Automated Feature Generation from Structured Knowledge

Weiwei Cheng
University of Marburg
Germany
roywwcheng@gmail.com

Gjergji Kasneci
Microsoft Research
Cambridge, UK
gjergjik@microsoft.com

Thore Graepel
Microsoft Research
Cambridge, UK
thoreg@microsoft.com

David Stern
Microsoft Research
Cambridge, UK
dstern@microsoft.com

Ralf Herbrich
Microsoft Research
Cambridge, UK
rherb@microsoft.com

ABSTRACT

The prediction accuracy of any learning algorithm highly depends on the quality of the selected features; but often, the task of feature construction and selection is tedious and non-scalable. In recent years, however, there have been numerous projects with the goal of constructing general-purpose or domain-specific knowledge bases with entity-relationship-entity triples extracted from various Web sources or collected from user communities, e.g., YAGO, DBpedia, Freebase, UMLS, etc. This paper advocates the simple and yet far-reaching idea that the structured knowledge contained in such knowledge bases can be exploited to automatically extract features for general learning tasks. We introduce an expressive graph-based language for extracting features from such knowledge bases and a theoretical framework for constructing feature vectors from the extracted features. Our experimental evaluation on different learning scenarios provides evidence that the features derived through our framework can considerably improve the prediction accuracy, especially when the labeled data at hand is sparse.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Design, Experimentation

Keywords

query language, semantic features, structured knowledge

1. INTRODUCTION

Machine learning tasks often require identifying complex patterns from data and use these patterns to make appropriate decisions. For example, a typical task in supervised learning is to predict labels for unseen data items based on a training sample of labeled data items. The problem is often formulated as finding a function $f : \mathcal{X} \to \mathcal{Y}$ which maps objects $x \in \mathcal{X}$ to labels or target values $y \in \mathcal{Y}$. Typically, the problem is broken down into two steps.

1. Choose a fixed feature map $\phi : \mathcal{X} \to \mathcal{F}$ which takes input items $x \in \mathcal{X}$ and maps them to feature vectors $\boldsymbol{\phi} \in \mathcal{F}$, where one often chooses $\mathcal{F} = \mathbb{R}^n$.
2. Define a (parameterized) class $\mathcal{H}$ of functions $h \in \mathcal{H}$ with $h : \mathcal{F} \to \mathcal{Y}$ and learn (the parameters of) the best function.

The predictive power of the learned classifier depends strongly on choosing a feature map ϕ that is informative with respect to the prediction task at hand. In the case of $\mathcal{F} \subset \mathbb{R}^n$ the components of the feature vector $\boldsymbol{\phi} \in \mathcal{F}$ are referred to as features.

In the machine learning literature, the input items x have traditionally been identified with their feature representation $\phi(x)$, and often the second of the two problems above has been considered in isolation. This may be natural if the data set is already given in the form $(\boldsymbol{\phi}_{(i)}, y_{(i)})$. However, for more and more large-scale learning problems that are typical for web-scale applications of machine learning, the situation is different. Consider, for example, the case of building a machine learning system for movie recommendation. The training data consists of users, movies and ratings (of movies by users). Suppose further that we are in possession of a knowledge base that contains information about users and movies. Instead of manually constructing and extracting user features and item features, would it not be great if we could specify a query that automatically constructs feature vector representations of users and items to be used by the machine learning system? Note, that the availability of such a mechanism would turn the knowledge base into a re-usable feature store that can provide different features for different applications.

The process of building appropriate features often involves two phases: A feature construction phase and a feature selection phase. In this paper, we will mainly be concerned with the question of how to construct good features from semantic knowledge bases. To this end we devise query patterns for constructing features from knowledge bases, in which the stored information is organized as entity-relationship-entity triples, e.g., in an RDFS representation.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

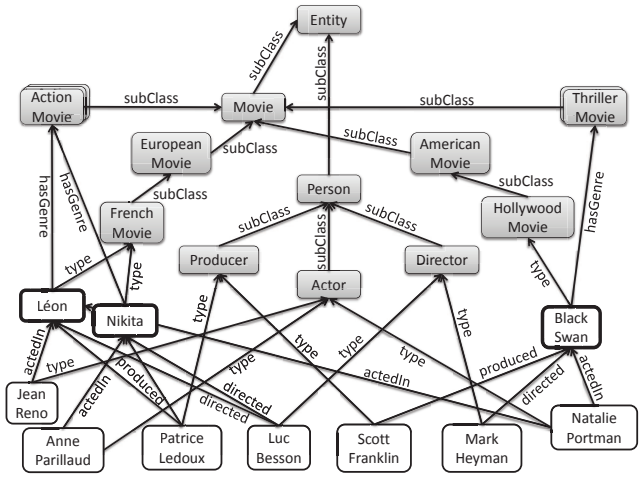

Figure 1: Sample knowledge graph from the YAGO knowledge base. The shaded nodes represent classes of entities, the remaining nodes represent individuals, and the edge labels represent relationships.

Such a knowledge base could be derived from a specific domain or from open, general-purpose RDF data sources such as YAGO [19], DBpedia [18], and LOD [17]. Figure 1 gives an example of a knowledge graph that organizes information about the movies "Léon", "Nikita" and "Black Swan" in the form of entities and relationships.

State-of-the-art knowledge bases typically have ontological components that organize entities into class-subclass hierarchies based on the *isA* or *type* and *subClass* relationships (see Figure 1). These taxonomic components are useful for understanding the roles of entities at different levels of abstraction. For example, a given movie may be in the class "French movies", which may be a sub-class of "European movies", which in turn could be a sub-class of "Movie". Other features about entities could be derived from their common relations to other entities, e.g., production dates, movie casts, costs, etc. Feeding these features to a suitable machine learning algorithm would enable it to learn a concept like user preference at the right level of abstraction from data. A framework that would allow the construction of this kind of features would be a significant step towards integrating the world of structured knowledge with that of machine learning.

2. RELATED WORK

The problem of improving predictive performance by incorporating structured knowledge has been addressed in previous work, especially in the fields of text classification and information retrieval [1, 2, 3, 4, 5, 6, 7, 8]. Despite a number of interesting results, the conclusions drawn from these works are multi-faceted, i.e., sometimes the structured knowledge is of benefit, and sometimes, even with carefully tuned NLP procedures and well-organized additional features, no clear empirical benefits are observed. The reasons for such negative observations are of course manifold and often task dependent. It is, however, worth noticing that most of the previous research in this realm focuses on semantics derived from the lexical level, which means that structured knowledge has been derived from the underlying text corpus by various parsing, pattern-matching, or other NLP techniques. Even when corpus-independent general-purpose knowledge sources are used (such as WordNet or other thesauri), the main features that are exploited are based on synonymy. For hypernymy-based features derived from general-purpose taxonomic thesauri, such as WordNet, it has been empirically shown that they can provide performance gains [1, 2, 8]. Along similar lines, [4, 5] provide preliminary evidence that class-oriented features derived from domain-specific knowledge bases can improve the underlying learning task. Therefore, we go one step further and advocate entity-relationship-oriented features with rich contextual and abstraction levels derived from related individuals and class-subclass hierarchies of state-of-the-art knowledge bases. In what follows, we discuss some of the previous work in more detail.

