Discovering Structure in the Universe of Attribute Names

Alon Halevy[‡] Natalya Noy[†] Sunita Sarawagi[†§*] Steven Euijong Whang[†] Xiao Yu[†]

[†]Google Research
{noy,swhang,yux}@google.com

[‡]Recruit Institute of Technology
alon@recruit.ai

[§]IIT Bombay
sunita@iitb.ac.in

ABSTRACT

Recently, search engines have invested significant effort to answering entity–attribute queries from structured data, but have focused mostly on queries for frequent attributes. In parallel, several research efforts have demonstrated that there is a long tail of attributes, often thousands per class of entities, that are of interest to users. Researchers are beginning to leverage these new collections of attributes to expand the ontologies that power search engines and to recognize entity–attribute queries. Because of the sheer number of potential attributes, such tasks require us to impose some structure on this long and heavy tail of attributes.

This paper introduces the problem of organizing the attributes by expressing the compositional structure of their names as a rule-based grammar. These rules offer a compact and rich semantic interpretation of multi-word attributes, while generalizing from the observed attributes to new unseen ones. The paper describes an unsupervised learning method to generate such a grammar automatically from a large set of attribute names. Experiments show that our method can discover a precise grammar over 100,000 attributes of COUNTRIES while providing a 40-fold compaction over the attribute names. Furthermore, our grammar enables us to increase the precision of attributes from 47% to more than 90% with only a minimal curation effort. Thus, our approach provides an efficient and scalable way to expand ontologies with attributes of user interest.

Keywords

attribute; grammar; ontology

1. INTRODUCTION

Attributes represent binary relationships between pairs of entities, or between an entity and a value. Attributes have long been a fundamental building block in any data modeling and query formalism. In recent years, search engines have realized that many of the queries that users pose ask for an attribute of an entity (e.g., liberia cocoa production), and have answered that need by building rich knowledge bases (KBs), such as the Google Knowledge Graph [29], Bing Satori,[1] and Yahoo's Knowledge Graph [3]. These KBs, albeit broad, cover only a small fraction of attributes, corresponding to queries that appear frequently in the query stream (e.g., Obama wife). For the less frequent queries (e.g., Palo Alto fire chief), search engines try to extract answers from content in Web text and to highlight the answer in Web results. However, without knowing that, for example, fire chief is a possible attribute of CITIES, we may not even be able to recognize this query as a fact-seeking query.

Recent work [26, 14, 16] has shown that there is a long and heavy tail of attributes that are of interest to users. For example, Gupta et al.[14] collected 100,000 attributes for the class COUNTRIES, and tens of thousands of attributes for many other classes (e.g., fire chief for CITIES). However, to make this long and heavy tail of attributes useful, we must discover its underlying structure.

Towards this end, we propose to represent the structure in extracted attribute names as a rule-based grammar. For example, for the class COUNTRIES we may find many attributes with the head word population, such as asian population and female latino population. The grammar we induce will represent these and similar attributes as rules of the form $Ethnicity population and $Gender $Ethnicity population. The rules are succinct, human-interpretable, and group together semantically related attributes, yielding several benefits. First, for curators attempting to add new attributes to an existing schema, the rules reduce the complexity of finding high-quality attributes and organizing them in a principled fashion. Second, for the search engine, the rules enable recognition of a much broader set of entity–attribute queries: For example, a rule such as $Gender $Ethnicity population enables the search engine to recognize new attributes, such as male swahili population or latino female population. Third, we discovered that these rules help distinguish between high-quality and low-quality attributes. Specifically, the rules tend to group together attributes of similar quality. Hence, by sampling the attributes covered by the rules, we can quickly find where the high quality attributes are.

Finding such rules is much more subtle than simply grouping attributes that share a head word. For example, the class US_PRESIDENTS has many attributes with head word name, but they fall into different subsets: names of fam-

*Work done while on leave from IIT Bombay at Google Research.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2882975 .

[1]http://searchengineland.com/library/bing/bing-satori

ily members (daughter name, mother name), names of pets (dog name), and names of position holders (vice president name, attorney general name). Hence, we must discover the more refined structure of the space of attribute names. We draw upon an automatically extracted IsA hierarchy of concepts [15, 36] to define a space of possible generalizations for our grammar. This space is huge and we have to select the right level of generalization without much supervision. Another challenge is noise in automatically generated attributes (e.g. brand name and house name for US_PRESIDENTS), and concept hierarchies (e.g. Dog IsA $Relative). In this paper we show how we combine these large and noisy sources to generate a precise and compact grammar.

This paper makes the following contributions.

- We introduce the problem of finding structure in the universe of attribute names as a rule-based grammar. Solutions to this problem are important for interpreting queries by search engines, and for building large-scale ontologies.
- We propose a grammar that interprets the structure of multi-word attributes as a head word with one or more modifiers that have the same parent in an IsA hierarchy. We present a linear-program–based formulation for learning the grammar by capturing the noise in the extracted attributes and concept hierarchy as soft signals. Our algorithm is completely unsupervised and infers negative training examples from occurrence frequency on the Web and word embedding similarity with other attributes.
- We demonstrate that the precision of our learned rules is 60% to 80%, which is significantly higher than competing approaches. Furthermore, we show that for large attributes collections (e.g. attributes of COUNTRIES), the top-100 rules explain 4,200 attributes, providing a factor of 42 reduction in the cognitive load of exploring large attribute sets.
- We show that the rules distinguish between mostly good attributes or mostly bad attributes. Put together with the high rule quality, this observation enabled us to set up an efficient pipeline for adding attributes to the schema of the Google Knowledge Graph.

2. PROBLEM DEFINITION

We focus on the problem of generating a grammar to organize a large set of attributes ($\mathcal{A}$). The attributes are extracted from query logs and Web text and are associated with a class of entities such as COUNTRIES, US_PRESIDENTS, and CARS. Our grammar is a set of rules that semantically encode groups of attributes using a concept hierarchy. A rule represents a multi-word attribute as a head word with zero or more modifiers that are hyponyms of a concept in the concept hierarchy. For example, we represent the attribute wife's name for class US_PRESIDENTS as head word name and modifier wife from the concept $Relative in some concept hierarchy. The same holds for attributes son's name, mother's name, and father's name of US_PRESIDENTS.

The rest of this section is organized as follows. We first characterize the attributes $\mathcal{A}$ on which we build the grammar. Then, we describe the IsA hierarchy $\mathcal{H}$ used to form the rules of the grammar. We then formally define our grammar and the challenges involved in learning it automatically.

The collection of attributes ($\mathcal{A}$): Several works [26, 16, 2, 14] have mined attributes from query streams and from Web text. We use a collection from Biperpedia [14] in this work. Biperpedia contains more than 380,000 unique attributes and over 24M class–attribute pairs (an attribute can apply to multiple classes such as COUNTRIES, CARS, and US_PRESIDENTS). Biperpedia attaches a score, i_a, to each attribute a within each class G. The score induces a ranking that roughly correlates with the confidence that the attribute belongs to the class. Thus, as we go further down in the ranking, the quality of the attributes degrades. However, we emphasize that even far down in the long tail we find high quality attributes. For example, for COUNTRIES, we find good attributes (e.g., aerospace experts, antitrust chief) close to the bottom of the ranked list. Our manual evaluation showed that attribute collection has 50% precision for the top 5,000 attributes of representative classes among which only 1% exist in Freebase [4] and 1% in DBpedia [1]. As we discuss later, the noise in the attribute collection introduces challenges to the problem we address in this paper. In this work, we consider only the attributes that have more than one word in their name (which is 90% of all attributes).

