Hierarchical Semi-supervised Classification with Incomplete Class Hierarchies

Bhavana Dalvi
Carnegie Mellon University
Pittsburgh, PA 15213
bbd@cs.cmu.edu

Aditya Mishra
University of Massachusetts
Amherst, MA 01003
adityam@cs.umass.edu

William W. Cohen
Carnegie Mellon University
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

ABSTRACT

In an entity classification task, topic or concept hierarchies are often incomplete. Previous work by Dalvi et al. [12] has showed that in non-hierarchical semi-supervised classification tasks, the presence of such unanticipated classes can cause semantic drift for seeded classes. The Exploratory learning [12] method was proposed to solve this problem; however it is limited to the flat classification task. This paper builds such exploratory learning methods for hierarchical classification tasks.

We experimented with subsets of the NELL [8] ontology and text, and HTML table datasets derived from the ClueWeb09 corpus. Our method (OptDAC-ExploreEM) outperforms the existing Exploratory EM method, and its naive extension (DAC-ExploreEM), in terms of seed class F1 on average by 10% and 7% respectively.

1. INTRODUCTION

A common way to organize information is by classification into a concept or topic hierarchy. For example, the Open Directory Project and the Yahoo! Directory are examples of topic hierarchies developed to organize pages on the Web, and Wordnet, NELL and Freebase are examples of large knowledge bases that organize entities into concept hierarchies. However, in an open-domain task, hierarchies are often incomplete, in the sense that there is meaningful structure in the data not captured by the existing hierarchy. E.g., consider a hierarchical entity classification task, with class hierarchy 'onto-1' shown in Figure 1 (left). There are 2 types of locations defined in it: 'State' and 'Country'. However, the unlabeled data about entities on the Web can include location entities of type 'City', 'Museum', etc., that are absent in this ontology, hence called 'unanticipated classes'.

Dalvi et al. [12] showed that in a non-hierarchical semi-supervised classification task, the presence of such unanticipated classes hidden in the unlabeled data can cause semantic drift for seeded classes. They also proposed an approach to solve this semantic drift problem by employing exploratory learning. This algorithm is an extension of the semi-supervised EM algorithm that can create new classes for datapoints that do not belong to the classes known upfront. It explores different numbers of classes while learning. In this paper we first summarize a naive extension of this approach (DAC-ExploreEM method), that applies exploratory learning in a top-down divide and conquer (DAC) manner. However, we notice that DAC-ExploreEM improves over flat ExploreEM in only some cases. Finally we present an optimized version of it (OptDAC-ExploreEM method [11]) that gives more significant improvements over the baseline.

Our proposed OptDAC-ExploreEM method traverses the class hierarchy in top-down fashion to detect whether and where to add a new class for the given datapoint. It also uses a systematic optimization strategy to find the best set of labels for a datapoint given ontological constraints in the form of subset and mutual exclusion constraints.

We demonstrate OptDAC-ExploreEM's effectiveness through extensive experiments on datasets constructed with subsets of the NELL ontology (shown in Figure 1) and text and semi-structured HTML table datasets derived from the ClueWeb09 corpus. In particular, OptDAC-ExploreEM improves seed class F1 on average by 13% when compared to its semi-supervised counterpart (OptDAC). It also outperforms both previously proposed exploratory learning approaches FLAT-ExploreEM and DAC-ExploreEM [11] in terms of seed class F1 on average by 10% and 7% respectively.

Contributions:

Our contributions are as follows.

- We present a method that can do hierarchical semi-supervised classification in the presence of incomplete class hierarchies. It enriches existing knowledge base in two ways: first, by adding new instances to the existing classes; and second, by discovering new classes and adding them at appropriate places in the ontology.
- Our proposed method, named OptDAC-ExploreEM uses divide and conquer style approach to add a new class and mixed integer programming based optimization to find the best set of labels for a datapoint given the ontological constraints.
- We demonstrate the effectiveness of our method through extensive experiments on four hierarchical entity classification datasets constructed with subsets of the NELL ontology (shown in Figure 1) and text, semi-structured HTML table datasets derived from ClueWeb09 [7].

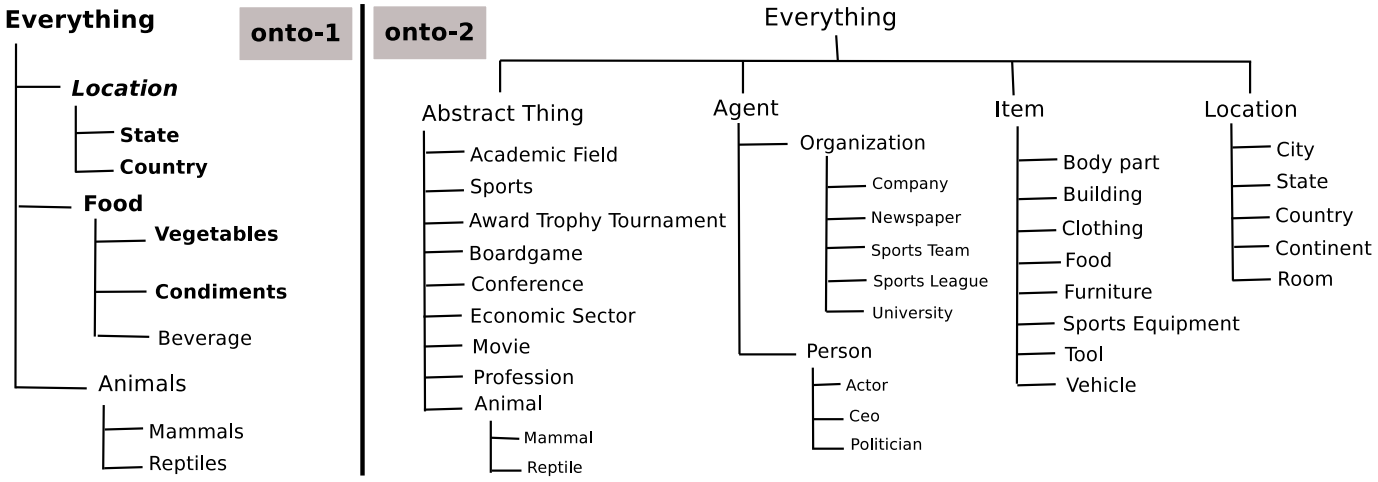

Figure 1: Subsets of NELL [8] ontology used in our hierarchical classification experiments. Some nodes in onto-1 are bold-faced, they are used as seeded classes in the experiments described in Section 5.