Scott et al. [1] incorporate the hypernymy information provided by WordNet into the Ripper rule learning system [9]. Tests on Reuter-21578 data show that the achieved improvement is category dependent. More specifically, when comparing with the conventional bag-of-words representation, the semantic features turn out to be effective for a small subset of categories. Later work, such as [8], by Kehagias et al., and [2], by Moschitti et al., follows the same line as [1], while taking more datasets and more classifiers into consideration. But neither [8] nor [2] provides sound evidence that the additional semantic features are attractive alternatives to the bag-of-words features. However, the authors of [2] have pointed out that the rather disappointing outcome of their analysis should not be taken as a general conclusion on the role of semantic features in classification tasks. It merely suggests that the elementary textual representation is very effective in many cases. And indeed, when the text is written consistently and the words are discriminative, there is not much room for improvement. However, more recent work, like [4], by Degemmis et al., and [5], by Bloehdorn et al., has reported more promising results. A key advantage of [4, 5] over earlier work is that the adapted features come from domain-specific ontologies. In [4], for example, background knowledge that is manually extracted from the Internet Movie Database (IMDb)[1] has improved the classification on the EachMovie dataset. Similarly, in [5] the MeSH thesaurus[2] (a tree-shape ontology that has been compiled out of the Medical Subject Headings of the United States National Library of Medicine) has led to better results than the more general WordNet in numerous cases, especially when medical knowledge is of interest.

Much of the related work done in the area of information retrieval has been in the context of query expansion techniques [3, 7]. Recent work by Lu et al. [6] proposes document and query-side processing to extract entity-oriented features. The proposed technique outperforms a major commercial web search engine[3] by 4% on average in terms of DCG@5.

Another domain in which structured knowledge has been widely adopted is question-answering. In [11], the usefulness of semantic features for such systems is well recognized and emphasized. Moreover, [10] points out that even simple question-answering approaches that use limited background knowledge can significantly improve a system's ability to find appropriate answers. This is also shown by the outstanding success of the Watson system [22], which exploits state-of-

[1]http://www.imdb.com/
[2]http://www.nlm.nih.gov/mesh/
[3]It was not pointed out which search engine it was.

the-art knowledge bases to compete with humans in the Jeopardy game.

2.1 Contributions

In all the above methods, the described semantic features are quite shallow; often they are derived from the underlying corpus through different NLP techniques, and when background knowledge is used, synonymy and hypernymy have been the main selection criteria. Furthermore, in those approaches that use background knowledge for feature generation, the construction of semantic features is somewhat ad-hoc, in that it does not follow a general and unified construction strategy. This makes the methods, from an evaluation point of view, incomparable (even when they use the same background knowledge) and often difficult to re-implement.

The contributions of this paper are the following.

- We introduce an expressive theoretical framework for constructing semantic features from a given knowledge base organized as a triple store. For a given entity, the framework allows the construction of entity-relationship-oriented features at different levels of abstraction.
- We discuss various strategies for incorporating the above features into a prediction model.
- Finally, we give experimental evidence for the effectiveness of semantic features constructed through the proposed framework.

The remainder of the paper is organized as follows. In Section 3.1 we introduce a graph-based query language for extracting entity-relationship-based features from a knowledge base. In Section 3.2 we present the theoretical framework for the construction of feature vectors and give an overview of viable methods for their applicability. The experimental evaluation of our approach is presented in Section 4.

3. SEMANTIC FEATURES

In the following we assume that we are given a corpus $\mathcal{C}$ over the elements of which we aim to make predictions (e.g., Web documents, tweets, movie descriptions, product reviews, etc.), a learning method $\mathcal{M}$ that expects semantic features about the entities that are recognized in $\mathcal{C}$, a knowledge base $\mathcal{K} = (\mathcal{G}, \mathcal{E}, \mathcal{R}, \mathcal{Q})$, where $\mathcal{E}$ and $\mathcal{R} \subset \mathcal{E}$ are the entities and relationships that are represented in $\mathcal{K}$, respectively, $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the knowledge graph (of entity-relationship-entity triples) stored in $\mathcal{K}$, and $\mathcal{Q}$ represents a set of functions (or a query language) for querying the knowledge base. Furthermore, a basic assumption is that $\mathcal{E}(\mathcal{C}) \subset \mathcal{E}(\mathcal{K})$, where $\mathcal{E}(\mathcal{C})$ and $\mathcal{E}(\mathcal{K})$ denote the sets of entities in the corpus and in the knowledge base, respectively.

For the construction of semantic features, we start out with the following tasks. Given an entity $x \in \mathcal{E}(\mathcal{C})$ which has been recognized in the corpus, (1) use the knowledge base $\mathcal{K}$ to extract semantic features, (2) provide an automatic mechanism for constructing a vector $\phi(x)$ from the extracted features. For the first task, we propose a graph-based query formalism that builds on SPARQL [15], the W3C recommendation for querying RDF data; for the second task we provide a theoretical framework for constructing the desired feature vector.

3.1 Querying for Semantic Features

In the literature, there have been numerous proposals for query languages on graph-structured data. Most prominent among these are declarative languages such as SPARQL [15], Conjunctive Datalog, or NAGA [20]. Both SPARQL and NAGA have been designed for querying triple stores, which represent semantic networks. A query can be viewed as a semantic graph consisting of entities, relationships and variables. The variables of the query are substituted appropriately with entities or relationships by matching the given parts of the query and its structure with a subgraph from the underlying knowledge graph $\mathcal{G}$. Syntactic sugar aside, a basic query for the above languages can be defined as follows.

DEFINITION 3.1. **Basic Query.** *A query over a set of variables $\mathcal{V}$ for a knowledge graph $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is a semantic connected graph $q \subseteq (\mathcal{E} \cup \mathcal{V}) \times (\mathcal{R} \cup \mathcal{V}) \times (\mathcal{E} \cup \mathcal{V})$.*

We see that a query is basically a graph, the edges of which are triples from the set $(\mathcal{E} \cup \mathcal{V}) \times (\mathcal{R} \cup \mathcal{V}) \times (\mathcal{E} \cup \mathcal{V})$. A query answer is defined as a subgraph of $\mathcal{G}$ that matches the query graph, or more specifically:

DEFINITION 3.2. **Query Answer.** *An answer to a query q on a semantic graph $\mathcal{G}$ is a graph homomorphism $\sigma : q \mapsto \sigma_q \subseteq \mathcal{G}$ that preserves the given entity and relationships in q, and substitutes the variables with entities and relationships from $\mathcal{G}$.*

In addition, SPARQL provides useful mechanisms for performing projections or aggregations on the variables of the query. For example, for a query q a projection is defined as the set $\pi_{v_1,...,v_k}(q)$ of entities and relationships that are substituted for the variables $v_1, ..., v_k \in \mathcal{V}(q)$ of q in the query answers. In SPARQL this is enabled through a `SELECT` clause[4] (see also example in Section 3.1.1).

