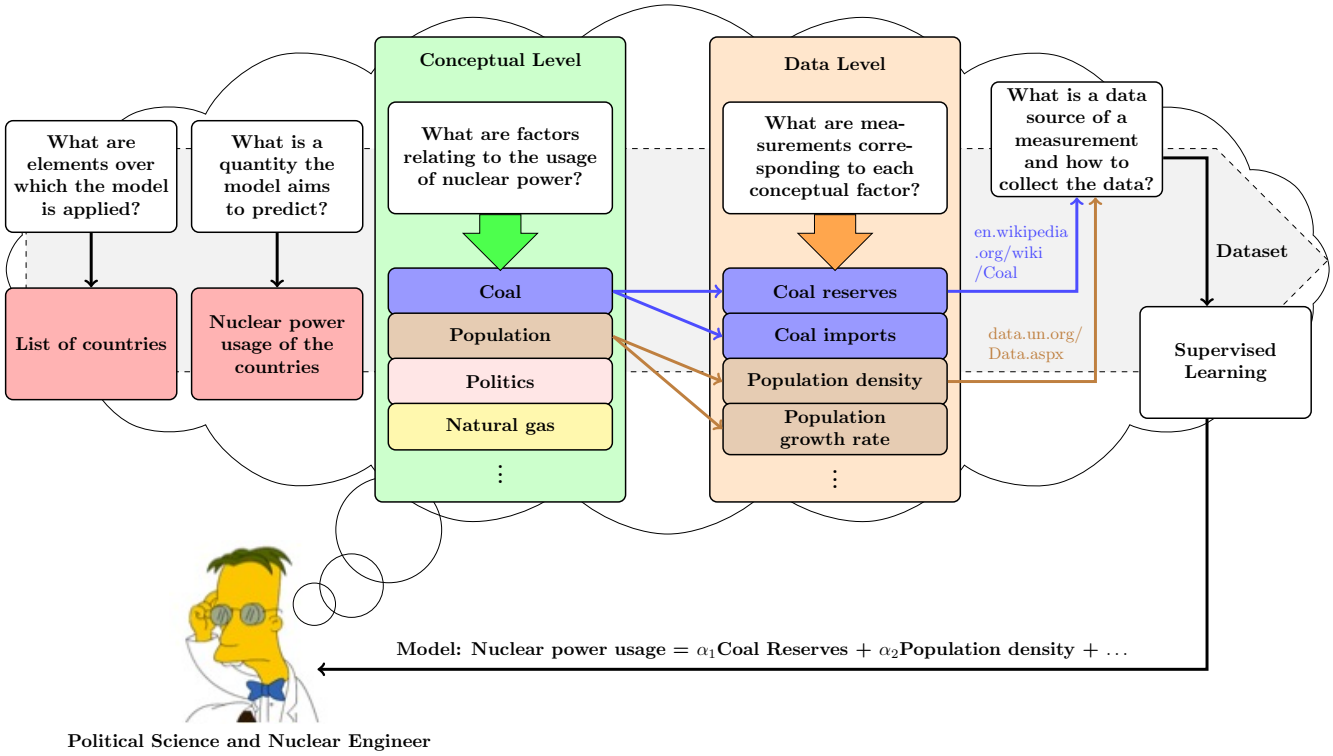

Figure 2: The modeling process extracted from Nelson and Sprecher [2008]. The expert carries out the process progressing from left to right to build a model for predicting nuclear power use. The expert starts by specifying a list of countries and then collecting nuclear power data of those countries from an existing data source. Background knowledge is used to come up with several hypotheses about the factors that may influence power usage. For example, a country with a large coal resource might be expected to use coal to produce electricity rather than using nuclear power. Thus, coal may affect nuclear power usage. Next, the expert moves to find measurements that correspond with the factors. For instance, coal reserves and coal imports are possible measurements relating to the coal in a country. The expert selects one of these measurements for each factor and uses a data source to provide data for the selected measurement. The collected data from these sources are used to construct a dataset for learning. Finally, the expert uses the resultant dataset with a designated learning method to build a model.

clear proliferation. Figure 2 shows the key ideas distilled from their modeling process. As experts, they used their knowledge of the nuclear domain to identify factors that are relevant to nuclear power use. Critically, the choices made regarding the data inputs for the model were directed by domain knowledge at a conceptual level rather than correlations in the data themselves. Moreover, most of the data that was used to build their model came from existing online data sources, such as government and public organization websites. While this modeling process had broad promise, it involves two challenges for a non-expert. Firstly, it relies heavily on domain knowledge from the person building the model. Secondly, the data collection itself was performed manually. Better data collection methods must be developed to help anyone collect data from different sources across a variety of formats.

We have set out to develop a semi-automated model building framework that adopts key ideas from, but also addresses the challenges of, the modeling process described above.[1]

[1] Although we have emphasized its use by Nelson and Sprecher, the approach represents a standard approach in several sciences.

The framework operates as follows: (i) An existing ontology is used as a source of background knowledge rather than relying on background knowledge from the person building a model. (ii) Since ontologies are precise machine manipulable representation of *a priori* structured relationships among concepts in a problem domain, they also enable a machine to explore the knowledge in an ordered manner to determine the relevance of domain concepts. Such concepts are then used to construct a hypothesis space, and data are used to find the best model in this space using a learning method. (iii) Operationalizing an ontological concept to corresponding measurements can be done by finding a data source corresponding to that concept, then looking for measurements from information published on that data source. (iv) A data extraction component is introduced into the framework to help a non-expert extract desired data from a source easily.

