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

This paper describes an end-to-end learning framework that allows a novice to create a model 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: certain conclusions may be supported by extra knowledge if, for example, there are reasons to trust a particular narrower set of hypotheses. This paper adopts the solution of using higher-level knowledge in order to allow this sort of domain knowledge to be inferred automatically, thereby selecting only 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; in the so-called data dredging problem, specious models may actually overfit and generalize poorly, and correlations might even be misinterpreted as causation. Without the capacity to distinguish real and spurious correlations, learning methods are prone to pick up regularity particular to a given dataset [Tukey, 1977]. An understanding of the processes that underlie and/or generated the data usually avoids these dangers, but the onus to be judicious ultimately falls on the person building the model.

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 fallacious conclusions. To help avoid these pitfalls, we developed a software framework that treats the whole modeling process rather than merely the model fitting stage. We describe a learning framework that exploits structural relationships in an ontology to automatically find concepts that are relevant to a given input query. We also introduce a data extraction component to help find and retrieve measurements related to ontological concepts from existing data sources.

The key idea in this paper is that knowledge in ontologies allows software to be cognizant of problem domain structure while choosing attributes for the learning algorithm. 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 an hypothesis space, and data are used to find the best model in this space using a learning method.

This paper attempts 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 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 such relationships to interpret and evaluate attribute relevance helps 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

We propose an end-to-end learning framework (see Fig. 1) that employs knowledge at two levels, *high-level concepts* and their relationships in an ontology, and *low-level data*

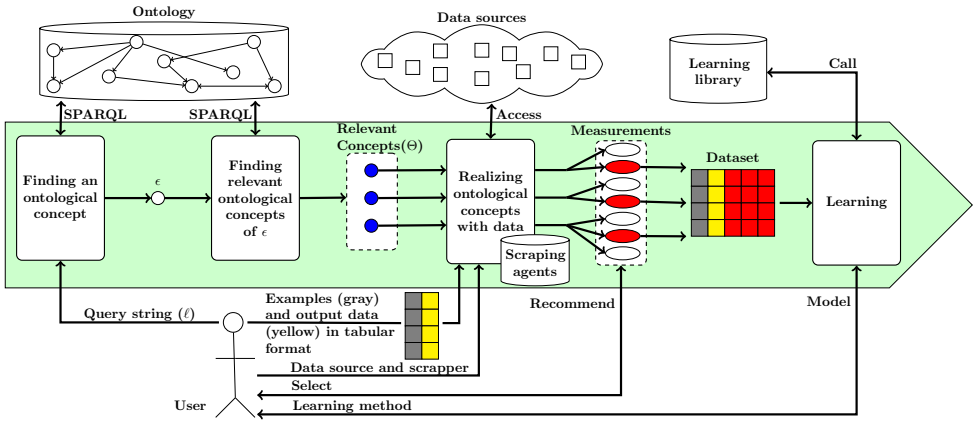


Figure 1: The proposed learning framework incorporates both an existing ontology and data sources to build a model from data. Knowledge in the ontology is used to construct an initial model that is composed of relevant ontological concepts and data, the latter being associated with the concepts and retrieved from existing data sources. The model is built and validated on the data using a selected learning method.

from existing data sources. The following is a sample scenario. A political scientist aims to understand civil nuclear proliferation, and so he builds a model that predicts the usage of nuclear power in a given country (*cf.* [Nelson and Sprecher, 2008]). He inputs a query string *"Nuclear power"* that will be the output of the model, representing the quantity he aims to predict. The software searches for a suitable concept in an ontology (which we assume is given) to represent the query. Denote the closest concept by ϵ. Next, the software uses structured relationships in the ontology to automatically retrieve a set of concepts relevant to ϵ, which we denote Θ.

So far, the model has only captures relationships at a high-level between concepts. To evaluate its predictive value, data corresponding to the concepts in Θ must be collected. The framework includes tools for the user to associate (i) elements from the list of examples (countries) and (ii) elements of data relating to that concept (nuclear power use of a country). For each concept in Θ, a data source (*e.g.* a webpage, excel file) 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 scientist 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 to extract all tables, and then chooses a table (the one providing coal reserves). This table has six columns (*e.g.* SubBituminous, Lignite, Total, *etc.*) containing data related to different aspects of coal reserves. He selects one of these columns. This links the high-level concept of *"Coal"* to low-level data used in the learning process. The user then selects a learning method (*e.g.*, regression, decision tree) and selected method is called from an existing library (*e.g.* scikit-learn, R). The model is returned to the scientist so he can use it for predictions, and determine (*e.g.*, from coefficients) the relative importance of the various concepts. He 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: a concept in an ontology, existing data sources, and measurements of the concept and the given examples. The following sections describe how the components 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 three general relationships among elements: (i) relationships between concepts, (ii) relationships between categories and concepts, (iii) relationships between categories. These are broadly applicable relationships, useful across many ontologies. We use SPARQL, the query language for RDF data, queries to acquire necessary 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. 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. The 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