Another useful functionality of SPARQL is the aggregation on the results of a query q. The aggregation function is given by $\gamma_{f_1(v_1),...,f_k(v_k)}(\pi_{v_1,...,v_k}(q))$, where $v_1, ..., v_k \in \mathcal{V}(q)$ are variables of q and each f_i is an aggregation function (e.g., `COUNT, SUM, MAX, AVG,` etc.) that is applied to the entities or relationships substituted for the variable v_i. Additionally SPARQL provides useful operators such as `DISTINCT`, for removing duplicates, `UNION`, for building the union of results, `OPTIONAL`, for query edges that should be matched optionally, etc. For further information on the functionality of SPARQL we refer the reader to [15].

NAGA on the other hand does not support projection and aggregation but goes beyond the basic queries defined in 3.1 by allowing the definition of regular expressions over relationships in the query edges. More specifically a NAGA query is defined as follows.

DEFINITION 3.3. **NAGA Query.** *Let $\texttt{RegEx}(\mathcal{R})$ be the set of all regular expressions over $\mathcal{R}$. A NAGA query over a set of variables $\mathcal{V}$ for a knowledge graph $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is a semantic connected graph $q \subseteq (\mathcal{E} \cup \mathcal{V}) \times (\texttt{RegEx}(\mathcal{R}) \cup \mathcal{V}) \times (\mathcal{E} \cup \mathcal{V})$.*

An answer to a NAGA query is defined similarly to Definition 3.2, with the difference that now entire paths from $\mathcal{G}$ can match the regular-expression of a query edge;

[4]In general the `SELECT` clause returns a multiset. The keyword `DISTINCT` can be used to remove duplicates.

this is the case when the sequence of relationship labels on an answer path matches the regular expression of the query edge. For an explicit definition of an answer to a NAGA query we refer the reader to [20].

The main point about NAGA queries is that they are useful when the relationships are transitive or when multiple relationships are of interest. For example, we could be interested in the query that asks for all genres and superclasses of the movie Black Swan in the knowledge graph. The corresponding NAGA query would be:

```
Black_Swan hasGenre|(type subClass*) ?x
```

In the above example, `?x` represents a variable. The `hasGenre` relationship accesses the movie genres of Black Swan, `type` returns all direct classes of Black Swan, and `subClass` returns the superclasses of a given class. Finally, '|' reflects the "OR" condition in the query, and the Kleene star captures the transitivity of the `subClass` relationship. An answer to the above query would replace the variable `?x` with a genre or a superclass of the movie 'Black Swan'. Assuming that Figure 1 represents the knowledge graph $\mathcal{G}$, a possible answer to the above query graph could be:

```
Black_Swan  type  Hollywood_Movie
Hollywood_Movie  subClass  American_Movie
American_Movie  subClass  Movie
```

For our goal of providing the user with a powerful tool for extracting meaningful graph-based features for entities, we need a query language that is expressive enough (e.g., enables the access of class and relationship information about entities in a transitive fashion) and thus allows us to select features at an adequate level of abstraction and granularity. To this end we combine the advantages of SPARQL with those of NAGA. Hence, we refer to the extension of SPARQL by the regular expression capabilities of NAGA as the *Extended SPARQL Query Language*, or ESPARQL for short.

We present some useful ESPARQL query patterns for feature construction in the following paragraph.

3.1.1 Exemplary Query Patterns

Suppose we were interested in all the entities from the knowledge graph $\mathcal{G}$ that are neighbors of an entity `e` that was recognized in the corpus $\mathcal{C}$. This could be useful for text classification tasks, when the underlying text corpus is sparse, and when the given entity has never been seen before. The corresponding query is shown below. Exemplary results to such queries on the YAGO knowledge base are given in Sections 3.2.2 and 4.3.2.

```
SELECT ?r ?x
WHERE { (e  ?r  ?x) UNION (?x  ?r  e) }
```

Note that the entity `e` might occur as a subject or an object in the triples of $\mathcal{G}$.

Another generic query for feature extraction is the one that asks for all superclasses of an entity `e` that was recognized in $\mathcal{C}$.

```
SELECT DISTINCT ?x
WHERE { e  type subClass*  ?x }
```

Note that we are using the regular expression `type subClass*` to access all the superclasses of `e`.

Now suppose that a movie `m` was recognized in the corpus $\mathcal{C}$. By using ESPARQL we can be precise about the semantic features we aim to extract for `m`. The example query below asks for the number of Oscar-winning actors or directors of `m`.

```
SELECT COUNT (DISTINCT ?x)
WHERE { ?x  starredIn|directed  m.
        ?x  hasWon  Oscar_Award  }
```

Another problem in classification is the case in which the feature representing a given entity does not generalize. For example, extrapolating a model of user behavior based on user IDs is often impossible. However, such a model could be derived if the IDs could be grouped by the social or geographical groups they belong to. A practical example is given when users are represented by IP addresses. A generic ESPARQL query pattern for generalizing the description of a user with ID `u` is depicted below.

```
SELECT ?y
WHERE { ?x  hasID  u.
         u  belongsToUserLocation  ?y.
        ?y  type subClass*  country  }
```

The above query enables the representation of users by their country information. In this example the user ID could be similar to an IP address and could belong to a specific country.

Note that all query patterns presented here are modular in nature and can be composed to derive richer semantic information from the knowledge base. Consequently, as we will see in the next section, our framework is modular as well. It allows the storage and composition of ESPARQL queries on demand (analogously to stored procedures). The question that we still need to answer is how to represent the semantic information extracted through ESPARQL queries adequately for a given learning task. This question is addressed in the next section.

3.2 Constructing Feature Vectors

Some of the most popular methods for solving supervised learning tasks, such as (generalized) linear models, k-nearest neighbors, kernel methods, etc., exploit a feature-based representation of the input data to deal with the prediction task. The typical computation that needs to be carried out by the above methods is the inner product between a weight vector $\mathbf{w}$ and a feature vector $\phi(x)$ for an input x, or between two feature vectors $\phi(x)$ and $\phi(y)$ (e.g., for computing a logit in logistic regression, a distance for a given 2-norm, or a polynomial kernel). In the spirit of [24], our goal is to provide a generic methodology for constructing such feature vectors from the semantic information that we extract from the knowledge base for a given entity.