This paper is an attempt to bridge the gap between information extraction (IE) and learning from data, helping a user easily accomplish the learning task but also ensuring an accurate model is built. The IE task automatically extracts structured information from unstructured/semi-structured data sources so that a machine can semantically interpret

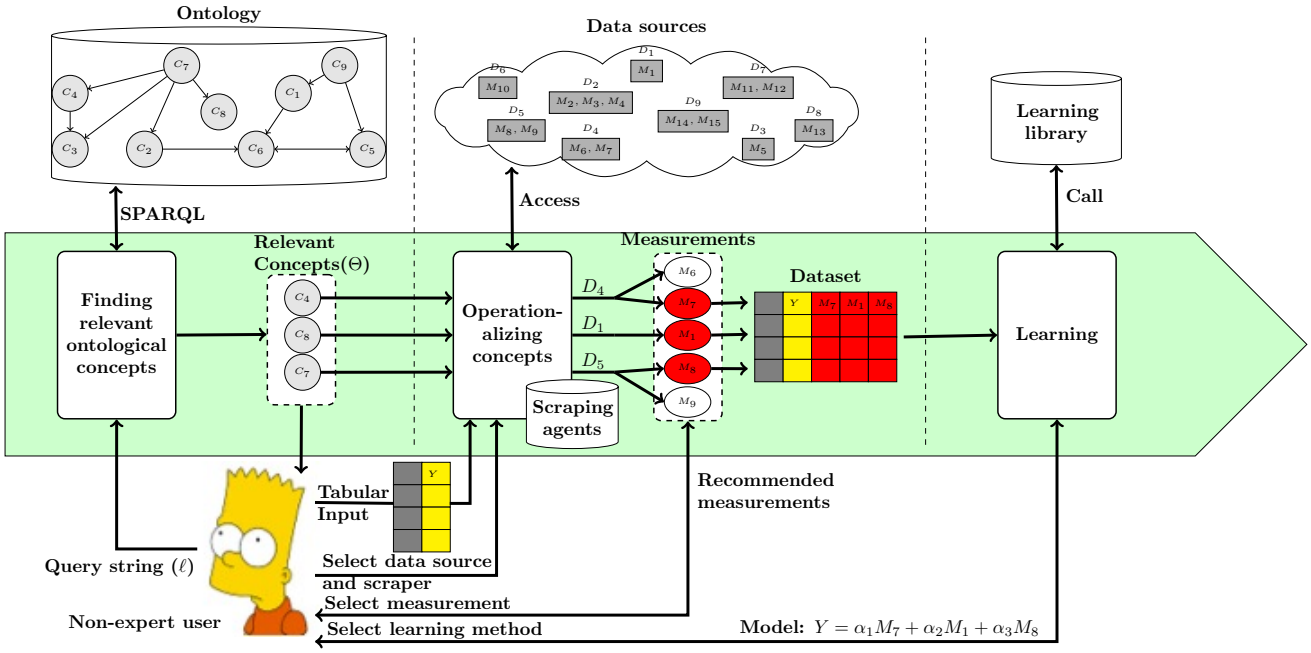

Figure 3: The proposed learning framework incorporates both an existing ontology and data sources to build a model from data. Knowledge in the ontology is used to construct an initial model that is composed of relevant ontological concepts and data, the latter being associated with the concepts and retrieved from existing data sources. The model is built and validated on the data using a selected learning method.

and automatically make use of those data [Etzioni et al., 2008]. Linking these two tasks through semantic relationships underlying the data enables a machine to automatically build a model of the domain. Using relationships of this form to interpret and evaluate attribute relevance helps to impose structure on the model to reduce overfitting. Thus, the framework is a way to make use of existing data and allow inexperienced users to cope with the data deluge.

Learning Framework

The framework is illustrated in Figure 3. It is an end-to-end approach that employs knowledge at two levels, *high-level concepts* and their relationships in an ontology, and *low-level data* from existing data sources. Model building proceeds from left to right and consists of three main components. Each component is associated with a corresponding outside component to perform its task. For example, using this framework to build a model for predicting nuclear power use of countries, a non-expert user starts by giving the first component a query string *"Nuclear power"* that will be the output of the model, representing the quantity he aims to predict. This component uses structured relationships in an ontology (which we assume is given) to automatically retrieve a set of concepts that are relevant to the input query, which we denote Θ.

So far, relationships are only captured at a high-level between concepts. To evaluate its predictive value, concepts in Θ must be operationalized to corresponding data. To perform this step the user describes the set of elements over which the model is applied (the model's domain), and what the model aims to predict (the model's output). Tabular input containing two columns is used for this purpose. In the example, the first column contains list of countries and the second column contains nuclear power usage data. We assume that the user already has this tabular input (*e.g.*, it could be collected from a web page publishing these data in tabular format). The input query and tabular input are used to form a question, in this case essentially asking *"Which attributes of countries are relevant to nuclear power?"* This tabular input is also stored in the framework as an initial dataset.

For each concept in Θ, a data source (*e.g.* a webpage, excel file, etc.) that provides measurements or values is specified. The user selects a suitable scraping module (*e.g.*, table scraping, list scraping) to extract contents. If one of the concepts is *"Coal"*, representing the energy resource, then the user may provide the Wikipedia article *http://en.wikipedia.org/wiki/Coal*, which contains several tables with data related to this concept. He selects the table scraping tool from the framework. The second component accesses the article and uses the selected scraper and data from the tabular input to extract all tables containing data about countries. These tables are represented as possible measurements related to the coal concept. The user chooses a table (the one providing coal reserves), which has six columns (*e.g.* SubBituminous, Lignite, Total, *etc.*) containing data related to different aspects of coal reserves. He selects one of these column. Data from the selected column are extracted and then added into the dataset. This step is repeated until all concepts in Θ are operationalized to data and added to the dataset. At this point, the dataset is ready for use in the learning process (*i.e.*, the third component). The user

selects a learning method (*e.g.*, linear regression, decision tree). The framework calls the selected method from an existing library (*e.g.* scikit-learn, R) to build a model from the dataset. The model is returned to the user so it can be used for predictions, and to determine (*e.g.*, from coefficients) the relative importance of the various concepts. The user might conclude, for instance, that large coal reserves reduce the likelihood of a country building a nuclear power station.

The framework exploits an existing relationship between three components: concepts in an ontology, existing data sources, and measurements of the concept and the given examples. Next we emphasize how the components involved help minimize the human effort required.