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

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 in category, max = Maximum sub-categories in category

Output: Θ = A set of ranked relevant concepts

1 $\epsilon \leftarrow$ **getConceptFromStr**(ℓ, O)
2 $Out \leftarrow$ **FindOutLink**(ϵ, O)
3 $In \leftarrow$ **FindInLink**(ϵ, O)
4 $I_R \leftarrow \epsilon \cup Out \cup In$
5 **foreach** $i \in I_R$ **do**
6 $Cats \leftarrow$ **getCategories**(i, O)
7 $vote \leftarrow 1/Length(Cats)$
8 **foreach** $c \in Cats$ **do**
9 $C_{vote}[c] \leftarrow C_{vote}[c] + vote$
10 $C_f[c] \leftarrow C_f[c] + 1$
11 **foreach** $c \in C_{vote}$ **do** $C_{score}[c] \leftarrow C_{vote}[c] \times C_f[c]$
12 $C_{score\_sort} \leftarrow$ **SORT**(C_{score})
13 **while** $k < t$ **do**
14 $c \leftarrow C_{score\_sort}[k]$
15 $mb \leftarrow$ **CountConcepts**(c, O)
16 $sb \leftarrow$ **CountSubCategories**(c, O)
17 **if** $mb < m$ ***AND*** $(min < sb < max)$ **then** **Add**(C_R, c)
18 $k \leftarrow k + 1$
19 **foreach** $i \in I_R$ **do** $I_f[i] \leftarrow 1$
20 **foreach** $c \in C_R$ **do**
21 $Cons \leftarrow$ **getConcepts**(c, O)
22 **foreach** $i \in Cons$ **do**
23 $I_{vote}[i] \leftarrow I_{vote}[i] + C_{score}[c]$
24 $I_f[i] \leftarrow I_f[i] + 1$
25 **foreach** $i \in I_{vote}$ **do** $I_{score}[i] \leftarrow I_{vote}[i] \times I_f[i]$
26 $\Theta \leftarrow$ **SORT**(I_{score})
27 Return Θ

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

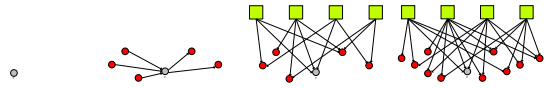
$$Score(i) = \sum_{c \in C_R} CF(i,c) \times \sum_{c \in C_R} CVote(i,c), \qquad (2)$$

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

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. In each step, the algorithm grows the graph as illustrated in Figure 2. The algorithm starts by finding a concept ϵ, whose label matches (textually) ℓ. If more than one concept is found, the first is selected. Next, all concepts that



(a) (b) (c) (d)

Figure 2: The graph shown at each step of Algorithm 1: (a) A seed node representing ϵ. (b) Outlinks and inlinks of ϵ (red filled circles) are added to the graph. (c) Categories of each added concept are discovered. Relevant categories (filled rectangles) are selected to add to the graph. (d) Extra concepts from the added categories are retrieved and then added.

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 discussion. We 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 information associated with that concept. Our implementation exploits this connection to automatically collect data from an article, allowing the system to build a model with minimal human effort, as highlighted in the sequence in Figure 3.

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 the 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 value of min and max is 6 and 30, respectively. Lastly, we added an heuristic method to 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

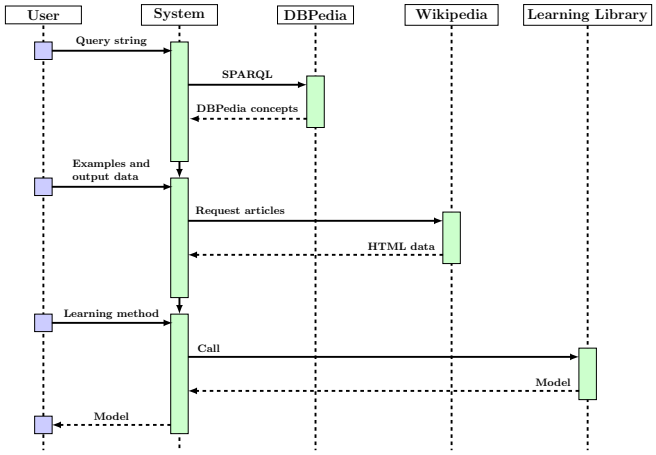


Figure 3: A diagram showing a use case interaction with our DBPedia and Wikipedia implementation. It shows how a model can be built with little human effort. Green rectangles indicate operations that are done automatically and purple ones show where user intervention is required.

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 countries (a fits the preceding example). The sample results obtained via this implementation using input queries: *Poverty* are shown in Table 1.

