Toward Never-Ending Object Learning for Robots

Yuyin Sun

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Dieter Fox, Chair

Ali Farhadi

Yejin Choi

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Toward Never-Ending Object Learning for Robots

Yuyin Sun

Chair of the Supervisory Committee:
Professor Dieter Fox
Computer Science and Engineering

A household robot usually works in a complex working environment, where it will continuously see new objects and encounter new concepts in its lifetime. Therefore, being able to learn more objects is crucial for the robot to be continuously useful over its lifespan. Moving beyond previous object learning research problem, of which mostly focuses on learning with given training objects and concepts, this research addresses the problem of enabling a robot to learn new objects and concepts continuously. Specifically, our contributions are as follows:

First, we study how to accurately identify target objects in scenes based on human users' language descriptions. We propose a novel identification system using an object's visual attributes and names to recognize objects. We also propose a method to enable the system to recognize objects based on new names without seeing any training instances of the names. The *attribute-based identification system* improves both usability and accuracy over the previous ID-based object identification methods.

Next, we consider the problem of organizing a large number of concepts into a semantic hierarchy. We propose a principle approach for creating semantic hierarchies of concepts via crowdsourcing. The approach can build hierarchies for various tasks and capture the uncertainty that naturally exists in these hierarchies. Experiments demonstrate that our method is more efficient, scalable, and accurate than previous methods. We also design a crowdsourcing evaluation to compare the hierarchies built by our method to expertly built ones. Results of the evaluation demonstrate that

our approach outputs task-dependent hierarchies that can significantly improve user's performance of desired tasks.

Finally, we build the first never-ending object learning framework, NEOL, that lets robots learn objects continuously. NEOL automatically learns to organize object names into a semantic hierarchy using the crowdsourcing method we propose. It then uses the hierarchy to improve the consistency and efficiency of annotating objects. Further, it adapts information from additional image datasets to learn object classifiers from a very small number of training examples. Experiments show that NEOL significantly improves robots' accuracy and efficiency in learning objects over previous methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Professor Dieter Fox for his unfailing support and generous mentorship. Dieter taught me how to do high-quality research, and I would not have completed this dissertation without his support. I also thank other members of my supervisory committee, including Professor Ali Farhadi and Professor Yejin Choi, and graduate school representative Professor Ming-Ting Sun.

I would also like to thank Liefeng Bo, collaborator and mentor, who has provided invaluable advice and support.

Many thanks to my wonderful collaborators Professor Andreas Krause and Adish Singla.

I thank my colleagues (in alphabetical order), Arunkumar Byravan, Michael Jae-Yoon Chung, Daniel Gordon, Peter Henry, Evan evan Herbst, Kevin Lai, Cynthia Matuszek, Luis Puig, Connor Schenck, Tanner Schmidt, and Aaron Walsman for their help and support.

Finally, I would like to thank fellow students, staff, and the faculty of the UW Department of Computer Science and Engineering for creating a supportive and productive environment.

DEDICATION

To my family and friends, who have given me unconditional love and support.

Chapter 1

INTRODUCTION

1.1 Never-Ending Object Learning for Robots

A household robot working in a complex working environment will continuously encounter new objects and new concepts in its lifetime. Therefore, being able to learn new concepts and recognize new objects is a crucial capability for a household robot to be continuously useful over its lifespan. Whereas people gain knowledge of objects over many years, most current robot systems learn about objects just once. Moving beyond previous work on object learning, which mostly focuses on learning and recognizing with given training and testing objects and concepts, this research focuses on enabling a robot to learn new objects and concepts continuously.

A personal robot interacting with human users should be able to recognize new objects based on humans' object descriptions. Computer vision and robotics communities have made substantial progress in object recognition [50, 94, 121]. However, virtually all work on object recognition assumes that an object can be identified by some known category names. However, this is not how people would use language to identify objects in everyday settings [44] and does not generalize to new objects. People often use an object's name and additional attributes to identify objects, as doing so allows them to refer even to unusual objects. To address this question, Chapter 2 and Chapter 3 of this thesis explore the problem of language-based object identification. Specifically, we study how to recognize both familiar and unusual objects based on names *and* attributes.

A robot continuously learning objects will encounter growing numbers of objects and object names. Whereas humans can recognize thousands of object types, most existing robots are unable to do so. Inspired by human object recognition [103], many computer vision systems utilize semantic hierarchies of concepts to improve object categorization with thousands of classes [50, 121]. However, most methods assume that the semantic hierarchies are already provided. This assumption

will not hold in the robotic settings because no existing hierarchy can cover all concepts that a robot may encounter. Chapters 4 and 5 of this thesis explore methods for robots themselves to build semantic hierarchies for any concepts they encounter. To be more specific, we propose a series of crowdsourcing algorithms to create, extend, and evaluate hierarchies.