IsA Hierarchy ($\mathcal{H}$): In principle, we could have used any concept hierarchy $\mathcal{H}$, such as Freebase or WordNet that provides a set of concepts (e.g., $Component, $Relative), and a subsumption relationship between them (e.g., Tyre is a $Component). However, the concept hierarchy in Freebase is too coarse for our needs because most of the concepts are too general. Instead, in this work we use a concept hierarchy extracted from Web text using techniques such as Hearst Patterns (in the spirit of [36]). For example, the text "Countries such as China" indicates that China is an instance of Countries. The resulting IsA relations can be either subconcept–superconcept or instance–concept. We refer to both instances and subclasses in the IsA hierarchy as *hyponyms*. This collection contains 17M concepts and 493M hypernym–hyponym pairs. Naturally, the hierarchy is inherently noisy. The IsA hierarchy captures this uncertainty by associating a *notability score*, $n_{k,c}$ for any concept $c \in \mathcal{H}$ and each hyponym k of c. Notability is the product of the probability of concept given the hyponym and the probability of the hyponym given the concept [36].

The attribute grammar: Formally, let Head_1, Head_2,..., Head_B denote a set of head words in attribute names. We use $Attribute_i to denote attributes derived from the head word Head_i. Each rule in the grammar has one of the following four types:

$Attribute_i ::= Head_i
$Attribute_i ::= $Modifier_ij optional-words $Attribute_i
$Attribute_i ::= $Attribute_i optional-words $Modifier_ij
$Modifier_ij ::= IsA Hypernyms of $Modifier_ij

An example grammar snippet for the class SPORTS_CARS is:

$Attribute_price	::= price
$Attribute_price	::= $Market $Attribute_price
$Attribute_price	::= $Attribute_price in $Market
$Attribute_price	::= $Component $Attribute_price
$Market	::= Singapore | USA | Dubai | London | ...
$Component	::= battery | bumper | tyre | door | ...

In the above example, the head word is price and the modifiers are captured by non-terminals $Market and $Component that represent all hyponyms of the corresponding concept nodes in our input IsA hierarchy.

We can often interpret an attribute in multiple ways. For example, here is another grammar for SPORTS_CARS attributes:

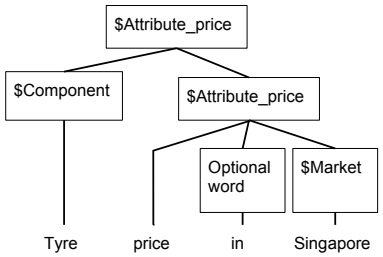

Figure 1: A derivation from the grammar for an attribute tyre price in Singapore **for class** SPORTS_CARS**.**

$Attribute_price	::= price
$Attribute_price	::= $Nation $Attribute_price
$Attribute_price	::= $Attribute_price in $Nation
$Attribute_price	::= $Product $Attribute_price
$Nation	::= Singapore | USA | UAE | UK | ...
$Product	::= battery | insurance | kit | door | ...

We associate a score with each rule to handle such inherent ambiguity. The score can be used to generate a ranked list of possible interpretations for any given attribute.

Use and advantages of our proposed organization.

There are several advantages to organizing attributes based on shared head words and concept nodes from an IsA hierarchy. First, such rules carry semantic meaning and are human interpretable, making them invaluable for curating attributes. For example, a rule such as $Crop production is more concise than a long list of attributes like coffee production, wheat production, rice production, and so on. Second, a correctly discovered set of rules will generalize to new or rare attributes, such as Quinoa production because our IsA hierarchy knows that quinoa is a crop. Such attributes are invaluable to a search engine trying to identify entity–attribute queries. Finally, the grammar provides the structure that exposes the patterns in the attributes, which is particularly useful for compound attributes with more than one modifier. For example, we defined our grammar recursively in terms of non-terminal $Attribute_price. This recursion allows us to correctly recognize a compound attribute like tyre price in Singapore as Figure 1 shows. Also, we can recognize variants like Singapore tyre price[2].

Such advantages do not accrue if we group attributes based only on a shared head word. A single rule like $Any price covers valid attributes like Rome price and Tyre price but also innumerable bogus strings like laugh price. Finding generalizable rules is non-trivial and we show how we combine several techniques from machine learning to discover them.

Challenges of rule selection The set of rules induced from a large set of attributes and a semi-automatically created IsA hierarchy can be bewildering both in terms of its size and the amount of noise that it contains. In our experiments, 100K attributes of COUNTRIES had 250K possible rules along our IsA hierarchy. Of these, fewer than 1% are likely good, but selecting the good rules is extremely challenging for several reasons. Consider attribute names ending with 'city' such as capital city, port city, and university city. There are 195 such attributes, and the IsA hierarchy $\mathcal{H}$ contains 267 concepts such as $Location, $Activity and $Device that generalize at least two city attributes. Because $\mathcal{H}$ contains only names of concepts, the match is syntactic and the attribute names will match a concept in $\mathcal{H}$ regardless of their semantics. Figure 2 presents a subset of the modifiers of these 195 attributes (top layer) and 267 concepts (bottom layer) with edges denoting the IsA relation. For example, the concept $Academic_institution contains modifiers college and university of attributes college city and university city, respectively. From these 267 concepts, we need to select a small set that generalizes most of the 195 attributes without introducing too many meaningless new attributes. A rule in the initial candidate set can be bad because of a variety of reasons, we list some below:

1. Wrong sense: Rules such as $Device city that generalize port city and gateway city are wrong because the sense of "port" in attribute port city and "gateway" in attribute gateway city is not the "device" sense of the term.
2. Too general: Rules such as $Activity city to cover party city, crime city, and business city are too general.
3. Too specific: Rules such as $Asian_country ambassador are too specific because a more general rule like $Country ambassador better captures attributes of COUNTRIES.
4. Wrong hyponyms: When IsA hierarchies are automatically created, they often also include wrong hyponyms in a concept. For example, "Florida" is a hyponym of $Country, and a "dog" is a hyponym of $Relative.

The rule selection problem is complicated further because we do not have a negative set of attributes that the grammar should reject. We cannot assume that we should reject anything not in $\mathcal{A}$ because $\mathcal{A}$ is only a partial list of the attributes that belong to the class. Even for the valid attributes, we have no human supervision on the choice of the head words and the choice of the concept node to serve as modifiers. For instance, we have no supervision of the form that a good rule for battery size is $Part size. In the next section, we address these challenges.

3. GRAMMAR GENERATION

We now present our method for learning the grammar rules over a set of attributes $\mathcal{A}$ given a concept hierarchy $\mathcal{H}$ (Section 2). Our first step is to use $\mathcal{A}$ and $\mathcal{H}$ to generate a set of candidate rules (Section 3.1). Next, we tackle the challenge of limited supervision by creating a set of new attributes to serve as negative examples. We depend on occurrence frequencies on the Web and similarity in an embedding space to infer these negatives (Section 3.2). Finally, we use a combined optimization algorithm to select a subset of rules to serve as a grammar for the attribute set $\mathcal{A}$ (Section 3.3). Figure 3 shows a flowchart for learning grammar rules, and Figure 4 presents an overview of our algorithm.

3.1 Candidate rule generation

Generating candidate rules proceeds in two steps: finding the head words and modifiers of attributes, and generalizing modifiers to concepts nodes from $\mathcal{H}$.

We rely on in-house Natural Language Parsing technology to identify the head words and modifiers in an attribute. For each attribute $a \in \mathcal{A}$, we generate its dependency parse [7], which is a directed graph whose vertices are labeled words and their part-of-speech, and the edges are syntactic relations between the words (Figure 5). The root word of the dependency parse is a head word for a rule. For example,

[2]An uncontrolled recursive application can also generate non-sensical attributes like tyre price in Singapore in Dubai. Any practical deployment will have to include constraints to disallow repeated application of the same rule, and limit the depth of the recursion.