Let $\mathcal{Y}$ be a set of target labels (i.e., categories). In our learning scenario we assume that the supervision is given by a training sample $S \subseteq \mathcal{E} \times \mathcal{Y}$. The goal is to find a mapping $\phi : \mathcal{E} \to \mathcal{F}$ from entities to feature vectors, so that traditional machine learning methods, such as the above, can be applied to the learning problem at hand.

Let $e \in \mathcal{E}$ be an entity that was recognized in the underlying corpus $\mathcal{C}$. Let $q(e)$ be an ESPARQL query for extracting semantic features of e. Let us further assume that $q(e)$ has k variables in the `SELECT` clause. Note that every variable of $q(e)$ takes values from $\mathcal{E}$ (where, as described in Section 3, $\mathcal{R} \subset \mathcal{E}$). Furthermore, the substitution of each variable in the `SELECT` clause is going to represent a semantic feature for e. Hence, q can be viewed as a mapping

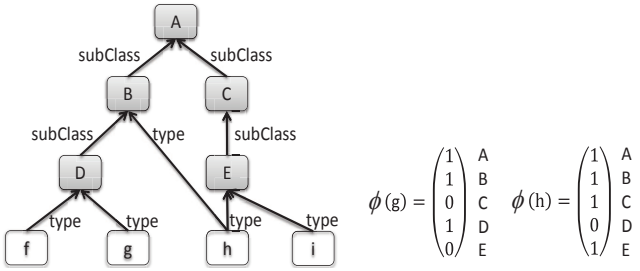

Figure 2: Example of a taxonomic knowledge graph with individuals `f`, `g`, `h`, `i` and types `A`, `B`, `C`, `D`, `E`. The feature vectors of `g` and `h` are shown on the left-hand side. Assuming that `g` and `h` are the entities from the training sample, the set $\mathcal{F}_S$ is given by the union of all their superclasses (i.e., {`A`, `B`, `C`, `D`, `E`}), as returned by the above query. A feature vector ϕ has a dimension for each element from $\mathcal{F}_S$. For each individual we set only those dimensions to 1 that represent superclasses of that individual.

$\mu : \mathcal{E} \to 2^{\mathcal{E}^k}$. The elements of $\mathcal{E}^k$ index the dimensions of the feature space $\mathcal{F} = \{0,1\}^{\mathcal{E}^k}$. Although the cardinality of $\mathcal{F}$ can be extremely large, in practice the features are derived from the training sample, which typically leads to a much smaller feature space. Hence, the effective feature space is given by:

$$\mathcal{F}_S := \{\mu(e) | (e, \cdot) \in S\}$$

Also note that often the domain of the knowledge base from which features are extracted can be restricted to the types of entities that are to be classified. For example if we are to classify movies, we would also like to extract semantic information from the knowledge base that is relevant and specific to movies. In summary the steps that are carried out by our framework in order to construct the feature vectors are the following:

1. Based on the training corpus $\mathcal{C}$ and the knowledge base $\mathcal{K}$ choose appropriate ESPARQL queries to extract semantic information about the entities of $\mathcal{C}$; the queries can be run on $\mathcal{K}$ or a restricted subset of $\mathcal{K}$ (see Section 3.2.2).
2. Unify the answer sets of the queries (for all entities from $\mathcal{C}$) to construct the set of semantic features that indexes the dimensions of the effective feature space $\mathcal{F}_S$.
3. The feature vector of an entity (which has the dimensionality of $\mathcal{F}_S$) can be built in the obvious way, by setting those dimensions which correspond to query answers for that entity to 1.

In the following, we show by means of examples how such feature vectors can be constructed and handled in practice.

3.2.1 Hypernymy-Based Feature Vectors

We start out by describing the simple case of constructing taxonomic (i.e., hypernymy-based) features. Suppose we have a knowledge base of individuals and all the classes they belong to. A sample knowledge graph is depicted in Figure 2. In order to extract all the hypernymy-based features for the entity `h`, we can run the following ESPARQL query q(`h`):

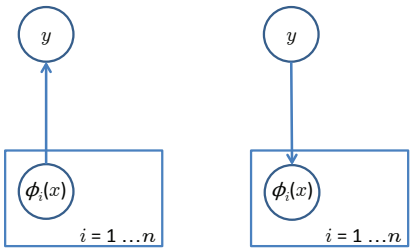

Figure 3: Difference between discriminative (left) and Naive Bayes model (right). The boxes represent n-fold replication of the feature values $\phi_i(x)$ for a given examples (x, y). Naive Bayes assumes independence of feature values given the label and hence leads to independent updates of weights, disregarding dependencies resulting from the ontological feature construction. This problem is avoided by discriminative models.

```
SELECT ?c
WHERE { h  type subClass*  ?c }
```

In Figure 2 we show how to construct hypernymy-based feature vectors for given entities. It is important to note that although these kinds of hypernymy-based vectors can be used by any of the mentioned supervised learning methods, they are best suited for methods that take the dependencies between the features into account. For instance, the Naive Bayes method does not consider these correlations and is thus expected to deliver a degraded performance. This is also shown empirically in our experimental section. It is relatively straight-forward to see that the subsumption of classes invalidates naive independence assumptions on the corresponding features. For example, on the taxonomy of Figure 2, a conditional independence assumption on the features $\phi_{\texttt{A}}$ and $\phi_{\texttt{B}}$ corresponding to the classes `A` and `B`, given a target label $y \in \mathcal{Y}$, would lead to $P(\phi_{\texttt{A}}, \phi_{\texttt{B}}|y) = P(\phi_{\texttt{B}}|\phi_{\texttt{A}}, y)P(\phi_{\texttt{A}}|y) = P(\phi_{\texttt{B}}|y)P(\phi_{\texttt{A}}|y)$. However $P(\phi_{\texttt{B}} = 1|\phi_{\texttt{A}} = 1, y) = 1$, hence $P(\phi_{\texttt{B}}|\phi_{\texttt{A}}, y) = P(\phi_{\texttt{B}}|y)$ if and only if $P(\phi_{\texttt{B}}|y) = 1$, which is in general not true. In Figure 3, we highlight the difference between discriminative models, which allow joint updates on weights, and Naive Bayes models whose conditional independence assumption on features leads to independent weight updates.

3.2.2 General Semantic Feature Vectors

Now consider the sample knowledge graph depicted in Figure 1. Suppose that the movies occurring there (i.e., Nikita, Léon, Black Swan) are movies from our training sample. Since in practice these kinds of knowledge bases can be very large, one can restrict the effective feature space by considering only types of entities that are relevant for the classification task at hand. A key concept for achieving this is given by the definition of a *restricted entity domain*.