The predominant pipeline of a visual recognition system is to train a set of classifiers on a predetermined set of objects with a predetermined set of labels and then to test their performance on new instances of the same set of names. This train-once-then-test approach is inadequate for personal robots due to two main reasons. First, it is infeasible to assume that all objects and all possible labels are known to the robots in advance. A robot will continuously encounter new objects and labels over its lifetime. Therefore, it should be able to learn objects and labels continuously. Second, current visual recognition systems are trained to recognize objects based on categories, which are mostly words from dictionaries, like WordNet [69]. These "words" might not be the "names" used by a human in his/her language to refer to the objects [166]. A robot should be able to recognize objects based on real "names". Chapter 6 of this thesis proposes a Never-Ending Object Learning (NEOL) framework for robots. NEOL has three major components built upon the techniques introduced in Chapter 2 - 5: 1) it enables a robot to organize concepts into a semantic hierarchy, 2) it utilizes the hierarchy to annotate objects efficiently, and 3) it uses existing background knowledge bases to learn objects with just a few instances.

1.2 Background and Related Work

In this section, we will present a high-level overview of concepts and related work described in the thesis. Each subsequent chapter details work specifically related to this section.

1.2.1 Object Recognition

Object recognition [79] is an active research area of computer vision. Its ultimate goal is to understand the semantic meaning of every pixel of an image. Object recognition can be categorized into sub-tasks, such as classification, localization, and detection. In this thesis, we will focus

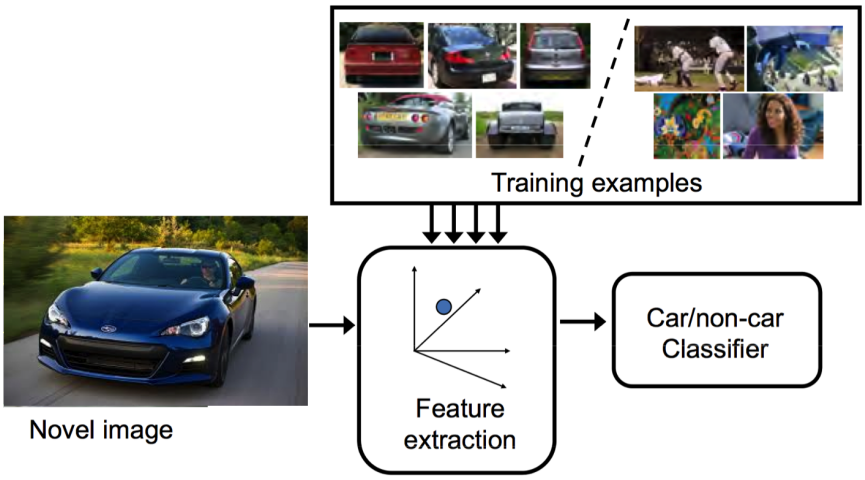


Figure 1.1: Predominant object recognition pipeline is to learn classifiers based on labeled training data and to test the classifiers on held out data.

on the classification task, which is about learning visual categories and then identifying new instances of learned classes. The predominant approach to object recognition/classification (shown in Figure 1.1) is to learn classifiers based on labeled training data and to test the classifiers on held out data [122, 50, 115]. Object recognition is a very broad field, so we will focus on discussing topics in object recognition, which are most relevant to our work.

RGB-D Object Recognition: Traditional visual recognition algorithms assume that sensors provide RGB images [50, 68, 6, 43, 66, 80]. Recently, commercialized RGB-D cameras, such as Microsoft Kinect [219] and PrimeSensor, can sense depth registered with RGB images with centimeter or millimeter accuracy. Depth information presents an opportunity for algorithms to increase their object recognition accuracy [122, 169, 14, 17, 16, 15]. Currently, RGB-D cameras have become standard sensors for most indoor robots. In this thesis, we assume all the visual inputs are RGB-D image pairs captured using Kinect sensors.

Feature Learning: Object recognition relies on robust feature representations of RGB images and depth maps. Hand-designed features, such as SIFT [133], HOG [43], and SURF [7], were the favorite features until the development of modern feature learning techniques [89, 215, 16, 14, 17, 115, 216, 170, 172, 85]. Feature learning, especially deep learning based methods, can learn features from raw images and have achieved huge success in the computer vision community [115, 216, 170, 172, 85]. Early deep learning methods [89, 215, 16, 14, 17] learned to extract hierarchies of features, layer by layer, using unlabelled image data. Recently, the creation of several large-scale labeled image datasets [50, 167] and the development of Graphics Processing Units (GPUs) have made the training of very deep and supervised deep learning model tractable [115, 172, 85]. Deep convolutional networks (ConvNets) have recently enjoyed great success in large-scale image and video recognition [115, 216, 170, 172, 85].