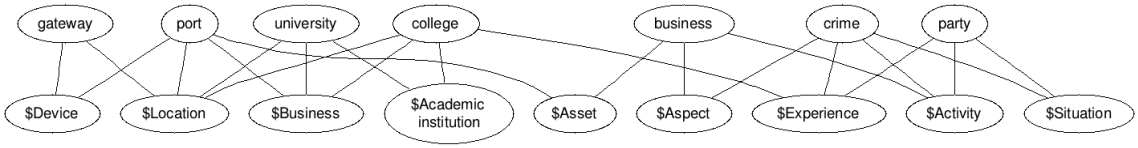

Figure 2: Modifiers of attributes with head word city (top-row), and the concept nodes in $\mathcal{C}$(bottom-row). $\mathcal{A}$ consists of each top-row node suffixed by head word city, e.g. gateway city, port city, etc. Candidate rules consist of concept nodes suffixed with city, e.g. $Device city, $Location city, etc. Most are bad.

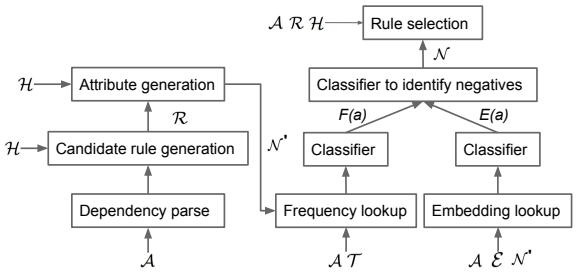

Figure 3: Grammar generation flowchart.

Inputs: Concept hierarchy $\mathcal{H}$, Attributes $\mathcal{A}$, Web corpus $\mathcal{T}$, Pretrained embeddings $\mathcal{E}$
Generating Candidate Rules $\mathcal{R}$ (Section 3.1)
Parse each $a \in \mathcal{A}$ to identify head words and modifiers
$\mathcal{R}$ = Generalize modifiers to concepts in $\mathcal{H}$ and form rules

Generating negatives $\mathcal{N}$ (Section 3.2)
$\mathcal{N}' = \cup_{r \in \mathcal{R}} \text{sample}(r)$ = Top attributes in $\text{gen}(r) - \mathcal{A}$
Get frequency $\#(a), \#(m_a)$ in $\mathcal{T}$ & find $F(a)$ $\forall a \in \mathcal{A} \cup \mathcal{N}'$
Get embeddings $\vec{m}_a$ from $\mathcal{E}$ & find $E(a)$ $\forall a \in \mathcal{A} \cup \mathcal{N}'$
Train models $\Pr(-|F(a)), \Pr(-|E(a))$
$\mathcal{N} = \{a \in \mathcal{N}' : (1 - \Pr(-|F(a)))(1 - \Pr(-|E(a))) < 0.5\}$

Rule scoring (Section 3.3)
Obtain soft signals $i_a, n_{a,r}, p_r$ from $\mathcal{H}$, $\mathcal{A}$, $\mathcal{N}$.
Solve linear program 4 using an LP-solver.
Return rules scores w_r for $r \in \mathcal{R}$.

Figure 4: Our grammar generation algorithm ARI.

size is the root word of the parse of the attribute average tank size. Each noun child of the root, concatenated with its descendants, forms a modifier. It can be tricky for the dependency parser to find the appropriate children of the root. In Figure 5, the parser correctly identified that average and tank are both modifiers of size in average tank size, while water tank is a single modifier of size in water tank size. For the attribute estimated average tank size, the dependency parse would have placed estimated as a child of average and therefore the modifiers would be estimated average and tank.

Next we create rules by generalizing the modifiers of attributes using concepts in the concept hierarchy $\mathcal{H}$. For example, the modifier tank can be generalized to (or, is hyponym of) concepts $Container, $Equipment and $Car component. Due to its ambiguity, tank can also be generalized to $Vehicle, $Weapon, and even $American singer. Each such concept forms a possible candidate rule. For example, tank size has the rules $Container size, $Equipment size, $Car_component size, $Vehicle size, $Weapon size and $American_singer size. For each attribute, we select the top-20 rules that generalize its modifiers with the highest notability scores (defined in Section 2). In the next steps, we consider all the rules that cover at least two attributes in $\mathcal{A}$.

3.2 Generation of Negatives

The candidate-generation step (Section 3.1) produces an enormous set of overlapping and noisy candidates. Our goal

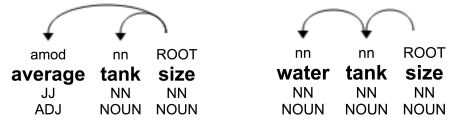

Figure 5: Dependency parses of two attributes.

is to select just the right subset of these rules to cover most of the given set of attributes $\mathcal{A}$ (i.e., positive examples), while not covering attributes that are not valid for the class. For this task, we need to identify strings that are not valid attributes (i.e., negative examples) and should not be covered by any selected rule. Because we do not have that supervision, we tap additional resources to infer such negatives.

For any rule $r \in \mathcal{R}$, let gen(r) denote the set of attributes that r generates. For example in Figure 2 a rule r of the form $Activity city of the COUNTRIES class can generate 0.9 million attributes corresponding to each hyponym of $Activity in $\mathcal{H}$. There three types of attributes in gen(r): (1) valid attributes of COUNTRIES that appear in $\mathcal{A}$, such as crime city, party city, business city; (2) valid attributes that do not appear in $\mathcal{A}$, such as art city; or (3) invalid attributes such as swimming city, yoga city, research city. We have supervision only for the first type but not for the second or third type. Because gen(r) is potentially large and the set of candidate rules $r \in \mathcal{R}$ is also large, we cannot afford to inspect every attribute in gen(r) to infer whether it is valid (in second group) or invalid (in third group). We therefore select a small subset (50 in our experiments) of gen(r) whose modifiers have the highest notability scores in r and do not appear in $\mathcal{A}$. We denote this set by sample(r). We will create a negative training set $\mathcal{N}$ from the union $\mathcal{N}'$ of sample(r) of all candidate rules r.

One option for $\mathcal{N}$ is to assume that all the attributes in $\mathcal{N}'$ are negative. Many text processing tasks that train statistical models only with positive examples use this strategy [30, 20]. However, we found good rules like $Crop production for COUNTRIES were unduly penalized by this blind strategy. In this example, $\mathcal{A}$ had a handful of such attributes like mango production and wheat production, and sample(r) had bean, vegetable, alfalfa, etc production which should not be treated as negative attributes. We therefore developed methods that can remove such likely positives from $\mathcal{N}'$.

We infer negatives from two signals—the occurrence frequency of attributes on the Web and the embedding similarity of attributes—and combine them via a novel training using only positive and unlabeled examples. We describe these features next and then present our training method.

3.2.1 *Relative frequency feature*

A strong signal for deciding if an attribute in sample(r) is valid comes from the frequency of occurrence of the attribute on the Web. Consider the candidate rule r=$Device city that wrongly generalizes attributes gateway city and port city. In this case sample(r) will contain attributes like sensor city and ipad city, which are meaningless strings and are perhaps not frequent on the Web. However, absolute frequency

of multi-word strings is less useful than relative measures of association like PMI and its variants [5, 35]. Even PMI and variants can be unreliable when used as a single measure in significance tests [6]. In our case, we have an additional signal in terms of $\mathcal{A}$ to serve as positive attributes. We use this signal to define a relative PMI feature. Let m_a, h_a denote the modifier and head word of attribute a. Let $\#(a), \#(m_a), \#(h_a)$ denote their respective frequencies on the Web. Using these frequencies, we calculate a relative PMI feature $F(a)$:

$$\log \frac{\#(a)}{\#(m_a)\#(h_a)} - \log \text{avg}_{b \in \mathcal{A}: h_b = h_a, b \neq a} \frac{\#(b)}{\#(m_b)\#(h_b)} \quad (1)$$

The first term in the equation is standard PMI. But the second term is a reference value calculated from the PMI of attributes in $\mathcal{A}$. This reference is the the average frequency ratio over the attributes in $\mathcal{A}$ that share a's head word[3]. One immediate advantage of this reference is that the relative PMI is independent of the frequency of the head word. This makes the relative PMI value more comparable across attributes with different head words—allowing us to use this as a feature of a single classifier across all head words.