DEFINITION 3.4. **Restricted Entity Domain.** *Let $T \subseteq \mathcal{E}$ be a set of types from a knowledge base $\mathcal{K} = (\mathcal{G}, \mathcal{E}, \mathcal{R}, \mathcal{Q})$. For a class $c \in T$, let D_c denote the set of entities that can be substituted for the variable* `?x` *in the following ESPARQL query on $\mathcal{K}$:*

```
SELECT ?x
WHERE { ?x  subClass|(type subClass*)  c }
```

The restricted entity domain *$\mathcal{D}_T$ for T is defined as $\mathcal{D}_T = \bigcup_{c \in T} D_c$.*

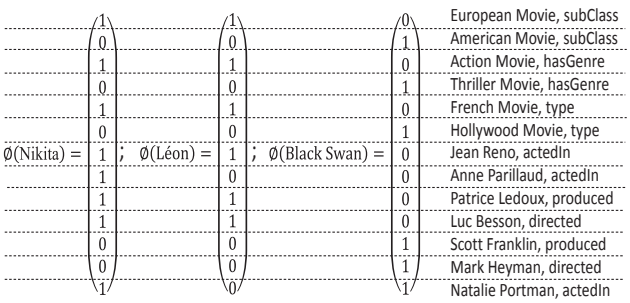

Figure 4: The elements in the union of the semantic neighborhood information (i.e., neighboring entities and relations) of the movies Léon, Nikita, and Black Swan index the dimensions of the effective feature space $\mathcal{F}_S$. Note that the higher semantic similarity between the movies Léon and Nikita is also reflected in the similarity between their semantic feature vectors.

For example, for the knowledge graph of Figure 1, we restrict the domain of entities to the types T={`Movie, Actor, Producer, Director`}. We then denote by $\mathcal{D}_T$ the *restricted entity domain* of all the subclasses and individuals of the classes in T.

With the restricted entity domain $\mathcal{D}_T$ from the above example, one possible way to construct the set $\mathcal{F}_S$ is by running two types of queries (1) a query to extract hypernymy-based features, similarly to above, and (2) a query to extract features about the semantic neighborhood of that entity, e.g.,

```
SELECT ?e ?r
WHERE { (m   ?r  ?e) UNION (?e  ?r  m) }
```

The results to these queries are then unified to construct feature vectors as depicted in Figure 4.

In practice, this kind of feature vectors will typically be very sparse. For example, if we use movie genres to classify movies, it is reasonable to expect every movie to belong to only a few from a long list of genres. Hence, for computational efficiency these vectors are implemented as sparse vectors. This way the complexity of the inner product is linear in the minimum number of dimensions set to 1 in one of the two corresponding vectors.

4. EXPERIMENTAL EVALUATION

In this section we evaluate the effectiveness of semantic features generated through the presented framework. For our experiments we use YAGO [19], a general-purpose knowledge base that was automatically constructed from Wikipedia, WordNet and other semi-structured Web sources. YAGO has been successfully used in many knowledge-oriented systems, such as the YAGO-NAGA project [21, 16], and most prominently in IBM's Watson [22], a system for answering Jeopardy questions.

In what follows, we empirically investigate two different learning tasks, recommendation and text classification. We first introduce the learning model that we have used and then describe the learning tasks and the datasets. The experimental results verify the hypothesis that semantic features are beneficial to predictive accuracy.

4.1 The Learning Model

As described in the previous section, each input instance (i.e., entity) x is formally represented by a sparse binary vector $\phi(x) = (\phi_1(x), \ldots, \phi_n(x))^T \in \mathcal{F}_S$.

In our model, the importance of the features in the vector $\phi(x)$ is represented by a weight vector $\mathbf{w} = (w_1, \ldots, w_n)$, where the component w_i reflects the importance of the feature $\phi_i(x)$. The model we use is a generalized linear Bayesian probit model, similar to the ones used in Matchbox [23] or AdPredictor [25]. This model computes a belief on the weights w_i, which is given by a Gaussian distribution with mean μ_i and variance σ_i^2. Hence, the state of the model is uniquely identified by vectors of means and variances of the weights, namely $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ and $\boldsymbol{\sigma}^2 = (\sigma_1^2 \ldots \sigma_n^2)$. If we knew the real values of the weights w_i, we could compute a score $s(x) = \sum_{i=1}^n w_i \phi_i(x)$, which in turn could be used to derive the probability of a target label $y \in \mathcal{Y}$ given the score $s(x)$, i.e., $P(y|s(x))$. The probability of a weight is given by:

$$p(\mathbf{w}) = \prod_i \mathcal{N}(w_i; \mu_i, \sigma_i^2)$$

Since in our model the weights w_i are represented as Gaussian variables, the score $s(x)$ is represented as a Gaussian random variable as well:

$$p(s|\mathbf{w}, x) = \mathcal{N}\left(s; \sum_{i=1}^n \phi_i(x)\mu_i, \sum_{i=1}^n \phi_i(x)\sigma_i^2\right)$$

Finally, one can model the probability of label y as a probit function:

$$P(y \mid s) = \Phi\left(\frac{ys}{\beta}\right)$$

with noise variance β^2, where $\Phi(t) = \int_{-\infty}^t \mathcal{N}(v; 0, 1)\, dv$. For the update equations in such a model we refer the reader to [25].

Such a model has several advantages:

1. By jointly updating the beliefs on the weights of the features, it can capture and exploit dependencies between features. As discussed in Section 3.2.1, this is expected to yield a better performance than models that treat parameters independently (e.g., Naive Bayes) in the presence of hierarchical hypernymy-based features. This aspect is empirically demonstrated in our experiments.
2. As shown by [23, 25], the model can deal with large sparse vectors. Furthermore, the learning can be carried out in an incremental fashion, as in an online setting, which is important for practical concerns.
3. Finally, the above model can handle different feedback types, including absolute, binary, and ordinal [23]. This flexibility allows us to apply it to various learning tasks.

4.2 Test Case: Movie Feedback Prediction

4.2.1 The Task

The task consists in predicting the feedback that a user will give on a movie. We use the generalized linear probit model from above with user and movie IDs as basic features, and add semantic features on top. Note that this task is simpler than state-of-the-art (i.e., collaborative-filtering-style) recommendation, where based on a user-movie feedback matrix, user and movie affinities are estimated [26, 23].

These affinities can be used to estimate the probability of a movie being liked by a user. However, together with the probit model, the task we consider shifts the focus on the benefits of the different feature types.

The really interesting cases for this kind of feedback prediction are those for which there are very few users and movies in the training set. This is similar in spirit to recommendation scenarios in which the feedback of users on movies is very sparse, or in which there is no feedback at all. For example, usually users are interested in the most recent movies; for these, however, there are hardly any feedback labels, which makes the recommendation task very difficult. We refer to this problem as the cold-start problem.