Finding Relevant Concepts from an Ontology

Given a query string and an ontology O containing knowledge encoded within an RDF data model, we retrieve and rank concepts from O by using its taxonomic hierarchy, which is a broadly applicable knowledge representation, useful across many ontologies. A taxonomic hierarchy is typically composed of two main elements: categories and concepts, in which closely related concepts are organized in the same category. Closely related specific categories are also organized into the same broader category forming super- and sub-categories relationships. We use SPARQL, the query language for RDF data, queries to acquire taxonomic knowledge from O without any preprocessing effort.

We adopt the ideas of the Hyperlink Induced Topic Search (HITS) algorithm [J.M.Kleinberg, 1999], originally for searching and ranking relevant web documents on a given topic by considering two different notions of relevance: hubs and authorities. Representing ontological categories as hubs and concepts as authorities, HITS can be employed on the ontology to find and rank relevant concepts for a given input query. The intuition underlying our algorithm is that if a category is relevant to the input query, then all concepts in that category should also be relevant to the query. Likewise, if a concept is relevant to the query then all categories containing the concept should be considered as relevant as well. Since an ontological element is explicitly defined either as a category or concept, it has only one score associated with it (*i.e.,* the hub score for a category and the authority score for a concept), which also means that the scoring computation from HITS can avoid iterative updates.

Scoring schemes

Two separate scoring schemes are used to quantify the relevance and utility of finding other related information for categories and concepts. Both schemes are the product of two components: voting and frequency. Voting captures the relationships between categories and concepts, while frequency tracks how often categories or concepts reappear during execution of the algorithm.

Category scoring: A category scores well if it links to many rare concepts that are relevant to an input query. Thus, category c's score comes from voting from concepts in c and how many relevant concepts appear in c so that

$$Score(c) = \sum_{i \in I_R} IF(i,c) \times \sum_{i \in I_R} IVote(i,c), \qquad (1)$$

where I_R denotes a set of relevant concepts, $IF(i,c)$ is a function with value 1 if a concept $i \in c$ or otherwise 0. $IVote(i,c)$ gives the vote based on rarity of a concept i, such that

$$IVote(i,c) = \begin{cases} 1/|\text{categories that contain } i| & \text{if } i \in c, \\ 0 & \text{otherwise.} \end{cases}$$

The $IVote$ function states that if i appears in many categories, it is not a rare concept and its vote is shared among the many categories, so a category containing many common concepts is penalized.

Concept scoring: A concept scores well if it is linked by many relevant categories. Thus, concept i's score comes from the votes of categories containing i, and how many relevant categories i appears in, such that

$$Score(i) = \sum_{c \in C_R} CF(i,c) \times \sum_{c \in C_R} CVote(i,c), \qquad (2)$$

where C_R denotes a set of relevant categories, $CF(i,c)$ is 1 if $i \in$ category c or otherwise 0, and $CVote(i,c)$ returns the score of category c as:

$$CVote(i,c) = \begin{cases} Score(c) & \text{if } i \in c, \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm Details

The algorithm performs three steps to find and rank concepts that are relevant to the input query ℓ in O. Details appear in Algorithm 1. The algorithm starts by finding a concept ϵ, whose label matches (textually) ℓ, to represent the query. If more than one concept is found, the first is selected. Next, all concepts that link to (inlinks of) or receive a link from (outlinks of) ϵ are retrieved from O to construct a set of initial relevant concepts, I_R.

Finding a set of relevant categories is performed in the second step (lines 5–24). For each concept in I_R, its categories are discovered and scores calculated using (1). The categories are sorted by score and the top n selected. Heuristics are used to further select categories from these top categories, finding those that (i) contain few concepts (*i.e.*, discarding categories which list names of films, animals, or scientists), (ii) contain neither very few nor many subcategories (*i.e.*, categories that are too specific or too general). The resulting set is denoted by C_R. Input parameters m, min, and max are used to adjust this behavior. Suitable values depend on the ontology and problem domain.

The final step (lines 25–34) retrieves all concepts from each category in C_R, scoring each with (2). All concepts are sorted by score before being returned as the output.

Implementation: DBpedia and Wikipedia

Our implementation leverages an existing ontology and online data sources; some details are worth discussing. We

Algorithm 1: Find and rank relevant ontological concepts

input: ℓ = Query string from user, O = SPARQL endpoint of ontology, n = Number of categories, m = Maximum concepts in category, min = Minimum sub-categories, max = Maximum sub-categories

Output: Θ = A set of ranked relevant concepts

$\epsilon \leftarrow$ **getConceptFromStr**(ℓ, O)
$Out \leftarrow$ **FindOutLink**(ϵ, O)
$In \leftarrow$ **FindInLink**(ϵ, O)
$I_R \leftarrow \epsilon \cup Out \cup In$
foreach $i \in I_R$ **do**
    $Cats \leftarrow$ **getCategories**(i, O)
    $vote \leftarrow 1/Length(Cats)$
    foreach $c \in Cats$ **do**
        $C_{vote}[c] \leftarrow C_{vote}[c] + vote$
        $C_f[c] \leftarrow C_f[c] + 1$
foreach $c \in C_{vote}$ **do** $C_{score}[c] \leftarrow C_{vote}[c] \times C_f[c]$
$C_{score\_sort} \leftarrow$ **SORT**(C_{score})
while $k < t$ **do**
    $c \leftarrow C_{score\_sort}[k]$
    $mb \leftarrow$ **CountConcepts**(c, O)
    $sb \leftarrow$ **CountSubCategories**(c, O)
    if $mb < m$ ***AND*** $(min < sb < max)$ **then** **Add**(C_R, c)
    $k \leftarrow k + 1$
foreach $i \in I_R$ **do** $I_f[i] \leftarrow 1$
foreach $c \in C_R$ **do**
    $Cons \leftarrow$ **getConcepts**(c, O)
    foreach $i \in Cons$ **do**
        $I_{vote}[i] \leftarrow I_{vote}[i] + C_{score}[c]$
        $I_f[i] \leftarrow I_f[i] + 1$
foreach $i \in I_{vote}$ **do** $I_{score}[i] \leftarrow I_{vote}[i] \times I_f[i]$
$\Theta \leftarrow$ **SORT**(I_{score})
Return Θ