3.2.2 Word embedding feature

The frequency feature is not useful for suppressing rules that cover frequent but irrelevant attributes. For example, the rule $Project cost for the CARS class was obtained by generalizing attributes production cost, maintenance cost, and repair cost in $\mathcal{A}$. The top few hyponyms of sample(r) are dam cost and highway cost. These attributes are frequent but not valid attributes of CARS.

A second feature quantifies the semantic similarity of other hyponyms of a concept with the hyponyms that occur as attribute modifiers in $\mathcal{A}$. For example, the concept $Project has hyponyms production, maintenance, and repair, which appear as modifiers of valid attributes in $\mathcal{A}$. Other hyponyms of $Project like dam and highway are further away from these three valid hyponyms than the three are from one another. We measure semantic similarity using word vectors trained using a Neural network [20]. These vectors embed words in an N-dimensional real space and are very useful in language tasks, including translation [19] and parsing [31]. We used pre-trained 500 dimensional word vectors[4] that put semantically related words close together in space.

We create an embedding feature for each attribute using these word vectors as follows. Let $\vec{m_a}$ denote the embedding of the modifier m_a of an attribute a. Let r be a rule that covers a. We define the embedding feature $E(a)$ for a with respect to a rule r that covers it as the cosine similarity between $\vec{m_a}$ and the average embedding vector of all positive attributes[5] covered by r.

$$E(a) = \text{cosine}(\vec{m_a}, \text{avg}_{b \in \mathcal{A}, r \in \text{rules}(b), a \neq b}(\vec{m_b})) \quad (2)$$

Example: Let $a =$ dam cost and let $r =$ $Project cost be a covering rule of a. The valid attributes r covers are production cost, maintenance cost, and repair cost. We first compute the average embedding vector $\vec{v}$ of production, maintenance, and repair. Then the embedding feature of a=dam cost is the cosine similarity with the embedding $\vec{m_a}$ of modifier $m_a =$ dam with $\vec{v}$, which is 0.2. In contrast, when $a =$repair cost, a positive attribute, we find the average vector of production and maintenance and measure the cosine similarity with the vector of repair to be 0.5.

It is useful to combine signals both from frequency and embedding, because frequency identifies wrong rules like $Device city and embedding identifies general rules like $Project cost. Embedding alone cannot eliminate a rule like $Device city because hyponyms of $Device such as "router", "ipad", "sensor" are semantically close to "port" and "gateway".

3.2.3 Training with positive instances

Now we train a classifier for classifying attributes in $\mathcal{N}' = \cup_{r \in \mathcal{R}} \text{sample}(r)$ as positive or negative using the frequency feature $F(.)$ (Eq 1) and embedding feature $E(.)$ (Eq 2). Attributes in $\mathcal{A}$ serve as positive labeled instances, we have no labeled negatives, only a large set $\mathcal{N}'$ to serve as unlabeled instances. This setting has been studied before [17, 8], but our problem has another special property that enables us to design a simpler trainer: both frequency and embedding features are monotonic with the probability of an instance being negative. We train a single feature logistic classifier separately on the frequency and embedding feature. For each feature, we find the p-percentile feature value among the positives.[6] We then train each classifier with all attributes in $\mathcal{A}$ as positive and the attributes in $\mathcal{N}'$ with feature value below this percentile as negative. This gives us two probability distributions $\Pr(-|E(a))$ and $\Pr(-|F(a))$. An instance $a \in \mathcal{N}'$ is negative if its negativity score $i_a =$

$$\Pr(-|F(a)) + \Pr(-|E(a)) - \Pr(-|F(a))\Pr(-|E(a)) \quad (3)$$

is more than half. The above formula is obtained by just assuming that the probability that an instance is positive is equal to the product of probability, $\Pr(+|F(a))\Pr(+|E(a))$. This has the effect of labeling an attribute as negative either if its frequency (PMI) is low relative to other positive attributes or its word embedding is far away from positive attributes. A summary of the process of generating negative attributes appears in Figure 4.

3.3 Rule selection

We are now ready to describe our rule selection algorithm, we call **ARI**. We cast this as a problem of assigning a score w_r to each candidate rule $r \in \mathcal{R}$ such that valid attributes ($\mathcal{A}$) are covered by correct rules and the number of invalid attributes $\mathcal{N}$ that are covered by these rules is minimized.

One important property of our algorithm is that it is cognizant of the noise in its input, viz., the attribute set $\mathcal{A}$, the negative attributes $\mathcal{N}$, and IsA hierarchy $\mathcal{H}$. The algorithm captures these as three soft signals as follows: Each attribute $a \in \mathcal{A}$ is associated with an importance score i_a to capture our uncertainty about a being a part of $\mathcal{A}$ (Section 2). Likewise for each negative attribute $a \in \mathcal{N}$ we have an importance score i_a (Eq 3). For each modifier m covered by a concept class of a rule r we take as input a notability score $n_{m,r}$ to capture noise in the concept hierarchy $\mathcal{H}$.

[3]During training when we measure the relative PMI for positive attribute a, we remove a from the reference set. This safeguards us from positive bias during training particularly when the reference set is small for rare head words.

[4]https://code.google.com/p/word2vec/

[5]Like for the frequency feature, when we measure the embedding feature of a positive attribute a during training we exclude a's embedding vector from the average.

[6]For our experiments, we used p=50 based on our prior that 50% of attributes are correct.

ARI puts together these various signals in a carefully designed linear program to calculate rule scores w_r via the following constrained optimization objective.

$$\min_{w_r \geq 0, \xi_a} \sum_{a \in \mathcal{A}} i_a \xi_a + \sum_{a \in \mathcal{N}} i_a \xi_a + \gamma \sum_{r \in \mathcal{R}} w_r p_r$$
$$s.t.$$
$$\xi_a \geq \max(0, 1 - \sum_{r \in \text{rules}(a)} n_{a,r} w_r), \quad \forall a \in \mathcal{A} \tag{4}$$
$$\xi_a \geq \max(0, \sum_{r \in \text{rules}(a)} n_{a,r} w_r), \quad \forall a \in \mathcal{N}$$

The above objective has three parts: the first part $\sum_{a \in \mathcal{A}} i_a \xi_a$ measures the error due to lack of coverage of the valid attributes $\mathcal{A}$, the second part $\sum_{a \in \mathcal{N}} i_a \xi_a$ measures the error due to wrongly including invalid attributes $\mathcal{N}$, and the third part $\sum_{r \in \mathcal{R}} w_r p_r$ is for penalizing overly general rules. The hyperparameter γ tunes the relative tradeoff between rules complexity and attribute coverage. We explain and justify each part. Our objective is influenced by the linear SVM objective for classifier learning but with several important differences in the details.

Error due to lack of coverage: The first part of the objective along with the first constraint requires that each attribute in $\mathcal{A}$ has a total weighted score that is positive and greater than one. Any deviation from that is captured by the error term ξ_a which we seek to minimize. This term uses two soft signals i_a and $n_{a,r}$ and we justify the specific form in which we used them.