With this in mind, in our experiments, we investigate the question whether the semantic features (for movies) generated by the proposed framework can improve the prediction accuracy.

4.2.2 The MovieLens Dataset

MovieLens is a commonly used movie recommendation dataset[5]. It contains 1,000,206 ratings of 3,952 movies by 6,040 users. Ratings are on an ordinal scale from 1 to 5. The dataset also provides some meta data including information about movies (e.g., cast, genre, etc.). As YAGO provides more comprehensive cast, genre and type information for movies, and to increase comparability, in our experiments, we neglect the meta data provided by MovieLens. Instead we use our framework to construct semantic features from YAGO. This is done by exploiting the movie titles provided by MovieLens. For a given movie title, a great deal of information can be retrieved from YAGO, such as its budget, release date, cast, genres, box-office information, etc. However, for the sake of a clear message, we only select the features regarding a movie's cast and type. For example, for the movie Black Swan, the following query

```
SELECT ?x
WHERE { ?x  actedIn  Black_Swan }
```

returns entities such as *Natalie_Portman*, *Mila_Kunis*, *Vincent_Cassel*, etc. Similarly, the query

```
SELECT ?x
WHERE { Black_Swan   hasGenre|(type subClass*)  ?x }
```

returns types and genres such as `Drama_Movie`, `Thriller_Movie`, `Tragedy_Movie`, `Ballet_Movie`, `American_Movie`, etc.

4.2.3 Experimental Results

For this experiment we consider two settings: (1) no semantic features are used; instead the model learns the bias of a user towards a movie by using user and movie IDs as features, (2) in addition, for movies, the model uses semantic features extracted from YAGO.

As expected, the semantic features provide better descriptions of the movies and are valuable in the learning process, especially in the case of very sparse feedback. In order to measure the prediction accuracy in such cold-start situations, we adopt the methodology of [13] and [23]. We divide the movies into two sets according to the release dates: a training set containing 50% of the movies and a test set containing the rest with newer release dates. First the linear probit model is trained on all ratings for the movies in the training set by taking the users who have rated those movies into account. Then, for each movie in the test set, we train the model further on a random subset L of its ratings, for $L = 0\%$ to 50%. We use the model to predict the rest of the ratings for that movie. Finally we report the mean absolute error (MAE), $|\hat{r}_i - r_i|$, where r_i is the true rating and $\hat{r}_i$ is the predicted rating. The experiments are repeated 10 times and the average results, together with corresponding standard derivations are summarized in Figure 5.

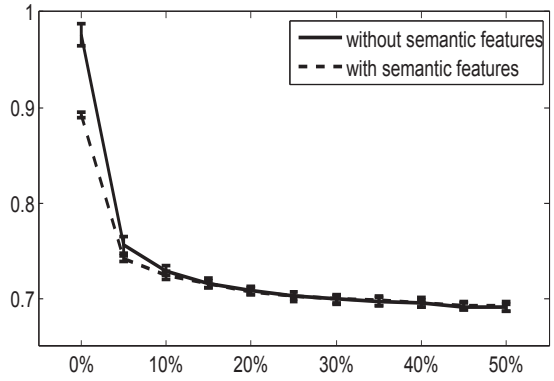

Figure 5: Mean absolute error (MAE) difference between two settings, with and without semantic features, for different values of L. It is important to note that these MAE results for semantic features are comparable to those of [26], which is a state-of-the-art recommendation system based on collaborative filtering. The MAE values of [26] range from 0.71 to 0.69, for $L = 5\%$ to $L = 75\%$ (see also [23]), whereas the values of the probit model with semantic features range from 0.74 to 0.69, for $L = 5\%$ to $L = 50\%$. This demonstrates the benefit of semantic features in combination with the linear probit model.

As expected, the improvement achieved by the semantic features is remarkable, especially when the ratings of a movie have not been largely revealed. For these cases the semantic features yield an improvement of almost 10% over the basic features. Such effectiveness, when dealing with the cold-start problem, is of particular interest for recommendation, as the service providers often experience the problem that users abandon their system and lose interest in returning due to the low quality of initial recommendations. We should also notice that the advantages of the semantic features start to become negligible when more and more ratings of a movie are observed. Finally, let us remark that the framework we propose has not reached its full potential yet. While there could be other meaningful semantic features for movie recommendation, we have only used the cast and type information. We hypothesize that semantic features for users could further improve the performance of the probit model. Furthermore, YAGO is rather a general-purpose knowledge base; better performance is expected when a more domain-specific knowledge base is used.

4.3 Tweet Classification

4.3.1 The Task

The classification task considered in this section is multi-label classification of tweets. More specifically, the goal is to assign the tweets to one or more of the nine

[5]http://www.grouplens.org/node/73

class	#instance	proportion
Business/Finance (BF)	2182	9.56%
Entertainment (E)	4235	18.56%
Lifestyle (LS)	4085	17.90%
Politics (P)	1199	5.25%
Science/Environment (SE)	789	3.45%
Sport (S)	1145	5.01%
Technology (T)	1880	8.23%
World Events (WE)	2122	9.30%
Other/Miscellaneous (OM)	12838	56.26%

Table 1: Statistics of the tweet classification dataset.

following categories: *Business/Finance (BF), Entertainment (E), Lifestyle (LS), Politics (P), Science/Environment (SE), Sport (S), Technology (T), World Events (WE), Other/Miscellaneous (OM).*

The problem with this task is that (since a tweet is restricted to 140 characters) tweets contain very few words, and many of those are often non-informative, especially when describing status updates. Moreover many tweets may contain words that have never been observed before (representing entities, such as new companies, products, upcoming celebrities, etc.). Semantic features for the entities recognized in the tweets can help mitigate such sparsity issues.

4.3.2 The Dataset

For a subset of tweets from the Twitter firehose, the categorizations were obtained using Amazon Mechanical Turk[6], and the results were manually verified by researchers from our lab. This dataset was collected and categorized for a personalized news service. For our experiments, a subset of the original dataset was selected, which contains 22,816 tweets with their class information. Table 2 shows three typical tweets from our training corpus. Some statistics of the dataset used in the experiments are shown in Table 1.

The sum of proportions is larger than 100%, since one tweet can simultaneously belong to more than one category. In this corpus the positive classes are often very sparse. Especially, the distribution of the classes is very unbalanced. For example in our dataset more than half of the instances belong to the category *Other/Miscellaneous*, while only less than 5% of them belong to *Science/Environment*. This is another reason (in addition to those presented in Section 4.3.1) why this tweet classification task is very challenging.