Table 1: Relevant *List_of* concepts of *Poverty*, where n = 250 and m = 150 (top 20 of 27 results are shown)

countries_by_percentage_of_population_living_in_poverty (498.98)
countries_by_unemployment_rate (123.24)
countries_by_employment_rate (123.24)
countries_by_Sen_social_welfare_function (50.95)
permaculture_projects (48.58)
sovereign_states_and_dependent_territories_by_fertility_rate (38.46)
global_manpower_fit_for_military_service (19.43)
wars_and_anthropogenic_disasters_by_death_toll (19.43)
countries_by_sex_ratio (19.43)
countries_by_infant_mortality_rate (19.43)
countries_by_life_expectancy (19.43)
sovereign_states_and_dependent_territories_by_death_rate (19.43)
countries_by_distribution_of_wealth (14.48)
countries_by_income_equality (14.48)
socialist_economists (7.58)
Australian_states_and_territories_by_gross_state_product (5.46)
sovereign_states_by_external_assets (5.46)
freedom_indices (5.46)
countries_by_economic_freedom (5.46)
countries_by_refugee_population (5.33)

Collecting Data from Wikipedia Tables

Algorithm 2 automatically extracts data from a table on a Wikipedia page. Data extraction from tables is advantageous 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. 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.

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

Heuristics are used to select a table that (i) has one column, we call an "example column," that partially matches to objects of the domain of the model (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 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.

Evaluation

We conducted an evaluation by building models of four different problem domains: Nuclear power, Gross Domestic Product (GDP), Poverty, and Homelessness. The 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 has to be addressed. For each prob-

lem domain, we examined t_{base} to remove any columns (except columns from t_{ont}) and rows where 70% 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. The framework was evaluated in two ways as follows.

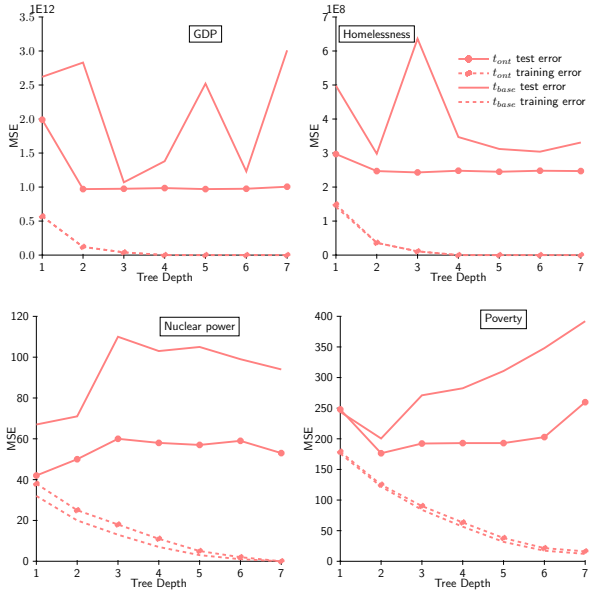


Figure 4: For each problem domain, decision trees with different depths were built using data from t_{ont} and t_{base}. Average MSE values from 10-fold cross validation when testing these trees on training and test sets are shown for each depth. The models built from t_{base} show the occurrence of overfitting, while generalization of learned models from t_{ont} is improved.

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} of 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 the overfitting phenomenon.

From the results 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.

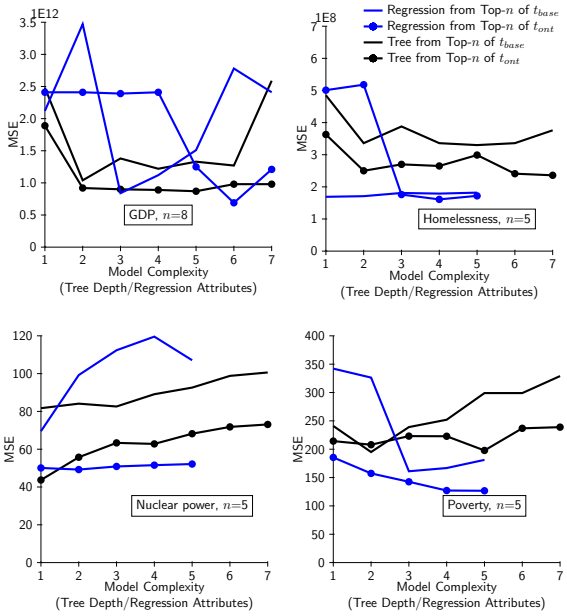


Figure 5: This figure compares test set prediction errors of models learned from top-n attributes from t_{ont} and t_{base} on the four problem domains. The models were constructed by using decision tree and linear regression methods. The results from both methods show that selecting attributes for learning by using prior domain knowledge helps improve generalization of the learned model.

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. For each problem domain, we constructed two datasets. The first con-

tained 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 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 of 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 format. Data in another format, such as list, text, or query results, can also be used in the framework.

Summary and Conclusion

This paper describes a learning framework that 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.

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)*.

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.

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. Technical report, Nuclear Security Science and Policy Institute; Department of Political Science, Texas A&M University.

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

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.

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