1. We measure the total score of an attribute a as the sum of the scores w_r of rules that subsume a (denoted by rules(a)) weighted by the confidence $n_{a,r}$ of a being a hyponym of the concept class in r. This usage has the effect of discouraging rules that cover an attribute with low confidence because then the rule's score w_r will have to be large to make the overall score greater than 1 and the third term of the objective discourages large values of w_r. This encourages attributes to be covered by concepts for which it is a core hyponym.
2. The importance score of an attribute i_a is used to weigh the error of different attributes differently. This method of using i_a is akin to *slack scaling* in structured learning [34]. We also considered an alternative formulation based on *margin scaling* but for reasons similar to those discussed in [28], we found that alternative inferior.

Error for including invalid attributes: The second term requires that the scores of all invalid attributes be non-positive. Unlike in SVMs, we require the rule scores w_r to be positive because for our application negative scores provide poor interpretability; it is not too meaningful to use the feature weights to choose the single rule that provides the best interpretation. Because rule weights are non-negative, the score of all attributes will be non-negative. The third term thus penalizes any invalid attribute by the amount that its score goes above the ideal value of zero. Each such error term is scaled by i_a, the importance of the attribute in its role as a negative attribute as calculated in Equation 3.

Rule Penalty: In the third part of the objective $\sum_{r \in \mathcal{R}} w_r p_r$ we regularize each rule's score with a positive penalty term p_r to discourage overly general or too many rules. This is similar to a regularizer term in SVMs where a common practice is to penalize all w_r-s equally. Equal p_r values cannot distinguish among rules on their generality. A next natural step is to make the penalty proportional to the size of the concept class in r. However, a large concept class is not necessarily bad as long as all its hyponyms generate valid attributes. For example, a rule like $Crop production for COUNTRIES. We define penalty of a rule r as the average rank of the valid attributes in the concept class of r. We assume that each concept in $\mathcal{H}$ has its hyponyms sorted by decreasing membership score $n_{r,a}$ when calculating its rank. For example, the modifiers of CARS attributes like tyre size, brake size, and wheel size appear at an average rank of 23007 in concept $Product of rule $Product size but at average rank of 4 for the concept in rule $Vehicle_part size. This makes the penalty on rule $Product size 23007/4 more than the penalty on $Vehicle_part size. The intuition behind this penalty is that a rule where valid attributes appear much later in the sorting, are likely to include several undesirable attributes before it.

Our **ARI** objective is a Linear program and can be solved using any off-the-shelf library such as Clp[7]. In Section 4 we compare our algorithm with other alternatives and show that our final method does substantially better. We also analyze the importance of each soft signal in our objective.

4. EVALUATING RULE QUALITY

In this section we evaluate the quality of the rules that we generate. We evaluated on the attributes in the Biperpedia collection for four classes: COUNTRIES, US_PRESIDENTS, UNIVERSITIES, and SPORTS_CARS and an in-house created concept hierarchy as described in Section 2. Because we are not able to share this data, we also created a fifth dataset from publicly available sources. We considered the set of attributes in DBpedia [1] with noun head words and modifiers,[8] and we used WordNet as our concept hierarchy. Table 1 lists these classes and their number of attributes.

We start by showing some interesting rules that our algorithm generated in Table 1: For the class US_PRESIDENTS, the rule $Service policy covers attributes immigration policy, welfare policy; the rule $Relative name covers wife name, dad name, son name. Rules for COUNTRIES cover such attributes as adult illiteracy rate, and youth unemployment rate. The latter rules have two modifiers and are covered by the successive application of two rules: $Age_group rate and $Social_problem rate. Similarly for UNIVERSITIES, we encounter such attributes as emergency cell number and library phone number that are covered by two rules $Service number and $Mobile_device number. When using DBpedia with WordNet, the rule $Publicize date covers DBpedia attributes like air date and release date.

We now present a quantitative evaluation.

Ground Truth. To generate the ground truth for the rules, we needed to label rules manually. As Table 1 indicates, there are more than 300K candidate rules over the five at-

[7]https://projects.coin-or.org/Clp

[8]We could not find a class in DBpedia with more than a few hundred multi-word attributes with noun modifiers. Therefore, we took a union of attributes over all classes so that our precision-recall curves are statistically significant. Manual inspection of the rules showed that our selected rules rarely grouped unrelated attributes. This dataset was the largest publicly available attribute collection that we found.

Class	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{R}\rvert$	Rules Example
COUNTRIES	108K	236K	$Tax rate : {income tax rate, present vat rate, levy rate} $Age_group rate, $Social_problem rate : {adult illiteracy rate, youth unemployment rate}
UNIVERSITIES	18K	39K	$Cost fee: {registration fee, tuition fee, housing fee} $Service number, $Mobile_device number:{emergency cell number, library phone number}
US_PRESIDENTS	12K	17K	$Service policy: {immigration policy, welfare policy} $Relative name: {wife name, dad name, son name}
SPORTS_CARS	1.8K	2K	$Component size: {fuel tank size, trunk size, battery size} $Car_parts price: {bumper price, tyre price}, $Country price: {uk price, dubai price}
DBpedia	1.1K	0.5K	number of $Administrative_district: {number of city, number of canton} $Publicize date: {air date, release date}

Table 1: The five classes in our evaluation set. For each class, the second column ($|\mathcal{A}|$) is the number of attributes in the collection, the third column ($|\mathcal{R}|$) denotes the number of candidate rules, the third column contain example rules that we discover.

tribute sets. Because it is infeasible to evaluate manually such a large rule set, we selected a subset of rules that either appear in the top-500 by total attribute scores, or that cover the top-20 head words by attribute importance. This process produced roughly 4,500 rules to label for COUNTRIES and 1,400 for US_PRESIDENTS. For DBpedia, we evaluated all 500 rules. Three experts labeled each rule as *good* or *bad* and we selected the majority label as the label of the rule.

We note some statistics that highlight the difficulty of the rule selection problem: (1) good rules constitute only 8% of the total rules, so we face the typical challenges of finding a "needle in a haystack"; (2) there is significant disagreement among experts on whether a rule is good: experts disagreed on 22% of the rules.

Methods compared. We are not aware of any prior work on discovery of rules over attributes. To evaluate our proposed method **ARI**, we compare with methods used in other related problems. We describe two broad categories of methods: integer programming and classifier-based approach.

Integer programming approach: Our core rule selection method of Section 3.3 can be cast as a classical rule induction problem which seeks to cover as many positive instances while minimizing number of negative instances covered. Several[9] algorithms exist for this problem, including the one used in the Patty system [24, 23] that we discuss in Section 6. As a representative we choose a formulation based on integer programming (IP) because for practical problem sizes we can get an optimal solution. The IP formulation is as follows:

$$\min_{w_r \in \{0,1\}} \sum_{r \in \mathcal{R}} \frac{|\mathcal{N}_r|}{|\mathcal{N}_r| + |\mathcal{A}_r|} w_r + \gamma w_r$$
$$s.t. \sum_{r \in \text{rules}(a)} w_r > 0 \quad \forall a \in \mathcal{A} \tag{5}$$

In the above we use $|\mathcal{N}_r|$, $|\mathcal{A}_r|$ to denote the number of attributes of $\mathcal{N}$, $\mathcal{A}$ respectively subsumed by rule r. Thus, the first part of the objective measures the fraction of negative attributes covered by rule r. The second term is a constant per-rule penalty to not select too many rules. Like in our earlier approach, the γ is a tunable parameter to control the tradeoff between grammar size and penalty for covering invalid attributes. Thus, the IP above seeks to cover positive attributes with the minimum number of low error rules.