- To facilitate further research on this topic, we will make the datasets from this paper publicly available.

Outline:

Rest of the paper is organized as follows: In Section 2, we define the problem formally and then present a generic version of the "Hierarchical Exploratory EM" algorithm. Section 3 describes different variants of this algorithm. Overview of the datasets is covered in Section 4 followed by experimental findings in Section 5. We discuss the related work in Section 6, and conclude in Section 7.

2. OUR APPROACH

Our method is derived from the Exploratory EM algorithm proposed by Dalvi et al. [12]. Exploratory learning takes the same inputs as traditional Semi-Supervised Learning (SSL) methods, i.e. a set of classes $C_1, C_2, \ldots C_k$, a few labeled datapoints X^l and a large number of unlabeled datapoints X^u. X^l contains a (usually small) set of "seed" examples of each class, the task is to learn a model from X^l and use it to label datapoints in X^u. Every example x may be predicted to be in either a known class $C_i \in C_1 \ldots C_k$, or a newly discovered class $C_i \in C_{k+1} \ldots C_m$.

There are two main differences between the Exploratory EM algorithm (ExploreEM) and standard classification-EM (SemisupEM) approaches to SSL. First, in the E step, each of the unlabeled datapoint x is either assigned to one of the existing classes, or to a newly-created class. A new class is introduced when the probabilities of x belonging to existing classes are close to uniform. This suggests that x is not a good fit to any of the existing classes, and that adding x to any existing class will lower the total data likelihood substantially. Second, in the M-step of iteration t, we choose either the model proposed by ExploreEM method which might have more classes than the previous iteration $t-1$, or the baseline model with same number of classes as iteration $t-1$. This choice is based on whether exploratory model satisfies a model selection criterion in terms of increased data likelihood and model complexity. This intuition is derived from the Structural EM approach [14].

As per the experimental results presented in [12], the ExploreEM method is comparable or better than "Gibbs sampling with Chinese Restaurant Process (CRP) approach" and does not involve tuning the concentration parameter for CRP. However, ExploreEM is limited to flat class hierarchies. In this paper we propose the Hierarchical Exploratory EM algorithm which can work with incomplete class hierarchies and small amount of seed labels.

2.1 Problem Definition

Given a set of class constraints Z_k, a small set of labeled data points X^l, their labels Y^l, and a large set of unlabeled data points X^u; the task is to label data points from X^u, adding m new classes if needed and extending the class constraints to Z_{k+m}. Here, each point from X^l can have multiple labels at different levels of the hierarchy satisfying constraints Z_k, and Z_{k+m} defines the class constraints on k seed and m newly added classes, $Z_k \subseteq Z_{k+m}$ and the labels Y^u of X^u satisfy Z_{k+m}. Next let us see different methods proposed to solve this problem.

2.2 Flat Exploratory EM

One simple way to use ExploreEM algorithm [12] for our purpose will be to run it as it is at each level of hierarchy. Henceforth, we refer to this approach as FLAT-ExploreEM and consider it as a baseline for our proposed Hierarchical Exploratory EM approach. At each level, it selects one of the existing classes or creates a new class in case the datapoint doesn't clearly belong to one of the known classes. This algorithm does not make explicit use of class constraints while making class assignments at each step. Hence the assignments done by this algorithm might not be consistent, since assignments at level 3 are not influenced by assignment at level 2.

2.3 Hierarchical Exploratory EM

We summarize a generic Hierarchical exploratory learning algorithm that can take a set of class constraints in terms of subclass or mutual exclusion constraints (proposed in our preliminary work [11]). This algorithm, in each iteration,

Algorithm 1 Generic Hierarchical Exploratory EM

1: **function Hierarchical-ExploreEM** (X^l, Y^l, X^u, Z_k): θ_{k+m}, Z_{k+m}, Y^u
2: **Input**: X^l labeled data points; Y^l labels of X^l; X^u unlabeled data points; Z_k manually input constraints on k seed classes ⟸
3: **Output:** $\{\theta_1, \ldots, \theta_{k+m}\}$ parameters for k seed and m newly added classes; Y^u labels for X^u; Z_{k+m} Set of class constraints between $k+m$ classes; ⟸
{Initialize classifiers θ_j for class C_j using seeds provided for C_j}
4: $\theta_1^0 \ldots \theta_k^0 = argmax_\theta L(X^l, Y^l)$
5: **while** class assignments AND #classes not converged **do**
6: k_{old} is #classes before E step. Log-likelihood $BaseLL = logP(X|\theta_{k_{old}}^{(t)}, Z_{k_{old}}^{(t)})$
{E step: (Iteration t) Classify each datapoint at each level}
7: **for** i=1 to $|X|$ **do**
8: Find $P(C_j|X_i)$ for all classes C_j
{Assign a bit vector of labels for each unlabeled datapoint. A new class gets created for a datapoint that does not fit into existing classes.}
9: $Y_i^{(t)} =$ **ConsistentAssignment**$(P(C_j|X_i), h, Z^{(t)})$ ⟸
{If a new class is created, then class constraints are updated accordingly.}
10: $Z^{(t)} =$ **UpdateConstraints**$(X^l, Y^l, X^u, Y^u, Z^{(t)})$ ⟸
11: **end for**
12: k_{new} is #classes after E step. Log-likelihood $ExploreLL = logP(X|\theta_{k_{new}}^{(t)}, Z_{k_{new}}^{(t)})$
{M step: Recompute model parameters based on $Y^{(t)}$ }
13: **if** Model selection criterion(k_{old}, $BaseLL$, k_{new}, $ExploreLL$) selects exploratory model **then**
{Adopt the new model with k_{new} classes}
14: $\theta_{k_{new}}^{(t+1)} = argmax_\theta L(X^l, Y^l, X^u, Y^{u(t)}|\theta_{k_{new}}^{(t)}, Z_{k_{new}}^{(t)})$
15: $Z^{(t+1)} = Z_{k_{new}}^{(t)}$
16: **else**
{Keep the old model with k_{old} classes}
17: $\theta_{k_{old}}^{(t+1)} = argmax_\theta L(X^l, Y^l, X^u, Y^{u(t)}|\theta_{k_{old}}^{(t)}, Z_{k_{old}}^{(t)})$
18: $Z^{(t+1)} = Z_{k_{old}}^{(t)}$
19: **end if**
20: **end while**
21: **end function**

assigns a bit vector to each data point with one bit per class. Class constraints decide whether a bit vector is consistent or not, e.g. if class constraints include "Car is of type Machine", then for consistency, when the bit for "Car" is set, the bit for "Machine" should also be set. Further new classes can be added during each iteration, hence the length of these bit vectors changes dynamically and the algorithm maintains class constraints containing old as well as newly added classes. The generic Hierarchical Exploratory EM algorithm is described in Algorithm 1. The important differences from the FLAT-ExploreEM algorithm are marked using ⟸ . There can be multiple ways to implement functions "ConsistentAssignment" and "UpdateConstraints", which we will discuss below.