Conventional features in text classification are bag-of-words features, which can be extracted directly from the tweets. To obtain semantic features, we first apply an HMM-based entity recognition algorithm to locate terms which correspond to possible entities in a tweet. Then, the detected terms are mapped to the most likely entities in the YAGO knowledge base. This is done by means of a co-occurrence-based heuristics that exploits the YAGO *means* relation[7]. Once the entities have been recognized as YAGO entities, we use our framework to retrieve semantic features from YAGO. For example, by processing the first tweet in Table 2, *Bush* is recognized as an entity and is mapped by the above heuristics with high confidence to the YAGO entity George W. Bush. In this set of experiments we have used queries of the following kind:

[6]http://www.mturk.com/

[7]YAGO relates terms to an entity through the *means* relation if the terms refer to that entity.

	businessperson
	president
	person
governors_of_texas	politician
harvard_business_school_alumni	republican
harvard_university_alumni	scholar
phillips_academy_alumni	serviceman
presidents_of_the_united_states	skilled_worker
texas_republicans	worker
time_magazine_persons_of_year	corporate_executive
united_states_air_force_officers	executive
yale_university_alumni	governor
	leader
	military_officer
	administrator
	alumnus

Table 3: Some sample features obtained for the YAGO entity *George_W._Bush*. Left: neighboring entities; right: hypernyms.

```
SELECT  ?x
WHERE  { (George_W._Bush  ?r  ?x) UNION
         (?x  ?r  George_W._Bush) }
```

The above query retrieves all entities that are neighbors of George W. Bush from the knowledge base. For more general features, we make use of the YAGO type and subclass hierarchy and retrieve all hypernyms with the following query:

```
SELECT  ?x
WHERE  { George_W._Bush type subClass* ?x }
```

Some of the resulting semantic features are shown in Table 3.

4.3.3 Evaluated Learning Models

We evaluate the probit model on different feature settings for bag-of-words and semantic features; for the semantic features we look at the influence of hypernymy-based features. Because of their hierarchical and inclusive nature (e.g., a *president* is always a *person*), a naive independence assumption is in general invalid. To verify the hypothesis that the generalized linear probit model outperforms learning models with naive independence assumptions on these kinds of semantic features, we include the Naive Bayes model in our comparison. Finally, we compare the following five settings:

SH: probit model with bag-of-words and semantic features including hypernyms.

SnH: probit model with bag-of-words and semantic features excluding hypernyms.

BOW: probit model with only bag-of-word features.

NBSH: Naive Bayes with bag-of-words and semantic features including hypernyms.

NBBOW: Naive Bayes with only bag-of-word features.

The tests are carried out in a 10-fold cross validation. For each of the nine classes in the dataset, each learning model is trained and tested independently (often referred to as binary relevance by the multi-label learning community [12]). The

tweet	BF	E	LS	P	SE	S	T	WE	OM
Obama blames Bush for all of his misdeeds and then takes credit for the successful war in Iraq http://is.gd/e0iVM (via @PennyStarrDC)	0	0	0	1	0	0	0	1	0
A Modern Approach to an Ancient Game: The story behind The Path of Go http://bit.ly/g04KtA	0	0	0	0	1	0	1	0	0
@alex_ Good morning! How was the trip?	0	0	0	0	0	0	0	0	1

Table 2: Three typical training instances in tweet classification with associated labels.

predictions are evaluated with negative log-likelihood (NLL) and area under the ROC curve (AUC):

$$\mathrm{NLL} = -\left(y_i \log(h(x_i)) + (1-y_i)\log(1-h(x_i))\right),$$

$$\mathrm{AUC} = \frac{1}{|P|\,|N|}\sum_{x_i \in P}\sum_{x_j \in N}\omega(h(x_i) - h(x_j)),$$

where $y_i \in \{0,1\}$ is the true class, and $h(x_i) \in [0,1]$ is the the prediction of the method. P and N are the sets of positive and negative instances, respectively. $\omega(z)$ is defined as:

$$\omega(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0.5 & \text{if } z = 0, \\ 0 & \text{if } z < 0. \end{cases}$$

While NLL is a commonly used classification loss, AUC measures the learner's ranking performance. When we randomly choose a positive instance and a negative one, AUC corresponds to the probability that the method ranks the positive instance ahead of the negative one.

4.3.4 Results

The results for the above settings, based on the NLL scores and AUC, are shown in Table 4 and Table 5. In both tables, the Friedman test shows a significant difference among these settings at α-level 0.1 [14].

Comparing the different settings for the probit model, the semantic features improve upon the bag-of-words features. For this learning task, however, it seems that hypernymy-based features are not very informative. Note that since the hypernymy information in YAGO is largely based on WordNet, our observations are consistent with the those of [1].

The performance across the different classes varies considerably. SnH is able to boost the performance for most classes. But for some classes, such as Technology and Politics, it seems very hard to improve. The tweets related to these topics are often about the latest developments and trends, and the recognized entities are not well-covered in YAGO. The performance of the Naive Bayes method is poor. Since the predictions of the Naive Bayes method are not well-calibrated and tend toward extreme values, the method suffers considerably in terms of NLL score. Furthermore, the Naive Bayes method fails to handle numerous correlated features as we expected. NBSH gives the worst results in our test.

5. CONCLUSION

Our work aims at bridging the gap between machine learning and semantic technologies. Along these lines, we presented a modular framework for automatically constructing semantic features from structured knowledge. While these features can be used with various learning algorithms, their hierarchical dependencies can be best exploited by learning methods that can capture these dependencies. Our experiments on movie recommendation and tweet classification give evidence for the improvement of predictive accuracy, especially in cold-start situations, when the data is sparse. Further improvement is conceivable if more accurate and more domain-specific knowledge bases are used. Therefore, we believe that future advances in the area of information extraction and knowledge base construction will pave the way for techniques such as the ones proposed in this paper.

6. REFERENCES

[1] Scott, S., Matwin, S.: Text Classification Using WordNet Hypernyms. In: Use of WordNet in Natural Language Processing Systems: Proceedings of the Workshop, pp. 45–51. Association for Computational Linguistics (1998)

[2] Moschitti, A., Basili, R.: Complex Linguistic Features for Text Classification: A Comprehensive Study. In: 26th European Conference on Information Retrieval Research, LNCS 2997, pp. 181–196. Springer (2004)

[3] Voorhees, E.: Using WordNet to Disambiguate Word Senses for Text Retrieval. In: 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 171–180. ACM (1993)

[4] Degemmis, M., Lops, P., Semeraro, G.: Learning Semantic User Profiles from Text. In: Advanced Data Mining and Applications, LNCS 4093, pp. 661–672. Springer (2006)

[5] Bloehdorn, S., Hotho, A.: Boosting for Text Classification with Semantic Features. In: Advances in Web Mining and Web Usage Analysis, LNCS 3932, pp. 149–166. Springer (2006)

[6] Lu, Y., Peng, F., Mishne, G., Wei, X., Dumoulin, B.: Improving Web Search Relevance with Semantic Features. In: 2009 Conference on Empirical Methods in Natural Language Processing, Vol. 2, pp. 648–657. Association for Computational Linguistics (2009)