Most classical rule induction algorithms select rules in a greedy iterative manner. However, modern day computing power allows use of this more expensive, optimal IP. We used the SCIP[10] off-the-shelf library.

Classifier-based approach: The second approach is based on the view that the grammar is a binary classifier between valid and invalid attributes. This approach is popular in modern grammar learning tasks such as in [32, 30] that we discuss later in Section 6. The classifier-based method creates a labeled dataset by assigning a label $y_a = +1$ for each attribute $a \in \mathcal{A}$, and a label y_a = -1 for each $a \in \mathcal{N}$. The "features" for each instance are the set of rules that cover that instance. The goal then is to assign weights to the features so as to separate the positive from the negative instances. We used a linear SVM objective:

$$\min_{w_r} \sum_{a \in \mathcal{A} \cup \mathcal{N}} i_a \max(0, 1 - y_a \sum_{r \in \text{rules}(a)} w_r n_{a,r}) + \gamma \sum_{r \in \mathcal{R}} |w_r|$$

where the first term measures the mismatch between the true label y and the classifier assigned score s. The second term is a regularizer like in the previous two methods. This approach is different from ours in Equation 4 in two ways: first, we require $w_r \geq 0$ to get more interpretable scores, second, we assign different regularizer penalty to rules.

In addition to the above three methods, we used a baseline that chooses the rule whose concept has the highest notability score for the attribute's modifier. All hyper-parameters were selected via cross-validation.

Evaluation metric. Given the diversity of applications to which our grammar can be subjected, we present a multi-faceted evaluation of rules. For applications that use rules to explore large sets of attributes and possibly curate them in the schema of a structured knowledge base, it is important to generate "good" rules that compactly cover a large number of $\mathcal{A}$ attributes. For applications that use the grammar to parse (entity, attribute) queries, it is important to evaluate correctness at an attribute-level. Accordingly, we compare the methods along four metrics:

1. **Rule Precision:** the percent of generated rules that are judged good by our manual labelers.
2. **Rule Coverage:** the total number of attributes that are covered by rules judged good.
3. **Attribute Precision@1:** the percent of attributes whose highest scoring covering rule is judged good.
4. **Attribute Recall:** the percent of attributes covered by generated rules.[11]

[9] https://en.wikipedia.org/wiki/Rule_induction

[10] http://scip.zib.de/

[11] Some attributes might have no good rule covering it. We remove such attributes when calculating this metric.

Comparing methods. Figure 6 compares methods on rule-precision and rule-coverage metrics. For plotting this graph, for each method we select the rules for which the method assigns a positive score and order the rules by the total importance of $\mathcal{A}$ attributes they cover. Then, for increasing values of k, we plot on the y-axis the fraction of good rules out of k (rule precision) and on the x-axis the total number of attributes in $\mathcal{A}$ covered by the good rules (coverage). Each marker on the line is at a multiple of 50 rules except for the first marker at 10 rules (for DBpedia, a multiple of 2 rules starting from 8 rules). A method has high precision if its plot is high up along the y-axis and produces compact rules if it extends to the right with fewer markers. We show the results separately for COUNTRIES (left), US_PRESIDENTS (middle), and DBpedia (right). For example, the left plot says that for COUNTRIES the top-100 rules (the third circle from left) of **ARI** cover 4,200 attributes in $\mathcal{A}$, and 68% of the rules are judged good, as we go down to top-500 rules we cover 9300 attributes at a precision of 60%. For US_PRESIDENTS, the top-10 rules of **ARI** cover 100 attributes at a precision of 80% and the top-100 rules cover 300 attributes and 67% of them are good. For DBpedia, the top-18 rules of **ARI** cover 48 attributes at a precision of 67%. We highlight our observations about the different methods of rule selection:

1. Overall, the **ARI** method provides the best tradeoff between precision of selected rules and the compactness they provide. At the last marker in each plot (after 500 rules for COUNTRIES, 250 rules for US_PRESIDENTS, and 18 rules for DBpedia), the precision of **ARI** is highest, and although **Integer Programming** sometimes yields more coverage, its precision is unacceptably low.
2. The **ARI** and **Classifier** approach eventually get the same precision for US_PRESIDENTS, but for the top few hundred rules the **ARI** method has significantly higher precision. The main reason for the low precision of the **Classifier** approach for top-rules is that it does not penalize general rules that appear at the top when we sort rules by total importance of $\mathcal{A}$ attributes.
3. The compactness of rule sets for COUNTRIES is much higher than for US_PRESIDENTS: for roughly the same precision of 62%, the top-250 rules of **ARI** cover 7000 and 500 attributes, respectively. This observation indicates that countries have many more related attributes (e.g., economic indicators), whereas attributes of presidents tend to refer to less structured data (e.g., legacy, achievements).

We next show how accurately each method can interpret attributes by comparing them on the attribute precision and recall metrics. Figure 7 shows the precision and recall values of the four methods for the top-k most important labeled attributes (covered by rules) for increasing k separately for COUNTRIES and US_PRESIDENTS. The DBpedia setting is slightly different where all attributes are used without ranking, and the baseline method is not compared because WordNet does not have notability scores. For COUNTRIES, **ARI** dominates all other methods on precision and recall. For US_PRESIDENTS and DBpedia, **ARI** provides much higher precision than other methods that have high recall. The poor performance of the **Integer Programming** approach highlights the importance of considering the noise in our inputs $\mathcal{A}$ and $\mathcal{R}$. The simple baseline that independently selects for each attribute the rule with the highest notability score provides poor precision. Hence, it is important to make global decisions for a rule based on other positive and negative attributes it covers.

	Attribute-level			Rule-level (top-100)	
Method	**Pr@1**	**Re**	**F1**	**Pr**	**Coverage**
All features	45	56	50	68	4491
No importance score	38	58	46	56	4188
No membership score	40	60	48	55	6133
No per-rule penalty	34	55	42	42	5682
All Negatives	43	55	48	68	4644

Table 2: Impact of different features in ARI. Here Pr denotes precision and Re denotes recall.

These evaluations show that our proposed method provides the best precision-compactness tradeoffs whether viewed broadly at top-covering rules or microscopically at individual attributes and their best interpretation by our grammar.

Analysis of **ARI.** Our method of rule selection has a number of novel features in terms of how it handles attribute importance scores, handles noise in inputs, penalizes general rules, and generates negative attributes. In this section, we analyze the impact of each feature by reporting accuracy with that feature removed. In Table 2 the first row is the performance of the full-featured method and each of the subsequent rows has one feature removed as follows:

1. The second row is with the importance scores i_a removed — that is, by setting $i_a = 1$ in Equation 4. We observe that both attribute-level and rule-level accuracies drop. Attribute-level F1 drops from 50 to 46 mostly because of reduced precision. Rule-level precision drops by a larger margin from 68 to 56.
2. The third row shows accuracy with the concept membership scores $n_{a,r}$ hardened to 1. Attribute-level F1 drops from 50 to 48 and rule-level precision drops from 68 to 55. Coverage increases because by dropping membership scores more general concepts that cover many attributes are preferred but many of these are bad as reflected in the dropped precision.
3. The fourth row shows accuracy with the same penalty for all rules instead of being proportional to their rank. We observe that attribute-level F1 drops from 50 to 42 and rule-level precision drops from 68 to 42; indicating that rank-based rule penalty is perhaps the most important feature of **ARI**.
4. The fifth row shows accuracy where we label all attributes in sample(r) as negative instead of using our method based on low embedding similarity and low frequency (discussed in Section 3.2). We observe that the quality of interpretation drops from 50 to 48.

These experiments demonstrate that **ARI** is an effective method for rule selection and the careful inclusion of varied soft signals to handle input uncertainty and incomplete supervision has paid off.