Lines 13-19 of Algorithm 1 does model selection. Similar to Exploratory EM [12], at the end of every E step, we evaluate two models, one with and one without adding extra classes. These two models are scored using a model selection criterion like AIC, BIC or AICc, and the model with best penalized data likelihood score is selected in each iteration. The extended Akaike information criterion (AICc) suited best for our experiments since our datasets have large number of features and small number of data points.

Algorithm 2 MinMax criterion for new class creation

1: **function Is Nearly Uniform**($[P(C_1|x) \ \ldots \ P(C_k|x)]$):
2: **Input**: $[P(C_1|x) \ldots P(C_k|x)]$: probability distribution of existing classes given a data point x
3: **Output:** *decision* : **true** iff new class needs to be created
4: k : current number of classes
5: $maxProb = \max(P(C_j|x))$
6: $minProb = \min(P(C_j|x))$
7: **if** $maxProb/minProb < 2$ AND $maxProb < 2/k$ **then**
8: $decision$ = true
9: **else**
10: $decision$ = false
11: **end if**
12: **end function**

Computing Probability of Classes given a Datapoint (Algorithm 1: Line 8)

This step computes probabilities $P(C_j|x_i;\theta_j)$, where θ_j is the current estimate of model parameters for class C_j. A variety of techniques may be used for this estimation, we briefly describe one such choice here: the seeded version of K-Means, proposed by Basu and Mooney [3]. In this model $P(C_j|x) \propto P(x|C_j) * P(C_j)$, and we define $P(x|C_j) = x \cdot C_j$, i.e., the inner product of a vector representing x and a vector representing the centroid of cluster j. Specifically, x and C_j both can be represented as L_1 normalized feature vectors. The centroid of a new cluster is initialized using feature counts from x. Since the EM algorithm can be used for both classification and clustering tasks, we will use the terms "class" and "cluster" interchangeably.

Modeling Class Constraints

Consider a toy example of ontological class constraints in Figure 2. Here, we can see two kinds of class constraints imposed by the ontology. Following are example constraints: (1) The "Subset" constraint between "Fruit" and "Food" categories suggests that if a datapoint is classified as "Fruit", then it should also be classified as "Food". (2) The "Mutual Exclusion" constraint between "Food" and "Organization" says if a datapoint is classified as "Food", then it should not be classified as "Organization", and vice versa.

Let $\{C_1, \ldots C_K\}$ be the Knowledge Base (KB) categories. Let *Subset* be the set of all subset or inclusion constraints, and *Mutex* be the set of all mutual exclusion constraints. In other words, $Subset = \{\langle i, k\rangle : C_i \subseteq C_k\}$ and $Mutex = \{(i, k) : C_i \cap C_k = \phi\}$. The class constraints referred to as Z_k in Algorithm 1 can be defined as $Z_k = \{Subset, Mutex\}$.

3. VARIANTS OF HIERARCHICAL EXPLORATORY EM

Hierarchical Exploratory EM (described in Algorithm 1) is a generic algorithm that can be instantiated in different ways, by changing the functions "ConsistentAssignment" and "UpdateConstraints". Next we describe two such vari-

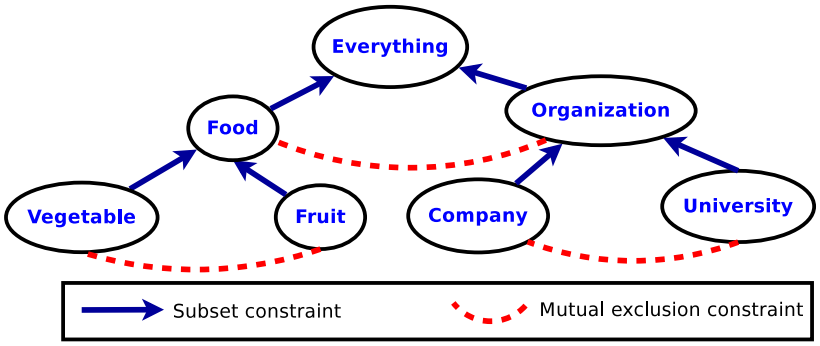

Figure 2: An example of ontological class constraints.

ants namely, DAC-ExploreEM and OptDAC-ExploreEM.

3.1 Divide and Conquer (DAC-ExploreEM)

One simple instantiation of Algorithm 1 can be done by using Divide-and-Conquer (DAC) strategy introduced in [11]. Here we assume that classes are arranged in a tree hierarchy, and classes at any one level are mutually exclusive. To do class assignments for any unlabeled datapoint, we traverse the class ontology from root to leaf level. Every data point belongs to the root node. Then at each level we chose the best label at that level and consider only its children as candidates at the next level.

Further we check whether the probability distribution among the candidate classes at each level is nearly uniform (using heuristic described in Algorithm 2) to decide whether to create a new class at that level. We do not describe the "ConsistentAssignment" and "UpdateConstraints" functions formally, however they can be easily derived by setting parameter $q = 1$ in the OptDAC-ExploreEM algorithm that we present next.

Also, note that the example ontologies in Figure 1 have tree structure. However, in practice, class constraints can be more complicated (e.g., overlapping classes can exist). The DAC-ExploreEM algorithm is limited to a tree structured ontology and assumes mutual exclusion of classes at any level of hierarchy. Next we present the OptDAC-ExploreEM algorithm that can work with more complicated class constraints.

3.2 Optimized Divide and Conquer (OptDAC-ExploreEM)

DAC-ExploreEM can do semi-supervised learning in the presence of unanticipated classes. However, we will see in the experimental evaluation (refer to Section 5.2) that DAC-ExploreEM could provide marginal improvements over the baseline (ExploreEM). During the error analysis we found that the classification at higher levels of hierarchy is not perfect, and once we make a decision at level i of the ontology, there is no way to change the decision once we move on to level $i+1$. Here we present a softer version of this method, that keeps track of top-q labels at each level instead of keeping only the best label.