used DBpedia [Bizer et al., 2009], the ontology counterpart of Wikipedia, as a source of background knowledge and used Wikipedia articles as data sources for corresponding DBpedia concepts. The vast amount of general knowledge in this ontology allowed testing on multiple case studies. Moreover, each DBpedia concept has a corresponding Wikipedia article often containing detailed information associated with that concept in different formats, such as text, list, or table. Our implementation exploits this connection to automatically get an article of a concept and use it as a primary data source for finding possible measurements of the concept. Thus, the user needn't specify a data source for a concept, allowing the system to build a model with minimal human effort (as highlighted in the sequence in Figure 4). Our system focuses on data that are published in tabular format since each table often encapsulates a complete, non-redundant set of facts [Bhagavatula, Noraset, and Downey, 2013], and tables structure data for easy automatic interpretation and extraction.

As shown in Figure 4, suppose the user would like to build a model for predicting GDP of countries. He can use this system by giving a query string *"Gross Domestic Product"*. The system retrieves relevant concepts from DBpedia. In the second step he selects a tabular input, with the first column containing a list of countries and the second column containing GDP data for each country. Then the system requests HTML data of Wikipedia articles corresponding to concepts output from the first step. The system processes HTML data of these articles to construct a dataset. In the third step, the user selects a learning method. The system calls that method from Scikit-learn library to build a model from the dataset. We can see that the user gives the system only three inputs and then lets the system carry out the remainder build a model. This meshes with our earlier motivation of producing a system to help the non-expert user to build a model from existing data easily. Implementation details of the first two steps are described in the following sections.

Finding relevant concepts from DBpedia

There are three important points when implementing Algorithm 1 with DBpedia. Firstly, internal links among wikipedia articles are used to find inlinks and outlinks of concept ϵ. DBpedia already contains these internal links as RDF triples via the *wikiPageWikiLink* predicate. Secondly, suitable values parameters n and m for DBpedia were found to be in the ranges 200–250 and 120–200, respectively, depending on a problem domain. While, the values of min and max were set to 6 and 30, respectively. Lastly, we added a DBpedia specific condition at the end of the algorithm to further select only concepts whose name (after removing all prefixes) starts with the term *List_of*. We found that corresponding articles for these concepts usually have tables providing data about a specific aspect of the concept. Focusing on these types of concept allows our implementation to automatically find and collect data as needed. For instance, the Wikipedia article *List_of_countries_by_GDP* has a table that contains data for GDP by country (suiting the preceding example). Figure 5 shows how this implementation works to find relevant concepts from DBpedia for an input query. The sample results obtained via this implementation using input queries: *Poverty* and *Gross Domestic Product* are shown in Table 1.

Collecting Data from Wikipedia Tables

Algorithm 2 automatically extracts data from a table on a Wikipedia page. For each concept in Θ, this algorithm starts by eliminating the concept that is redundant with ϵ by checking whether the string ℓ appears in the concept's label. It then acquires the URL of a Wikipedia article associated with the concept from DBPedia, requests the article in HTML, and extracts all tables from the result.

Heuristics are used to select a table that (i) has one column, we call an "example column," that partially matches to elements in the first column of the tabular input (countries in the nuclear example) (ii) has this "example column" appearing in the first or second column. If no table is selected, the concept is discarded. The algorithm then seeks columns in the selected table that contain numerical data. The numerical column closest to the example column is selected; if no numerical column is found, the example column is used to construct a new column that contains binary data (*i.e.*, value 1 is assigned to indicate that an element from the first column of the tabular input appears in the example column, otherwise value 0 is assigned).

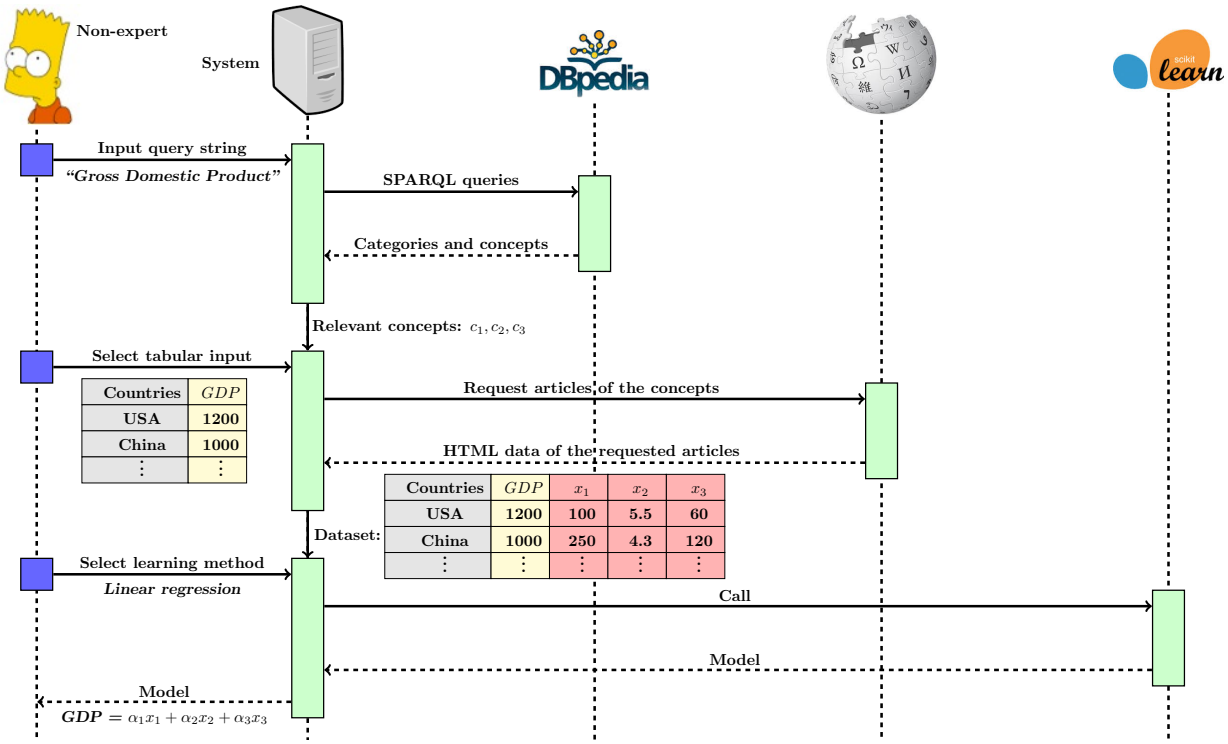

Figure 4: A diagram showing an interaction use case with our DBpedia and Wikipedia implementation. It shows how a model can be built with little human effort. Light (green) rectangles indicate operations that are done automatically and dark (blue/purple) ones show where user intervention is required.