5. APPLYING RULES TO FOCUS MANUAL CURATION OF ATTRIBUTES

In this section we demonstrate one application of our selected rules—their use in curating attributes for expanding knowledge bases. While the automatically extracted attributes are invaluable in capturing the interests of users,

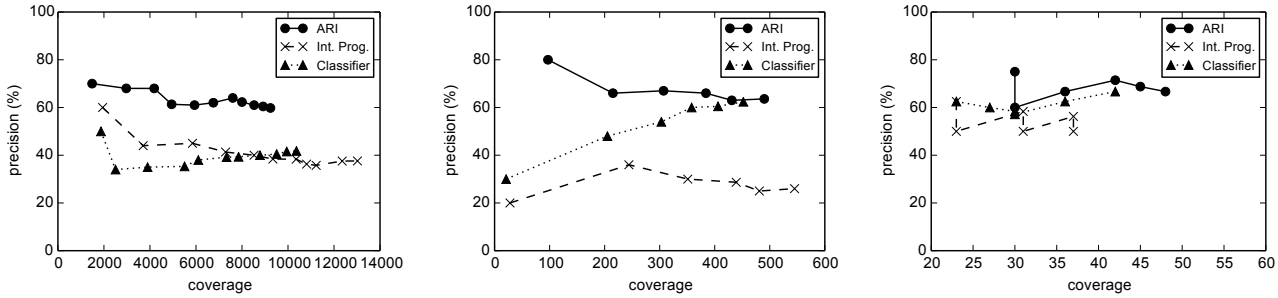

Figure 6: Rule Precision versus Coverage of top-500 rules of COUNTRIES (left), top-250 rules of US_PRESIDENTS (middle), and top-18 rules for DBpedia (right).

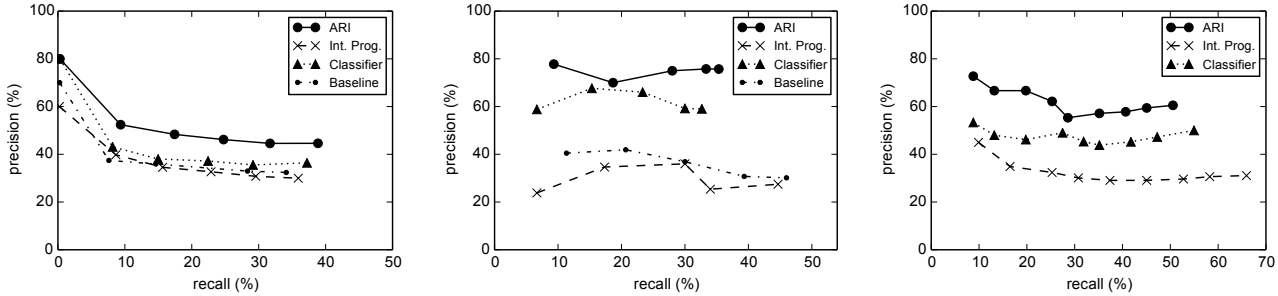

Figure 7: Attribute Precision versus Recall of 4135 attributes of COUNTRIES (left), 238 attributes of US_PRESIDENTS (middle), and 91 attributes of DBpedia (right).

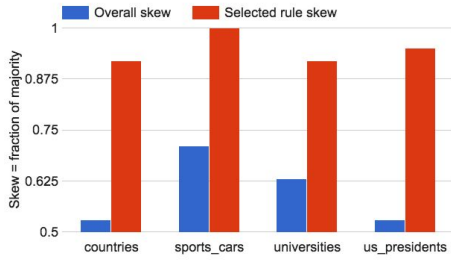

Figure 8: Overall skew and rule-wise skew for each collection. Skew=fraction of attributes in the majority class and is always between 0.5 and 1.

they do not meet the precision standards of a knowledge base and often require human curation. Because our rules discover semantically related attributes they hold great promise in reducing the cognitive burden of this curation. Furthermore, our curation mechanism exploits the following hypothesis: *The quality of attributes within most rules is highly skewed: Either the vast majority of the attributes covered by a rule will be judged by human evaluators as good attributes or the vast majority will be judged as bad.* For example, in SPORTS_CARS, the rule $Fastener pattern covers mostly good attributes including bolt pattern, lug nut pattern, and stud pattern while the rule $Automaker engine covers mostly bad attributes including bmw engine and ford engine. We evaluated the hypothesis using the following process: On each Biperpedia class in Table 1, we select the top-20 rules for which our algorithm assigns a positive score ordered by the total importance of $\mathcal{A}$ attributes they cover. We then get an expert to label up to 15 randomly selected attributes in each rule as either *good* or *bad* for the class. Using the labels we plot two kinds of skew in Figure 8:

1. overall skew: fraction of attributes covered by the majority label. For instance, for SPORTS_CARS, 71% of the labeled attributes were good, so the skew is $\max\{0.71, 1-0.29\} = 0.71$. For US_PRESIDENTS, 47% were good, so the skew is $\max\{0.47, 1-0.47\} = 0.53$.
2. rule-wise skew: measure skew within each rule and average. For instance, for SPORTS_CARS rule-wise skew is 1 since each rule had either all bad or all good attributes.

Figure 8 shows that for all collections rule-wise skew is much higher than overall skew. For COUNTRIES, for example, the skew of 0.53 went to 0.92 for the selected rules. In other words, for any given rule that our algorithm selected, on average, 92% of attributes covered by the rule had the same judgement ("good" or "bad"). Thus, these results confirm our hypothesis that the rules that we generate help us distinguish between clusters of "good" and "bad" attributes.

These results enable us to reduce dramatically the curation required to increase the precision of attributes: An expert (or, crowd) labels on only a small sample of attributes from a rule identify which rules are heavily skewed towards positive attributes. Once we detect such a rule with desired level of confidence, we can select all its attributes as positive. Because we have a large number of skewed rules, this process yields many good attributes at any desired precision-level. Figure 9 shows the number of good attributes that we gathered by increasing manually labeled attributes. For large collections like COUNTRIES we gathered 200 attributes after labeling just 5 and 500 after labeling just 90.

We would not get such behavior by rules that group on head words alone. Often, different rules with the same head have opposing skews. For example, US_PRESIDENTS has many attributes with head word name, such as mother name, dog name, hat name, brand name, and car name. Our rules segregate them into mostly good attributes ($Relative name, and $Pet name) and mostly bad attributes ($Place name). Also, not all candidate rules exhibit such skew. Our candidates set might include very general rules like $Person name

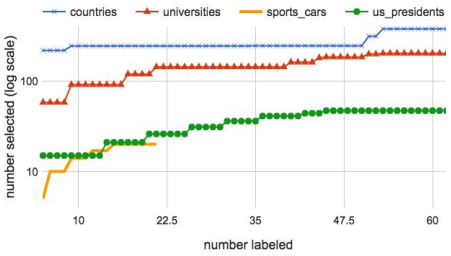

Figure 9: Number of positive attributes selected against number of attributes labeled. Precision of the selected set is more than 90% in all cases.

that cover semantically unrelated attributes (vice president name and mother name). Our rule selection algorithm is able to eliminate them by penalizing overly general rules.