Algorithm 3 describes how the "ConsistentAssignment" and "UpdateConstraints" functions are implemented for this approach. It is similar to DAC-ExploreEM method in the sense that we traverse the classes in top down fashion, and check whether new class needs to be created at each level of the hierarchy. However, at each level l of the class hierarchy, a mixed-integer program is run to get optimal class assignments for the active classes from levels 1 to l. Further instead of considering only the best classes in previous levels, the top-q classes from each level are selected to be added into the set of active classes, which are used in turn to select the candidate classes at the next level of hierarchy.

This method combines the Divide-and-Conquer strategy to detect/place new classes and the mixed integer programming based optimization strategy to assign an optimal set of labels to a datapoint given the ontological constraints. The optimal label assignment method is generic enough to support any set of subset and mutual exclusion class constraints. The new class detection based on Divide and Conquer can be extended for non-tree hierarchies using a breadth first search strategy that can be applied to any graph. Hence the OptDAC-ExploreEM method can be extended easily for non-tree structured ontologies.

Note that like DAC-ExploreEM, it makes greedy decisions about new cluster creation. However, it performs overall optimization of label assignments while satisfying the ontological constraints. This lets us correct any sub-optimal decisions made by the greedy heuristic at higher levels of the class hierarchy. Generally as we increase the value of q, we may get some improvement in accuracy at the cost of increased runtime. Since the value of q directly decides the size of active nodes set used while taking the decision at each level of the hierarchy, there is a trade-off between time complexity and solution optimality. For all the experiments in this paper, we added the top two classes per level to the set of selected classes (i.e. $q = 2$) in Line 21 of Algorithm 3. This approach is referred to as OptDAC-ExploreEM below.

Optimal Label Assignment given Class Constraints (Algorithm 3: Line 20)

Let $X = \{x_1, \ldots x_N\}$ be the datapoints, and $\{C_1, \ldots C_K\}$ be the KB categories. Let $y_{ji} \in \{0, 1\}$ be an indicator variable specifying whether x_i belongs to C_j. Let θ_j denote the centroid for category C_j. Using the model parameters θ_j for class C_j, we can estimate $P(C_j|x_i)$, the probability of x_i belonging to C_j. Given the category membership probabilities $\{P(C_j|x_i)\}$ estimated in the E step, this step computes the category membership variables $\{y_{ji},\ \forall\ 1 \leq i \leq N,\ 1 \leq j \leq$

Algorithm 3 OptDAC-ExploreEM

function ConsistentAssignment-OptDAC ($P(C_j|x)$, Z_k}): Y_x, Z_{k+m}
Input: $P(C_j|x)$ probability distribution of classes given a datapoint x; Z_k class constraints on k seed classes.
Output: $label(x)$ assignment of x to classes satisfying Z_k; Z_{k+m} extended set of class constraints on $k+m$ classes.
h is the height of the class ontology.
for $l = 1$ to h **do**
  if x has seed label at level l **then**
    label(x, $level_l$) = seed label;
  else
    $candidateC$ = children of active classes
    if $candidateC$ is not empty **then**
      Let P_{cand} = probability distribution over $candidateC$
      if Is Nearly Uniform (P_{cand}) (using Algorithm 2) **then**
        Create a new class C_{new} at level l
        Initialize parameters for class C_{new} using x
        Set parent(C_{new}) = class choice at level $l-1$
        Add C_{new} to active classes
      end if
      P_{active} = probability distribution over active classes
      Z_{active} = class constraints between active classes
      Choose $label(x, level_l)$ by computing optimal label assignment considering (P_{active}, Z_{active}) (using Eq. 1)
      Add top-q classes to the set of active classes using P_{active} as scores
    end if
  end if
end for
end function
function UpdateConstraints-OptDAC (X, Y, Z^{old}): Z^{new}
Input: X: Dataset; Y: Class assignments for each point in X;
Z^{old}: Old constraints on the existing set of classes.
Output: Z^{new}: Updated set of class constraints,
Each newly created class is assigned a single parent at the time of creation
Add each parent, child relationship as a constraint in $Z_{k'}$
end function

K}. We solve a Mixed-Integer Program (MIP) to estimate y_{ji}'s. One such problem is solved for each datapoint. This MIP takes the scores {$P(C_j|x_i)$}, and class constraints Z_k as input and produces a bit vector of labels y_{ji}'s as output.

$$\underset{\{y_{ji}\},\zeta_{jk},\delta_{jk}}{\text{maximize}} \left(\sum_{\mathrm{j}} \mathrm{y_{ji}} * \mathrm{P(C_j|x_i)} - \sum_{\langle \mathrm{i,k} \rangle \in \mathrm{Subset}} \zeta_{\mathrm{ik}} - \sum_{(i,k) \in Mutex} \delta_{ik} \right) \quad (1)$$

subject to,

$$y_{ji} \geq y_{ki} - \zeta_{jk}, \quad \forall \langle j,k \rangle \in Subset$$
$$y_{ji} + y_{ki} \leq 1 + \delta_{jk}, \quad \forall (j,k) \in Mutex$$
$$\zeta_{jk}, \delta_{jk} \geq 0,\ y_{ji} \in \{0,1\}, \quad \forall j,k$$

The MIP formulation for a datapoint x_i is presented in Equation 1. For each datapoint, this method tries to maximize the sum of scores of selected labels, after penalizing for violation of class constraints. Let ζ_{jk} be the slack variables for *Subset* constraints, and δ_{jk} be the slack variables for *Mutex* constraints. To solve these mixed integer linear programs we used the MOSEK solver [2].

Such optimization techniques have been shown to be effective for the task of semi-supervised learning in the presence of multiple data views and hierarchical class constraints [9, 13]. Here we use this formulation for the task of hierarchical exploratory learning.

4. DATASETS AND EXPERIMENTAL METHODOLOGY

In this section we present the experimental results of our Hierarchical Exploratory EM approach. Figure 1 shows two ontologies that we used in this paper, each being a subset of NELL's ontology at different point in NELL's development.