At the end of this step, the system produces a tabular dataset containing data from attributes of elements over which the model is applied (countries in the nuclear example) and these attributes are also relevant to the input query.

Even though Algorithm 2 can automate table detection and data scraping of a Wikipedia article to operationalize a concept, this heuristic approach may select the wrong table or column to retrieve data yielding a poor final outcome. In a manual approach, on the other hand, the system only retrieves tables from an article and shows them to a user. The user then selects a table and a column to add to a dataset. Doing so, however, relies on the user having enough knowledge to select the correct table and column. A hybrid approach could be the better option, where the system initially chooses a table and a column for users and then lets them decide whether to accept that choice or change to another more appropriate table/column. The system can also provide a score for each table/column based on its contents (*e.g.,* number of matched countries in a table) to help the user make a wise choice.

Algorithm 2: Extracting data from Wikipedia article

input: Θ = Output from Algorithm 1, ℓ = Query from Algorithm 1, O = DBPedia's SPARQL endpoint, t_0 = Examples and output values in tabular format
Output: A dataset t in tabular format

$t \leftarrow t_0$
foreach $i \in \Theta$ **do**
  if ***Contain***(i, ℓ) **then** continue
  $e\_idx \leftarrow 100$
  $p \leftarrow$ **GetWikiArticle**(i, O)
  $T \leftarrow$ **ExtractTables**(p)
  foreach $\tau \in T$ **do**
    $c\_idx \leftarrow$ **GetExampleCol**$(\tau, t_0[0])$
    if $c\_idx \neq -1$ ***AND*** $c\_idx < e\_idx$ **then**
      $t_b \leftarrow \tau$
      $e\_idx \leftarrow c\_idx$
  $n\_idx \leftarrow$ **FindNCol**(t_b, e_idx)
  if $n\_idx = -1$ **then** $col_{new} \leftarrow$ **BCol**$(t_b[e\_idx], t_0[0])$
  else $col_{new} \leftarrow$ **NCol**$(t_b, e\_idx, n\_idx, t_0[0])$
  $t \leftarrow$ **AddCol**(t, col_{new})
Return t

Implementing the Framework with Other Ontologies and Data Sources

It is worth discussing implementation of the proposed framework with ontologies and data sources other than DBpedia and Wikipedia. Firstly, in Algorithm 1 we assume that there is a concept, ϵ, whose label matches an input query textually. An ontology used by the user could contain terminology that is more variable (*e.g.,* different acronyms and synonyms that do not directly match an input query). An additional step that can either automatically enrich the input query with synonyms or acronyms or find concepts that are semantically related to the query [Freitas et al., 2011;

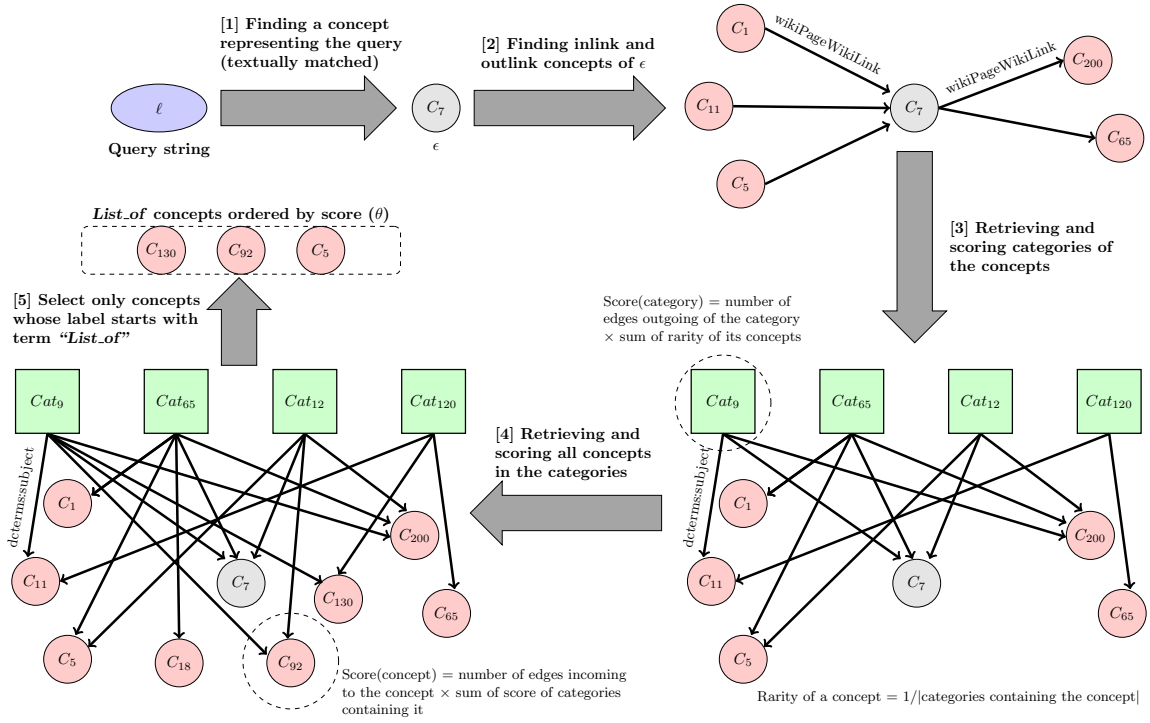

Figure 5: Running each step of the Algorithm 1 with taxonomic knowledge in DBpedia grows the graph as illustrated: [1] Finding a seed node, ϵ representing the input query, ℓ. [2] Outlinks and inlinks of ϵ (red filled circles) are added to the graph. [3] Categories of each added concepts are discovered. Relevant categories (filled rectangles) are selected to add to the graph. [4] Extra concepts from the added categories are retrieved and then added. [5] Concepts whose label starts with term *"List_of"* are selected and then returned as output