6. RELATED WORK

Open-domain IE systems [9, 10, 18, 21] have automatically extracted large collections of textual relations (e.g., "starred in" and "was elected to") between entities. These textual relations can be viewed as attributes, but are typically in verb form rather than noun form. Recently, many efforts have attempted to extract attributes of classes from query streams and text [26, 16, 2, 14, 37, 27, 12, 25]. The focus of all these work is on extracting high quality attributes, and not on finding structure in the space of extracted attribute names, which is the focus of our work. One exception is the Patty system [24, 23], which generalizes a textual relation such as "Taylor effortlessly sang Blank Space" to a pattern such as "#Singer * sang #Song" and arranges the patterns in a taxonomy. The focus of Patty is on generalizing w.r.t. the subject and object of the relation, not on finding structure of the relation names themselves. The algorithm used for generalization in Patty is a specific instance of a rule-learning algorithm; and the Integer Programming (IP) -based approach we compared with in Section 4 is another. We chose to compare with the later because the IP formulation could be solved optimally for our problem sizes.

A tangentially related stream of work is on unsupervised grammar induction in the NLP literature [32, 30] where the goal is to learn semantic parses of a set of sentences. The important differences with our work is that in sentences are much longer than attributes, and require a more complex interpretation. In addition, all sentences are assumed to be correct, which is not true in our case. Negative examples are generated by perturbing the given sentences [30]. This method does not work for us since negatives generated by random word replacements are unlikely to help us discriminate between overlapping concept hierarchies. [11] presents another method of generating negatives based on a partial completeness assumption that applies when generating rules over multiple relations. Their method is not applicable to our setting and we are not aware of any prior work that generates negatives like we do by combining Web frequency and embedding similarity.

The work on noun compound understanding (e.g., [33]) attempts to parse descriptions of *sets* of objects (e.g., NATIVE AMERICAN AUTHORS). In contrast, our work focuses on understanding the structure of attribute names of entities. However, an important line of research is to investigate whether noun-phrase understanding can benefit from understanding attributes and vice versa.

Another related problem is linking extracted binary relations to a verb taxonomy like WordNet. For example this work tries to link the relation "played hockey for" to the "play1" verb synset and to link "played villain in" to the "act" verb synset[13]. The problem we address here is very different. Instead of linking individual attributes to a taxonomy, we introduce new rules to group related attributes using the taxonomy to provide hypernyms.

Mungall et al [22] used the regularities in the syntactic structure of class names in the Gene Ontology (GO) to generate formal definitions for classes. Like our work, they also relied on parsing complex names and then grouping them to create the rules. Because their approach needed to work for relatively small number of entities, they relied on heuristics rather than machine learning to generate the rules and the approach was heavily tailored to class names in GO.

7. CONCLUSION

This paper introduced the problem of finding structure in the universe of attribute names via rules comprising a head word and a concept node from a IsA hierarchy. Such rules offer a concise semantic representation of attributes and the ability to recognize new attributes names and variations of existing attributes. The rules can also be used to build high-quality ontologies at scale with minimal curation effort. The methods described in this paper are already in use for schema exploration and curation at Google.

Our rule learning algorithm takes as input two noisy inputs— class attributes extracted from query stream and text, and an IsA hierarchy also extracted from text— carefully models their noise in a constrained linear program to produce a set of high quality rules. Our algorithm is totally unsupervised and leverages web frequency and embedding vectors to automatically discover negative attributes.

We perform extensive experiments over four large attribute collections. Our experiments show that our rules have precision between 60% and 80% and compress attribute names by up to a factor of 42. We show that our selected rules are highly skewed in the quality of attributes they cover. This skew aids in significantly reducing the curation effort needed in adding attributes to a knowledge base.

Our work is the first step in discovering the structure of attribute names. This paper presented one method of organization based on rules. Another alternative we considered was to use clustering (using embedding vectors) to group related attributes like economy and currency that rules cannot. However, clustering tends to be noisy (mixing good and bad attributes together if they are semantically similar) and was not effective in surfacing structurally related attributes. Rules and clusters play complementary roles and in future, we would like to combine the two methods. Also, we would like to understand more deeply the semantics of the rules. For example, we discovered rules of the form $Metal production, where the rule modifier can be understood as a selection condition on a (logical) table that contains production of various medals. In contrast, other modifiers may simply be mathematical functions applied to an attribute (e.g., average unemployment rate). Attaining a deep understanding of variations in attribute naming is a major step towards building ontologies that capture the way users think about the world and providing better services for them.

8. REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[2] K. Bellare, P. P. Talukdar, G. Kumaran, F. Pereira, M. Liberman, A. McCallum, and M. Dredze. Lightly-supervised attribute extraction. *NIPS 2007 Workshop on Machine Learning for Web Search*, 2007.

[3] R. Blanco, B. B. Cambazoglu, P. Mika, and N. Torzec. Entity recommendations in web search. In *The 12th International Semantic Web Conference (ISWC 2013)*, pages 33–48. Springer, 2013.

[4] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[5] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[6] O. P. Damani and S. Ghonge. Appropriately incorporating statistical significance in PMI. In *EMNLP*, 2013.

[7] M.-C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure trees. In *LREC*, 2006.

[8] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *SIGKDD*, 2008.

[9] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, 2008.

[10] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545, 2011.

[11] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

[12] R. Ghani, K. Probst, Y. L. 0002, M. Krema, and A. E. Fano. Text mining for product attribute extraction. *SIGKDD Explorations*, 8(1):41–48, 2006.

[13] A. Grycner and G. Weikum. HARPY: hypernyms and alignment of relational paraphrases. In *COLING 2014*, pages 2195–2204, 2014.

[14] R. Gupta, A. Y. Halevy, X. Wang, S. E. Whang, and F. Wu. Biperpedia: An ontology for search applications. *PVLDB*, 7(7):505–516, 2014.

[15] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[16] T. Lee, Z. Wang, H. Wang, and S.-W. Hwang. Attribute extraction and scoring: A probabilistic approach. In *ICDE*, pages 194–205, 2013.

[17] W. S. Lee and B. Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, 2003.

[18] Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni. Open language learning for information extraction. In *EMNLP-CoNLL*, pages 523–534, 2012.

[19] T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[21] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *AAAI-15*, 2015.

[22] C. J. Mungall. Obol: Integrating language and meaning in bio-ontologies. *Comparative and Functional Genomics, vol. 5, no. 6-7, pp. 509-520, 2004. doi:10.1002/cfg.435*, 5(6-7):509–520, 2004.

[23] N. Nakashole, G. Weikum, and F. Suchanek. Discovering semantic relations from the web and organizing them with patty. *SIGMOD Rec.*, 42(2), July 2013.

[24] N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*, pages 1135–1145, 2012.

[25] M. Pasca. Turning web text and search queries into factual knowledge: Hierarchical class attribute extraction. In *AAAI*, 2008.

[26] M. Pasca and B. Van Durme. What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI*, volume 7, pages 2832–2837, 2007.

[27] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *hltemnlp2005*, pages 339–346, Vancouver, Canada, 2005.

[28] S. Sarawagi and R. Gupta. Accurate max-margin training for structured output spaces. In *ICML*, 2008.

[29] A. Singhal. Introducing the knowledge graph: things, not strings. *Official Google Blog, May*, 2012.

[30] N. A. Smith and J. Eisner. Guiding unsupervised grammar induction using contrastive estimation. In *In Proc. of IJCAI Workshop on Grammatical Inference Applications*, pages 73–82, 2005.

[31] R. Socher and C. D. Manning. Deep learning for NLP (without magic). In *Tutorial at NAACL*, 2013.

[32] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *HLT-NAACL*. The Association for Computational Linguistics, 2010.

[33] S. Tratz and E. H. Hovy. A taxonomy, dataset, and classifier for automatic noun compound interpretation. In *ACL*, 2010.

[34] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[35] P. D. Turney, P. Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

[36] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: a probabilistic taxonomy for text understanding. In *SIGMOD*. ACM, 2012.

[37] M. Yahya, S. Whang, R. Gupta, and A. Y. Halevy. Renoun: Fact extraction for nominal attributes. In *EMNLP*, pages 325–335, 2014.