4.1 Datasets

Our task includes discovering new classes that are not present in the input class ontology. To make the evaluation easier, we created datasets that have ground truth labels for all entities in them, i.e. the entire class hierarchy is known. However, only part of that hierarchy and corresponding training data is given as input to the algorithm. Rest of the classes and corresponding labels are unknown to the algorithm, and used only during evaluation. Thus we are simulating the scenarios where some classes are known while others are unanticipated. To achieve this, we derived four datasets using the two ontologies (shown in Figure 1) and two feature sets extracted from the ClueWeb09 corpus. The first ontology, named onto-1 in Figure 1 (left), has 3 levels and 11 classes. The second ontology, named onto-2 in Figure 1 (right), has 4 levels and 39 classes.

We created our datasets using the two corpora: *Text-Patterns* and *HTML-Tables*, both derived from ClueWeb09 data [7]. *Text-Patterns* corpus contains frequency counts of text context patterns that occurred with each entity w.r.t. text sentences that appeared in ClueWeb09 dataset. *HTML-Tables* corpus contains frequency counts of table columns in which entities occurred, derived from HTML tables that appeared in ClueWeb09 dataset. E.g. an entity "Pittsburgh" having a *Text-Patterns* feature, value being ("lives in _arg1", 1000) means that the entity "Pittsburgh" appeared in arg1 position of the context "lives in _arg1" for 1000 times in the sentences from ClueWeb09 dataset. Similarly, an entity "Pittsburgh" having a *HTML-Tables* feature, value being ("clueweb09-en0011-94-04::2:1", 1) means that the entity "Pittsburgh" appeared once in HTML table 2, column 1 from ClueWeb09 document id "clueweb09-en0011-94-04".

To create a dataset from an ontology, we took all entities that are labeled with at least one of the classes under consideration, and retrieved their feature representation in terms of occurrences with text patterns or HTML table columns. Thus we created four datasets Text-Small to Table-Medium, using combinations of two ontologies and two corpora. Table 1 describes the statistics about these four datasets. We plan to make our hierarchical entity classification datasets publicly available upon publication of the paper.

4.2 Methods

We experimented with three different methods for the entity clustering task: FLAT, DAC and OptDAC. Each of them have SemisupEM and ExploreEM variants. The SemisupEM variant performs semisupervised learning with the seeded classes, whereas ExploreEM variant can add extra classes discovered from unlabeled data.

- FLAT: This method performs flat entity classification

Dataset	Feature Type	Ontology	#Classes	#Levels	#Classes per level	#Entities	#Contexts	#(Entity, context) pairs	#(Entity, label) pairs
Text-Small	Text	onto-1	11	3	1, 3, 7	2.5K	3.4M	8.8M	7.2K
Text-Medium		onto-2	39	4	1, 4, 24, 10	12.9K	6.7M	25.8M	42.2K
Table-Small	Tables	onto-1	11	3	1, 3, 7	4.3K	0.96M	6.3M	12.2K
Table-Medium		onto-2	39	4	1, 4, 24, 10	33.4K	2.2M	21.4M	126.1K

Table 1: Statistics of the hierarchical entity classification datasets used in this paper. Refer to Section 4.1 for more details.

at each level of the class hierarchy. Decisions made at each level are independent.

- DAC: This method performs hierarchical classification of entities, by making class assignment decision in top-down fashion. At each level maximum probable class is selected and candidates at next level are children of the class selected at previous level. This greedy algorithm is described in Section 3.1.
- OptDAC: This is another hierarchical classification method. It also makes class assignment decisions in top-down fashion. The creation of extra classes and placing them in hierarchy is similar to DAC method. However, class assignments at each level l are determined by solving a linear optimization problem considering all nodes under consideration (*ActiveC* in Algorithm 3) spanning levels 1 to l. Hence the greedy decisions about new cluster creations are combined with overall optimization of label assignments while following ontological constraints.

Further we used seeded K-Means algorithm for clustering (as described in Section 2.3) and the MinMax criterion [12] for new class creation (described in Algorithm 2).

4.3 Methodology

Remember that for the experiments in this paper, we feed a part of the ontology (seed classes) to the algorithm and rest of the part (unanticipated classes) is hidden from the algorithm. SemisupEM variant of each method learns classifiers only for seed classes. Along with this, ExploreEM variant of each method can add new classes. To make the semi-supervised and exploratory variants of each method comparable, we use "macro-averaged seed class F1" as the evaluation metric. It is computed by macro averaging F1 values of seed classes only. Further, if ExploreEM variant improves the seed class F1 over SemisupEM variant, it indicates that adding extra classes helped towards keeping the seed classes pure (i.e. reducing semantic drift).

This "macro-averaged seed class F1" metric is further averaged for 10 runs of each algorithm. Each run's input consists of different seed ontologies and randomly sampled 10% of relevant datapoints as seed examples. The same set of inputs is given to all algorithms being compared. Note that, for a given dataset with a choice of of seed classes and training percentage, there are many ways to generate a train-test partition. We report results using 10 random train-test partitions of each dataset. The same partitions are used to run all the algorithms being compared and to compute the statistical significance of results.

For Text-Small and Table-Small, we generated 10 sets of seed examples, for the same seed ontology. The chosen seed ontology is bold-faced in Figure 1 (left). Here seed ontology always contains the same 7 out of 11 classes. For Text-Medium and Table-Medium, seed ontology also varies across runs. In each run, we randomly chose 10 leaf nodes according to their class frequency (i.e. popular class is more likely to be chosen in each run). After sampling 10 leaf nodes (sampling without replacement), we included all their ancestors to create the seed ontology for that run. The average ontology size generated using this method was 16.7. Table 3 column 2 shows avg. number of seed classes chosen at each level of the hierarchy. 10% of the datapoints from leaf classes are them randomly sampled, and hierarchical labels of this datapoints are used as training data for the run.

5. EXPERIMENTAL RESULTS

In this section we will compare semi-supervised and exploratory variants of FLAT, DAC and OptDAC methods, in order to answer certain research questions.

5.1 Do ontological constraints help?

Table 2 shows the comparison between semi-supervised versions of all three methods. The best values in each row (per dataset per level in the hierarchy) are bold-faced. We can see that for every row, the best performance was given by a method that uses ontological constraints while clustering. Hence we can conclude that using ontological constraints do help.

We also did statistical significance tests with 0.05 (and 0.1) significance level denoted by ▲ (and △) in Table 2. Results show that in 5 out of 10 cases the gains of DAC over FLAT are statistically significant, whereas OptDAC method proved to be better by producing statistically significant gains over FLAT in 7 out of 10 cases.