Pirro, 2012] should precede the algorithm to resolve this issue. The user first selects one of concepts returning from this step and uses it as ϵ. Secondly, other common predicates, such as *owl:sameAs* or *rdfs:seeAlso* could be used to find inlinks and outlinks of ϵ rather than using *wikiPageWikiLink* predicate, which is specific to DBpedia. Thirdly, for Algorithm 2, choosing a corresponding data source for a concept can be done directly by a user (*e.g.,* an Excel file containing data corresponding to the concept). A suitable user interface should be supplied to allow a user to specify the location of that data source. A search engine service, such as Google search service, could be employed in the system to help a user easily find a suitable data source from the Internet. By preprocessing a label of a concept and then inputing it as a query to the search service, the user can get locations of online data sources corresponding to the concept and then select one of them to retrieve the data. Lastly, scrapers for different data formats, such as lists, texts, files, should be added to the data scraping toolbox, along with an interface that allows a user to select a suitable scraper. Inconsistent or poorly structured/formated data pose difficulties for scraping. One way to handle this problem is to have the data scraping module resort to an interactive mode where a user can see scraped results immediately and adjust the module to resolve any scrapping errors on the fly.

In this work, we describe only the back-end of the system. To deploy this system, a suitable interface needs to be developed so that even inexperienced users without programming background can use the system. The system should also be able to record each decision made at each step by the user, so that the information can be used to improve the system. Moreover, the system should provide a mechanism to save and load a model, making the model reusable and supporting later refinement of the model.

Evaluation

Based on the motivation underlying this work, two separate evaluations were conducted. The first was conducted to show that the generalization of a model built from a dataset constructed by our system is improved over one based purely on data. Results from this evaluation provide support for the claim that the framework can help a non-expert build an accurate model. The second evaluation was conducted to assess quality of input attributes selected through use of the ontology's background knowledge. Results from these evaluations help show that the framework can help a non-expert user avoid problems associated with data dredging.

We conducted these evaluations by building models of four different problem domains: Nuclear power, Gross Domestic Product (GDP), Poverty, and Homelessness.[2] The

[2]These four were chosen because we found published examples of models built in order to make predictions in these domains. The generated models found via the framework did at least as well as these original models.

Table 1: Top 10 *"List_of"* concepts of input queries: *"Poverty"* and *"Gross Domestic Product"*

ℓ = ***"Poverty"***	ℓ = ***"Gross Domestic Product"***
countries_by_percentage_of_population_living_in_poverty	Australian_states_and_territories_by_gross_state_product
countries_by_unemployment_rate	sovereign_states_by_external_assets
countries_by_employment_rate	countries_by_economic_freedom
countries_by_Sen_social_welfare_function	freedom_indices
permaculture_project	countries_by_Sen_social_welfare_function
sovereign_states_and_dependent_territories_by_fertility_rate	countries_by_percentage_of_population_living_in_poverty
global_manpower_fit_for_military_service	countries_by_percentage_of_population_suffering_from_undernourishment
wars_and_anthropogenic_disasters_by_death_toll	countries_by_energy_intensity
countries_by_sex_ratio	countries_by_future_gross_government_debt
countries_by_infant_mortality_rate	countries_by_public_debt

elements over which the model is applied are: countries for the first three; U.S. states for the last one. For each problem domain, we formed two datasets in order to build two different models for comparison. The first dataset is constructed using our implementation as described in the preceding section. We denote this dataset by t_{ont}. The second baseline dataset, denoted t_{base}, was constructed by processing a URL of a Wikipedia category page containing links to many articles about the examples (*e.g.*, *http://en.wikipedia.org/wiki/Category:Lists_of_countries* for countries). We visit every article in the category that has the term *List_of_(countries/U.S. states)_by* appearing in its URL and has not yet been visited when constructing t_{ont}. Algorithm 2 is executed on these articles to create a temporary dataset denoted by t_{temp}. The dataset t_{base} is then constructed by concatenating all columns from t_{ont} and t_{temp}.

Before using t_{ont} and t_{base} for learning, the issue of missing data in these datasets had to be addressed. For each problem domain, we examined t_{base} to remove any columns (except columns from t_{ont}) and rows where 70% or more of their data are missing. We also removed the same set of examples from t_{ont}. Then, we manually filled in the remaining missing data for each column in both datasets by using an average value of data in that column. Finally, we invoked a learning method from the Scikit-learn library to build models from t_{ont} and t_{base} for each problem domain. The mean square error (MSE) is used to assess the quality of these models.

Improving Generalization of a Model

Using knowledge in the ontology to select input attributes for learning could help improve generalization beyond given examples if the ranking incorporates (either implicitly or explicitly) causal and/or independence assumptions. To test this claim, the datasets t_{ont} and t_{base} for each problem domain were divided into training (80% of examples) and test sets. Training sets from t_{ont} and t_{base} contained the same set of examples (*i.e.* both test sets also contain identical instances in the rows, but t_{ont} has strictly fewer columns). Two decision trees for regression were learned from these training sets and then each tree was tested with the corresponding training and test sets to calculate MSE values. Finally, 10-fold cross validation was used to find the average MSE. We performed this evaluation repeatedly while increasing the depth of both trees, so as to examine overfitting phenomenon.