[7] Voorhees, E.,: Query Expansion Using Lexical-Semantic Relations. In: 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 61–69. Springer (1994)

[8] Kehagias, A., Petridis, V., Kaburlasos, V., Fragkou, P.: A Comparison of Word- and Sense-Based Text Categorization Using Several Classification Algorithms. Journal of Intelligent Information Systems, Vol. 21(3), pp. 227–247. Kluwer Academic Publishers (2003)

[9] Cohen, W.: Fast Effective Rule Induction. In: 12th

class	SH	SnH	BOW	NBSH	NBBOW
Business/Finance	0.1663±.0083(3)	**0.1569±.0073**(1)	0.1637±.0076(2)	2.3281±.0575(4)	2.3511±.0350(5)
Entertainment	0.2993±.0057(2)	**0.2816±.0134**(1)	0.3000±.0117(3)	2.4119±.0795(5)	1.1625±.0358(4)
Lifestyle	0.3499±.0145(2)	**0.3402±.0058**(1)	0.3609±.0154(3)	2.3222±.0790(5)	1.2885±.0314(4)
Politics	0.1074±.0047(3)	0.1018±.0053(2)	**0.0990±.0042**(1)	5.1388±.0453(5)	3.7856±.0799(4)
Science/Environment	0.0935±.0076(2)	**0.0920±.0090(1)**	0.0938±.0069(3)	6.5148±.0654(5)	5.1472±.0691(4)
Sport	**0.0839±.0109**(1)	0.0874±.0087(3)	0.0862±.0029(2)	5.7259±.0407(5)	3.9350±.0608(4)
Technology	0.1311±.0068(3)	0.1272±.0084(2)	**0.1255±.0116**(1)	3.0432±.0728(5)	2.3635±.0588(4)
World Events	0.1752±.0056(2)	**0.1693±.0111**(1)	0.1783±.0078(3)	3.2537±.1058(5)	2.4399±.0555(4)
Other/Miscellaneous	0.4101±.0106(2)	**0.4054±.0073**(1)	0.4316±.0083(3)	1.0783±.0560(5)	0.8987±.0333(4)
average rank	2.22	1.44	2.33	4.89	4.11

Table 4: Comparison based on negative log-likelihood. Numbers in parentheses represent the rank of the methods in the corresponding class.

class	SH	SnH	BOW	NBSH	NBBOW
Business/Finance	.9238±.0101(3)	**.9395±.0090**(1)	.9335±.0094(2)	.8289±.0115(5)	.8947±.0133(4)
Entertainment	.8987±.0047(2)	**.9124±.0077**(1)	.8973±.0085(3)	.7984±.0150(5)	.8607±.0114(4)
Lifestyle	.8393±.0147(2)	**.8534±.0068**(1)	.8370±.0112(3)	.7554±.0140(5)	.7927±.0107(4)
Politics	.9464±.0129(3)	.9567±.0056(2)	**.9592±.0085**(1)	.8114±.0105(5)	.9004±.0133(4)
Science/Environment	.9051±.0266(3)	**.9251±.0136**(1)	.9153±.0137(2)	.7372±.0156(5)	.8239±.0178(4)
Sport	.9595±.0154(2)	**.9616±.0091**(1)	.9592±.0115(3)	.7869±.0214(5)	.8784±.0160(4)
Technology	.9484±.0122(3)	.9560±.0043(2)	**.9571±.0092**(1)	.8385±.0143(5)	.9138±.0107(4)
World Events	.9167±.0037(3)	**.9263±.0061**(1)	.9170±.0102(2)	.8226±.0138(5)	.8762±.0161(4)
Other/Miscellaneous	.8909±.0072(2)	**.8955±.0055**(1)	.8815±.0064(3)	.8458±.0110(5)	.8798±.0064(4)
average rank	2.56	1.22	2.22	5.00	4.00

Table 5: Comparison based on area under the ROC curve. Numbers in parentheses represent the rank of methods in the corresponding class.

International Conference on Machine Learning, pp. 115–123. Morgan Kaufmann (1995)

[10] McGuinness, D.: Question Answering on the Semantic Web. IEEE Intelligent Systems, Vol. 19(1), pp. 82–85. IEEE Computer Society (2004)

[11] Maybury, M. (Ed.): New Directions in Question Answering. AAAI Press (2004)

[12] Cheng, W., Hüllermeier, E.: Combining Instance-Based Learning and Logistic Regression for Multi-Label Classification. Machine Learning, Vol. 76, pp. 211–225. Springer (2009)

[13] Lam, X., Vu, T., Le, T., Duong, A.: Addressing Cold-Start Problem in Recommendation Systems. In: 2nd International Conference on Ubiquitous Information Management and Communication, pp. 208–211. ACM (2008)

[14] Demsar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research, Vol. 7, pp. 1–30. Microtome Publishing (2006)

[15] SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`

[16] The YAGO-NAGA Project: Harvesting, Searching, and Ranking Knowledge from the Web. `http://www.mpi-inf.mpg.de/yago-naga/`

[17] W3C SweoIG: The Linking Open Data Community Project. `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

[18] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia: A Nucleus for a Web of Open Data. In: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, pp. 722–735. Springer (2007)

[19] Suchanek, F. M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: 16th International World Wide Web Conference, pp. 697–706. ACM Press (2007)

[20] Kasneci, G., Suchanek, F. M., Ifrim, G., Ramanath, M., Weikum, G.: NAGA: Searching and Ranking Knowledge. In: 24th International Conference on Data Engineering, pp. 953–962. IEEE (2008)

[21] Kasneci, G., Ramanath, M., Suchanek, F. M., Weikum, G.: The YAGO-NAGA Approach to Knowledge Discovery. In: SIGMOD Record, Vol. 37(4), pp. 41–47. ACM (2008)

[22] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N., Welty, C.: Building Watson: An Overview of the DeepQA Project. In: AI Magazine, Vol. 31(3), pp. 59–79. AAAI (2010)

[23] Stern, D., Herbrich, R., Graepel, T.: Matchbox: Large Scale Bayesian Recommendations. In: 18th International World Wide Web Conference, pp. 111–120. ACM (2009)

[24] Haussler, D.: Convolution Kernels on Discrete Structures. Technical Report (UCS-CRL-99-10), University of California at Santa Cruz (1999)

[25] Graepel, T., Candela, J. Q., Borchert, T. Herbrich, R.: Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In: 27th International Conference on Machine Learning, pp. 13–20. Omnipress (2010)

[26] Lam, X. N., Vu, T., Le, T. D., Duong, A. D.: Addressing cold-start problem in recommendation systems. In: 2nd International Conference on Ubiquitous Information Management and Communication, pp. 208–211. ACM (2008)