5.2 Is Exploratory learning better than semi-supervised learning for seed classes?

We present the comparison of Semi-supervised vs. Exploratory versions of all three methods in Table 3. The best performance in each row is bold-faced. From Table 3 we can see that, ExploreEM version of each algorithm improves seed class F1 when compared to SemisupEM for all three methods in 24/30 cases. Our proposed approach OptDAC-ExploreEM improves seed class F1 on average by 13% when compared to its semi-supervised counterpart.

While comparing among the ExploreEM approaches, OptDAC method independently beats previously proposed FLAT and DAC methods in 8/10 cases each. Empirically, OptDAC-ExploreEM outperforms FLAT-ExploreEM and DAC-ExploreEM in terms of seed class F1 on average by 10% and 7% respectively.

Further we did statistical significance tests for perfor-

Dataset	Level	Macro-avg. Seed Class F1		
		W/o constraints	W constraints	
		FLAT	**DAC**	**OptDAC**
		SemisupEM	SemisupEM	SemisupEM
Text-Small	2	46.6	47.1	**52.0**
	3	23.5	**26.1**	25.8 △
Text-Medium	2	53.2	**53.7**	53.3
	3	27.9	33.4 ▲	**33.9** ▲
	4	17.4	24.5	**26.8** △
Table-Small	2	69.5	74.6 ▲	**74.8** ▲
	3	36.8	38.8 △	**38.9** ▲
Table-Medium	2	62.7	**64.8**	62.2
	3	43.7	46.4 ▲	**48.0** ▲
	4	47.3	**57.7** ▲	57.1 ▲

Table 2: Comparison of FLAT, DAC, and OptDAC methods in the semi-supervised setting. ▲ (and △) indicates that improvements of the DAC and OptDAC methods are statistically significant w.r.t the FLAT method with 0.05 (and 0.1) significance level. Please refer to Section 5.1 for details.

Dataset	#Seed /#Ideal classes	Level	Macro-avg. Seed Class F1					
			FLAT		**DAC**		**OptDAC**	
			SemisupEM	ExploreEM	SemisupEM	ExploreEM	SemisupEM	ExploreEM
Text-Small	2/3	2	46.6	**64.4**	47.1	61.8	52.0	62.6
	4/7	3	23.5	32.7	26.1	36.3	25.8	**42.3** ▲
Text-Medium	3.9/4	2	53.2	50.2	**53.7**	47.2	53.3	52.5
	9.4/24	3	27.9	27.0	33.4	26.8	33.9	**34.9** ▲
	2.4/10	4	17.4	25.8	24.5	29.4	26.8	**31.6** ▲
Table-Small	2/3	2	69.5	75.8	74.6	76.2	74.8	**80.0** ▲
	4/7	3	36.8	**43.9**	38.8	40.9 ▽	38.9	41.5
Table-Medium	3.9/4	2	62.7	61.3	64.8	**65.0** ▲	62.2	**65.0** ▲
	9.4/24	3	43.7	49.1	46.4	48.6	48.0	**50.1**
	2.4/10	4	47.3	52.2	57.7	**59.9** ▲	57.1	58.4 ▲

Table 3: Comparison of FLAT, DAC, and OptDAC methods using KM representation on Text-Small to Table-Medium. ▲ (and △) indicates that improvements of the DAC-ExploreEM and OptDAC-ExploreEM methods are statistically significant w.r.t FLAT-ExploreEM method with 0.05 (and 0.1) significance level. Please refer to Section 5.1 for more details.

mance improvements of DAC-ExploreEM and OptDAC-ExploreEM over FLAT-ExploreEM, with 0.05 (and 0.1) significance level denoted by ▲ (and △) in Table 3. It shows that improvements of DAC-ExploreEM are statistically significant in 2/10 cases, whereas improvements of OptDAC-ExploreEM are significant in 6/10 cases. Thus OptDAC-ExploreEM turns out to be the best method being compared.

5.3 What is the effect of varying the number of labeled examples?

We ran OptDAC-ExploreEM method on datasets Text-Small and Table-Small with different values of training percentage averaged over 10 random train/test partitions of our data.

In Figure 3, we can see that as the training percentage increases the performance of OptDAC-ExploreEM method improves. Also note that as we go down the hierarchy relative improvements are more larger. For example, there are larger relative improvements at Level 3 compared to Level 2 of the hierarchy.

5.4 How do the methods compare in terms of runtime?

Here we compare runtimes of all methods averaged over all the runs with different seed ontologies and seed training data. In our MATLAB implementation, the running time of Exploratory EM is much longer. Table 4 shows that the increase in runtime of ExploreEM variants w.r.t. their SemisupEM counterparts is by the factor 3.2 on average across all methods and datasets. Further the exploratory methods take on average 10.3 times the time of SemisupEM variant of the FLAT method.

In particular, OptDAC-ExploreEM method is on average twice as expensive as DAC-ExploreEM. This is due to the fact that DAC-ExploreEM takes greedy decisions at each level of the hierarchy, whereas OptDAC-ExploreEM keeps track of the top-q active nodes. We set $q = 2$ in these experiments. Thus the value of q results in a trade-off between improvement in seed class F1 and increased runtime of the algorithm.

5.5 Evaluation of extended cluster hierarchies

In this section, we present the evaluation of extra clusters added by our Hierarchical Exploratory EM algorithm to the incomplete class ontology given as input. If it started with k classes, and produced $k + m$ classes as output, we first need to label these m extra classes. Since our datasets are completely labeled, each new class can be assigned a majority label based on entities assigned to that class and level of

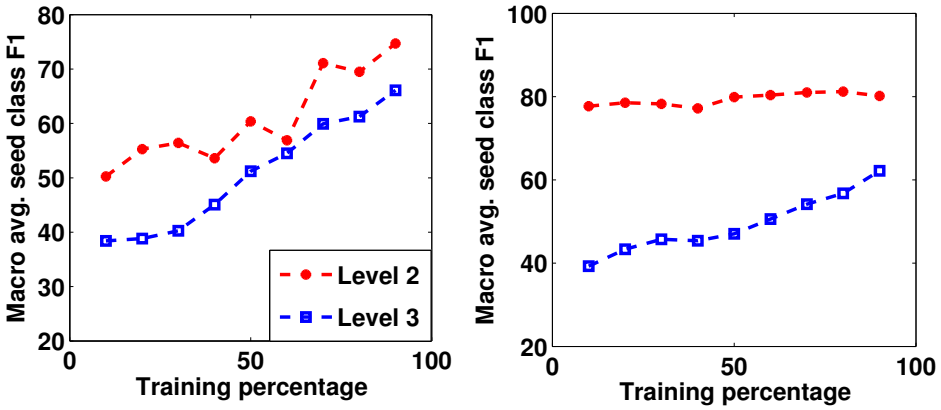

Figure 3: Comparison of OptDAC-ExploreEM method with different training percentage on datasets Text-Small (left) and Table-Small (right).