From the results summarized in Figure 4, we observe that in all problem domains trees learned using t_{base} are prone to overfitting (*i.e.*, when the complexity of the tree increases, prediction error of the tree on the training set decreases rapidly, but increases on the test set), whereas the learned trees from t_{ont} produce lower prediction errors on test sets when examining comparable depth. The results show that the framework helps improve generalization of a learned model so it predicts more accurately on unseen examples.

Quality of the Top Ranked Attributes

We conducted a further experiment to show that the attributes automatically selected using the ontology's background knowledge are superior to attributes selected by correlations in a dataset. For each problem domain, we constructed two new datasets to carry out this experiment. The first dataset copies the first-n columns from t_{ont}, where $n = 5$ for all domains except GDP where $n = 8$ is used. Since each column in t_{ont} is ordered based on ranking of its corresponding concepts, this dataset captures the top-n concepts from our algorithm. The second dataset is constructed by using a univariate feature selection technique that selects columns based on correlations in the dataset to select the top-n columns from t_{base}. Then, we performed the same evaluations as before to assess performance of trees built from these new datasets. The results in Figure 5 show that trees learned by using top-n columns from t_{ont} produced prediction errors on test sets lower than trees learned by using top-n columns from t_{base} in all problem domains. These results suggest that selecting input attributes by using prior knowledge helps improve generalization of the learned model.

We also examined linear regression models to show that our framework is independent from the learning method used to build a model. One expects that the generalization of a model built from t_{ont} should still be better than using t_{base} even if the learning method is changed. For each problem domain, we constructed two datasets. The first contained the first-n columns of t_{ont}. The second dataset is constructed by building a linear model from all attributes in t_{base}, ranking attributes based on absolute value of their coefficient (from high to low), and then selecting the top-n attributes. Linear models with different complexities were built by limiting the number of attributes used. We started with all attributes

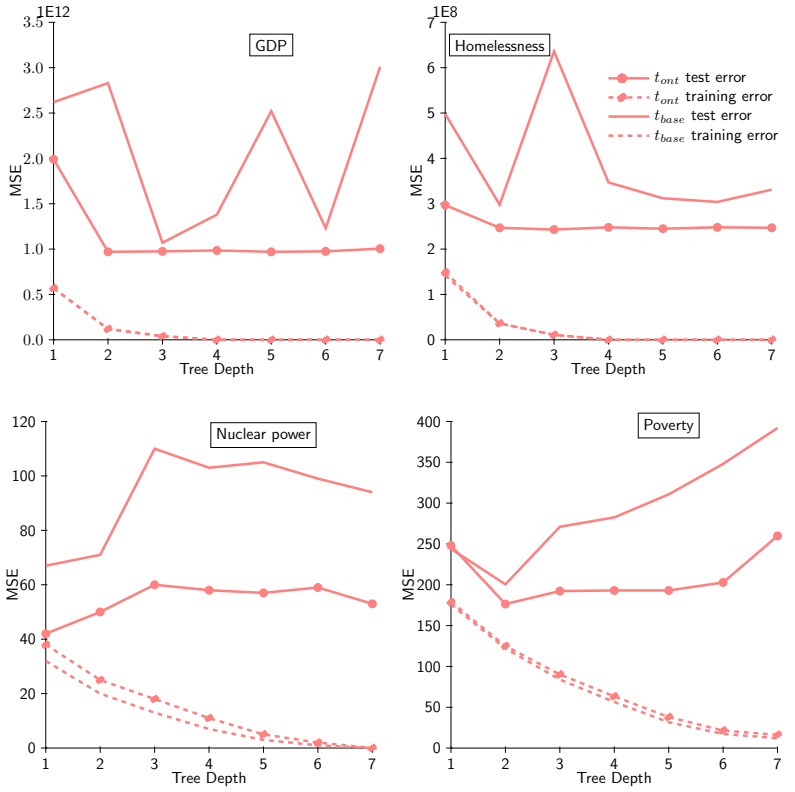

Figure 6: For each problem domain, decision trees with different depths were built using data from t_{ont} and t_{base}. Average MSE values from 10-fold cross validation when testing these trees on training and test sets are shown for each depth. The models built from t_{base} show the occurrence of overfitting, while generalization of learned models from t_{ont} is improved.

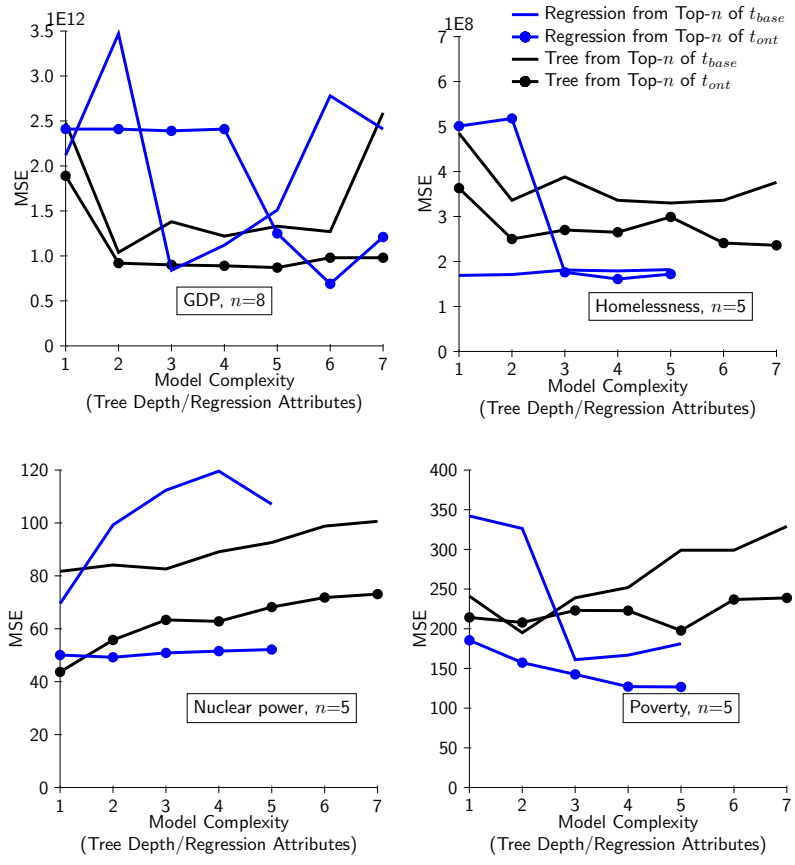

Figure 7: This figure compares test set prediction errors of models learned from top-n attributes from t_{ont} and t_{base} on the four problem domains. The models were constructed by using decision tree and linear regression methods. The results from both methods show that selecting attributes for learning by using prior domain knowledge helps improve generalization of the learned model.

in the dataset and iteratively removed the lowest ranked attribute. The results in Figure 5, especially in Nuclear power and Poverty domains, support the same conclusion as the decision tree results.