Dataset	Avg. runtime in sec.	Average run-time in multiple of FLAT-SemisupEM					
	FLAT	**FLAT**		**DAC**		**OptDAC**	
	SemisupEM	ExploreEM	SemisupEM	ExploreEM	SemisupEM	ExploreEM
Text-Small	53.5	8	1	6	7	17
Table-Small	50.7	3	3	6	10	21
Text-Medium	524.7	5	7	11	11	25
Table-Medium	5932.4	4	6	7	7	10

Table 4: Comparison of average runtimes of all methods. Please refer to Section 5.4 for more details.

the class in the hierarchy. E.g. if the class is at level 2 in the hierarchy then we choose the best label at level 2 of the class hierarchy.

Figure 4 shows an example of an extended class hierarchy generated by OptDAC-ExploreEM algorithm on Table-Small starting with 7 seed classes from onto-1. The bold-faced nodes are from seed ontology, and nodes in blue (non bold-faced) are the ones added by our proposed exploratory learning methods. It is labeled using the above described majority label approach. (Note that there are multiple clusters with same label, to differentiate them we have labeled them as "Food_1", "Food_2" etc.)

Once this labeling is done, we can measure the precision of labels across (parent, child) links in the cluster hierarchy. E.g. in Figure 4, parent-child link between (Food_2, Beverage_2) is correct, however the link between (Location, Beverage) is incorrect. These links can be classified into seed or extra links based on whether the child cluster was one of the seed clusters or introduced by the Hierarchical Exploratory EM algorithm.

Table 5 shows the link precision values for OptDAC-ExploreEM algorithm when run on all four datasets. We can see that for seed clusters, accuracy numbers are high (81 - 100%) for all four datasets. In terms extra clusters, for the Text-Small and Table-Small datasets with smaller ontology (onto-1), and unanticipated class fraction being low, the precision of edges for extra clusters is very high in around 85%.

However for the Text-Medium and Table-Medium datasets, with a medium sized ontology (onto-2), and with higher fraction of unanticipated classes, the precision for extra clusters is quite low around 30%. This indicates that the task of detecting new classes and adding them at right positions in the ontology is a challenging task, and it gets even more challenging with the complexity and degree of incompleteness of the input ontology.

Dataset	Fraction of unanticipated classes	%Precision of (parent, child) edges		
		Seed	Extra	All
Text-Small	0.36	88.3	84.5	85.5
Table-Small	0.36	100.0	90.4	92.4
Text-Medium	0.62	80.7	22.6	46.8
Table-Medium	0.62	95.7	36.1	63.9

Table 5: Precision of child, parent edges created by OptDAC-ExploreEM. Please refer to Section 5.5 for details.

Even though the newly created clusters are not perfect, we observed that these hierarchical exploratory learning methods improve seed class F1 on all four entity classification datasets (refer to Table 3). Thus, these techniques help reduce semantic drift of seeded classes by filtering out those datapoints that do not belong to any of the existing classes.

6. RELATED WORK

Here we propose a novel hierarchical SSL method that is robust when the unlabeled data contains unanticipated classes i.e. classes other than those present in the class hierarchy given as input. To the best of our knowledge this specific problem is relatively less explored, even though in

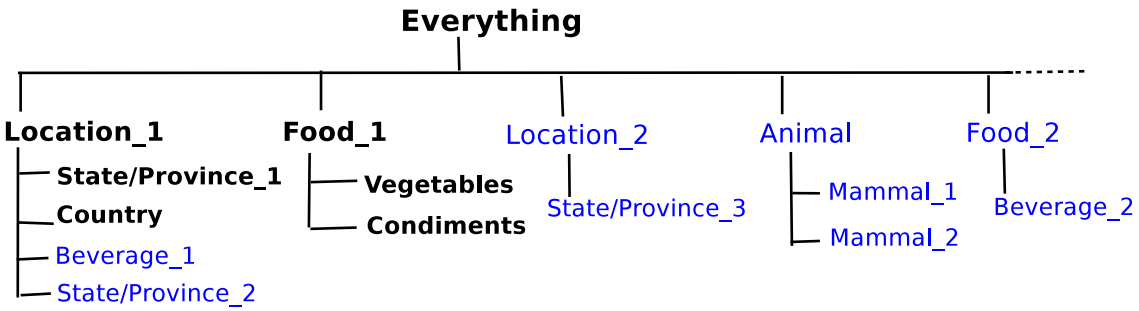

Figure 4: An example extended ontology applying our OptDAC-ExploreEM method on the Table-Small dataset. The seeded classes are bold-faced, whereas the newly added classes are in Blue and are not bold-faced. Please refer to Section 5.5 for details.

real-world settings, there can be unanticipated (and hence unseeded) classes in any large-scale hierarchical SSL task.

There has also been some work in unsupervised hierarchical clustering [17, 23, 5] that can discover cluster/topic hierarchies given a large unlabeled dataset, however they do not make use of any supervision that might be available. Exploratory learning differs in that we learn the number of clusters as well as centroids for those clusters jointly in a single run of the EM algorithm, while using the available supervision for seed clusters. Apart from standard K-Means, our EM framework can also be used with other clustering/classification algorithms like Naive Bayes and von Mises-Fisher, and we specifically evaluate the performance difference on the seeded classes.

There has been a lot of research done in the area of supervised hierarchical classification [6, 16, 24]. These methods assume that the class hierarchy is complete and there is enough training data to learn classifiers for each node in the class hierarchy. On the other hand we considered the situation where only part of the ontology is known upfront with very few seed examples for each of the seed class. Further our method can be easily extended to cases where class constraints are more complicated than the examples considered in this chapter, e.g. overlapping classes and mutual exclusion constraints.

Another related research area is constructing web-scale knowledge bases [8, 1] by doing information extraction from various data-sources. NELL internally uses Coupled semi-supervised learning [8] that takes into account subclass and mutual exclusion constraints among classes to filter extraction patterns and instances at the end of each bootstrapping iteration. The ontology (class hierarchy) is not explicitly used in the prediction process. I.e. it does flat classification with class constraints applied as post-processing in between two iterations of bootstrapping. Our approach on the other hand does collective hierarchical classification.