We note that GDP and Homelessness domains are challenging domains, and the learned models all have high prediction errors (MSE $\times 10^{12}$ and 10^{8}, respectively). These errors indicate that current attributes fail to capture the complexity of these problem domains. Improving our framework to enhance the quality of these models is part of future work.

Related Work

Several learning algorithms, such as Knowledge-Based Artificial Neural Network [Shavlik and Towell, 1989] and Bayesian belief networks [Pearl, 1988; Russell and Norvig, 2010], employ background knowledge to form an initial model and then use data to validate that model. Even though these algorithms have been demonstrated to outperform purely inductive learning [Mitchell, 1997], the main limitation of them is that they can accommodate only to a specific knowledge representation and learning method. In this work we present a framework that makes use of existing knowledge bases and data sources to build models of different problem domains. Also, this framework is designed to be independent of the learning method itself.

Semantic Web technology provides data models for publishing background knowledge in a structured format so that a machine can automatically interpret and make use of the knowledge. Searching for elements that are relevant to a given query from structured knowledge is one of the main topics in the field of Information Retrieval [Franz et al., 2009; Cheng et al., 2008; Blanco, Mika, and Vigna, 2011]. Most of these works, however, require some preprocessing effort. Unlike this paper, none of those works are specifically concerned with finding relevant ontological concepts to select attributes for learning.

Google Fusion Tables [Sarma et al., 2012] and WikiTables [Bhagavatula, Noraset, and Downey, 2013] include an operation termed "Relevant Join" which uses data published in tabular format and aims to find suitable columns from different tables for joining to a given table. Our work can be viewed as a system that automatically performs a Relevant Join to construct a dataset for learning. The main difference in our approach is that relevance of a column is justified by using prior domain knowledge rather than context in a table. Our system, moreover, need not be limited to data appearing in tabular formats. Data in another format, such as list, text, or query results, can also be used in the framework.

Conclusion and Future Work

This paper describes a learning framework that semi-automatically constructs a model using relevant ontological concepts and data attributes corresponding to those concepts. The attributes used in learning are selected by exploiting high-level knowledge separate from correlations within the data itself. As a consequence, the learned model is expected to generalize better than standard feature selection approaches. We implemented this framework with DBpedia and Wikipedia and then used the implementation to build four models from four different problem domains. Prediction errors on unseen examples from these models are shown to validate our claim. Moreover, the implementation helped build the models with very little human involvement.

What we present in this work is an attempt to address the changing needs of science: making it easier to produce models opens up vistas for inexperienced users, and helping automate the process of making sense of —and providing new interpretations for— existing data is one way to tame the deluge of data.

We believe that we are just starting to find uses of knowledge in an ontology to improve model generalization. Even though the results from the evaluations point out the possibility of using ontologies as background knowledge to help an inexperienced user build an accurate model from data, some important questions remain. Most obvious is that no formal explanation of how and why the system works was provided. Addressing these aspects is ongoing.

References

Bhagavatula, C. S.; Noraset, T.; and Downey, D. 2013. Methods for exploring and mining tables on wikipedia. In *Proc. of Interactive Data Exploration and Analytics (IDEA)*. ACM SIGKDD.

Bizer, C.; Lehmann, J.; Kobilarov, G.; Auer, S.; Becker, C.; Cyganiak, R.; and Hellmann, S. 2009. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web* 7:154–165.

Blanco, R.; Mika, P.; and Vigna, S. 2011. Effective and efficient entity search in rdf data. In *ISWC*, 83–97.

Cheng, G.; Ge, W.; Wu, H.; and Qu, Y. 2008. Searching semantic web objects based on class hierarchies. In *LDOW*.

Etzioni, O.; Banko, M.; Soderland, S.; and Weld, D. 2008. Open information extraction from the web. *CACM* 51(12):68–74.

Franz, T.; Schultz, A.; Sizov, S.; and Staab, S. 2009. Triplerank: Ranking semantic web data by tensor decomposition. In *ISWC*, 213–228.

Freitas, A.; Oliveira, J. G.; O'Riain, S.; Curry, E.; and Silva, J. C. P. D. 2011. Querying linked data using semantic relatedness: A vocabulary independent approach. In *Proc. Int. Conf. on Natural Language Processing and Information Systems (NLDB)*, 40–51.

J.M.Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *J. ACM* 46(5):604–632.

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill.

Nelson, P., and Sprecher, C. M. 2008. What determines the extent of national reliance on civil nuclear power. In *Proceedings of the INMM 49th Annual Meeting*.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann.

Pirro, G. 2012. Reword: Semantic relatedness in the web of data. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence*, 129–135.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A modern approach*. Prentice Hall.

Sarma, A. D.; Fang, L.; Gupta, N.; Halevy, A.; Lee, H.; F.Wu; Xin, R.; and Yu, C. 2012. Finding related tables. In *ACM SIGMOD*.

Shavlik, J., and Towell, G. 1989. An approach to combining explanation-based and neural learning algorithms. *Connection Science* 1(3):233–255.

Smith, G., and Shah, E. 2002. Data dredging, bias, or confounding: they can all get you into the bmj and the friday papers. *BMJ* 325(7378):1437–1438.

Tukey, J. 1977. *Exploratory Data Analysis*. Addison-Wesley.