There has also been some work to extend existing ontologies. Mohamed et al. [19] propose a co-clustering based two step approach to discover new relations between two already existing classes in the knowledge base. These new relations are named using centroid features of the intermediate clusters. This method is focused on relation discovery between known classes. Snow et al. [22] discover new WordNet synsets by using evidence from multiple sources, however their approach is focused on discovering new isA relations, and not meant for building hierarchical classifiers for the learnt hierarchy. Pal et al. [20] proposed a technique based on Indian Buffet Process that could learn with existing feature hierarchies as well as extend them based on structure discovered from unlabeled data. Their method relies only on the containment relationships and the hierarchies they experimented with are domain specific e.g. restaurants domain.

Reisinger and Paşca [21] addressed the same problem as ours, working with the Wordnet hierarchy. Their fixed-structure and sense selective approaches use the Wordnet hierarchy directly and annotate existing concepts with generic property fields (attributes). On the other hand, Nested Chinese Restaurant Process (nCRP) approach is hierarchical extension of LDA to infer arbitrary fixed-depth tree structures from data. nCRP generates its own annotated hierarchy whose concept nodes do not necessarily correspond to Wordnet concepts. Our method is in the middle of these two approaches, as it uses the existing class hierarchy with small amount of training data and extends it dynamically as new clusters of datapoints are discovered.

Another set of techniques focus on completely unsupervised information extraction and ontology discovery [15, 4, 21, 10]. Though very effective, these approaches are not making use of the valuable information hidden in the existing knowledge bases. Our approach is relatively novel in the sense that it is in between semi-supervised and unsupervised learning, where some part of ontology is known, and this knowledge is used to discover the missing parts of the ontology along with populating it with new data instances.

To define the similarity among two entities we use bag of word features about co-occurrences of text patterns with the noun-phrases. There can be more effective approaches of document representation like the lower dimensional continuous Skip-gram features, proposed by Mikolov et al. [18]. Their technique learns low dimensional vectors that potentially embed semantics of noun-phrases.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we propose the Hierarchical Exploratory EM approach that can take an incomplete class ontology as input, along with a few seed examples of each class, to populate new instances of seeded classes and extend the ontology with newly discovered classes. Experiments with subsets of NELL ontology and and text, semi-structured HTML table datasets derived from the ClueWeb09 corpus show encour-

aging results in terms of seed class F1 scores. We will make our hierarchical entity classification datasets publicly available to encourage future research in this area.

Our proposed hierarchical exploratory EM method, named OptDAC-ExploreEM performs better than flat classification and hierarchical semi-supervised EM methods at all levels of hierarchy, especially as we go further down the hierarchy. Experiments show that OptDAC-ExploreEM outperforms its semi-supervised variant on average by 13% in terms of seed class F1 scores. It also outperforms both previously proposed exploratory learning approaches FLAT-ExploreEM and DAC-ExploreEM in terms of seed class F1 on average by 10% and 7% respectively.

In future, we would like to apply our method on datasets with non-tree structured class hierarchies. We briefly discussed how our proposed OptDAC-ExploreEM method can be used for this task. Further, our experiments focused on an information extraction task of classifying entities into a knowledge base class hierarchy. However, our techniques can also be applied to other tasks like document classification into topic hierarchies on datasets like Reuters, and classifying images into a class hierarchy for datasets like ImageNet.

References

[1] Freebase. *http://freebase.com.*

[2] AIMMS. The MOSEK toolkit.

[3] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *In Proceedings of 19th International Conference on Machine Learning (ICML-2002.* Citeseer, 2002.

[4] D. M. Blei, T. L. Griffiths, and M. I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):7, 2010.

[5] D. M. Blei, T. L. Griffiths, M. I. Jordan, and J. B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. *Advances in neural information processing systems*, 16:17, 2004.

[6] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 78–87. ACM, 2004.

[7] J. Callan. The clueweb09 dataset. `http://boston.lti.cs.cmu.edu/Data/clueweb09/`.

[8] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka Jr, and T. M. Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110. ACM, 2010.

[9] B. Dalvi and W. W. Cohen. Multi-view hierarchical semi-supervised learning by optimal assignment of sets of labels to instances. 2014.

[10] B. Dalvi, W. W. Cohen, and J. Callan. Websets: Extracting sets of entities from the web using unsupervised information extraction. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 243–252, New York, NY, USA, 2012. ACM.

[11] B. Dalvi, W. W. Cohen, and J. Callan. Classifying entities into an incomplete ontology. In *Proceedings of the 2013 workshop on Automated knowledge base construction*, pages 31–36. ACM, 2013.

[12] B. Dalvi, W. W. Cohen, and J. Callan. Exploratory learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 128–143. Springer, 2013.

[13] B. Dalvi, E. Minkov, P. P. Talukdar, and W. W. Cohen. Automatic gloss finding for a knowledge base using ontological constraints. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 369–378, New York, NY, USA, 2015. ACM.

[14] N. Friedman. The bayesian structural em algorithm. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 129–138. Morgan Kaufmann Publishers Inc., 1998.

[15] Z. Ghahramani, M. I. Jordan, and R. P. Adams. Tree-structured stick breaking for hierarchical data. In *Advances in Neural Information Processing Systems*, pages 19–27, 2010.

[16] S. Gopal, Y. Yang, B. Bai, and A. Niculescu-Mizil. Bayesian models for large-scale hierarchical classification. In *Advances in Neural Information Processing Systems*, pages 2411–2419, 2012.

[17] C. D. Manning, P. Raghavan, and H. Schtze. Introduction to information retrieval. In *Cambridge University Press*, 2008.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[19] T. P. Mohamed, E. R. Hruschka, Jr., and T. M. Mitchell. Discovering relations between noun categories. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1447–1455, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[20] A. Pal, N. Dalvi, and K. Bellare. Discovering hierarchical structure for sources and entities. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[21] J. Reisinger and M. Paşca. Latent variable models of concept-attribute attachment. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 620–628. Association for Computational Linguistics, 2009.

[22] R. Snow, D. Jurafsky, and A. Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 801–808. Association for Computational Linguistics, 2006.

[23] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[24] L. Xiao, D. Zhou, and M. Wu. Hierarchical classification via orthogonal transfer. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 801–808, 2011.