Domain Adaptation: Many large-scale labeled image datasets, such as ImageNet [50] and LabelMe [167], have been established recently for evaluating computer vision algorithms and training deep convolutional neural networks. These datasets are inherently biased to their application domains [192], however, and models built using them do not generalize well to robotics applications, which were not their intended applications. Many visual domain adaptation methods have been developed to deal with the dataset bias problems, for example, unsupervised adaptation [76, 75, 20, 97] and supervised adaptation [47, 118]. Recent work [93, 74, 137] trains deep neural networks to align multiple domains.

Attribute Learning: Visual attributes have been successfully used in object recognition research. Farhadi et al. describe unfamiliar objects by their attributes and show that attributes are generalizable to new categories. Lampert et al. use attributes for doing "zero-shot" learning and show that attributes are useful to detect unseen object categories. Parikh and Grauman model relative attributes and show the advantages over binary attributes in solving tasks such as image retrieval. Sun et al. use visual attributes extracted from natural language description to identify objects in complex scenes. Besides appearance attributes, like color, shape, and material, names are also treated as

attributes. Most work on attribute learning [66, 123, 153, 183] assumes that all attribute values are known beforehand to train corresponding classifiers. In practice, however, a robot is likely to face new names since people may use various names to refer to both known and unknown objects.

1.2.2 Integrating Language and Vision

Understanding Referral Language: Using language to refer an object in a scene to a hearer (other people or an autonomous robot) has been studied extensively by linguistics and cognitive scientists. Dale and Reiter [44] found that, for the goal of identification, people tend to use easily perceivable attribute-value pairs, like *color-red*, which do not lead to false implications. Kazemzadeh et al. design a crowdsourcing game to collect sentences used by humans to refer to objects in cluttered photographs. Ordonez et al. study how to predict entry-level categories, which are the labels people use frequently to name objects. Ordonez et al. propose models combining visual recognition predictions with linguistic resources like WordNet and proxies for word "naturalness" mined from the text corpus on the web to predict entry-level categories for a large number of images. The proposed models can generate more natural, human-like labels for images. Feng et al. study how to assign an entry-level name to every synset in WordNet.

Generating Descriptions: Computer vision researchers remain interested in generating language description of images. Some pioneering methods [91, 176, 67, 151, 100] treat the generation task as a retrieval task and search the most compatible captions in the training set as the prediction for the testing images. Other earlier approaches [84, 116, 67, 213, 61, 5] generate image descriptions based on fixed sentence templates. These earlier works are limited in their ability to generate various possible outputs. The most recent work [107], especially Recurrent Neural Network based methods [139, 200, 58, 108, 65, 34], can generate full sentence descriptions for images and has made significant improvements over previous methods.

1.2.3 Never-Ending Learning

Never-ending learning, a new learning paradigm, learns many functions in a cumulative manner. These functions, learned over years of accumulation, form an extensive background knowledge that improves subsequent learning [148]. Early works have focused on learning natural language (*e.g.*, Never-Ending Language Learning [31]), visual information (*e.g.*, Never-Ending Image Learning [36, 96]), autonomous control [189], *etc.*. It is worth noting that most never-ending learners focus on coverage, and therefore, usually deploy autonomous or semi-autonomous algorithms to learn from the web. Coverage-oriented learning systems often suffer from the problem of concept drift and low accuracy. In this thesis, we will propose a new never-ending learner for robots that will emphasize both coverage and accuracy.

1.2.4 Crowdsourcing

Putting humans in the loop of intelligent systems has provided many solutions to difficult problems that are hard for computers but relatively easy for people [153, 154, 220, 40, 131, 24, 203, 56]. Some successful applications of crowd-powered systems include protein folding [40] and galaxy discovery [131]. The computer vision community has also been deploying crowdsourcing to build better and more robust recognition systems. Some useful tasks include image annotation verification [24, 203], free-form object annotation [201], feature selection [56], and others. Many robotics researchers also start deploying crowdsourcing to teach robots capabilities such as grasping novel objects [177], learning human preferences [98, 1] for planning, and imitating human behaviors [39]. In this work, we utilize crowdsourcing to learn the semantic structure of names as well as to annotate objects.

1.3 Thesis Statement

The central thesis of this work follows

A robot can learn new objects more efficiently and accurately by integrating knowledge from multiple resources, such as semantic structures of concepts, the vision knowledge, natural language

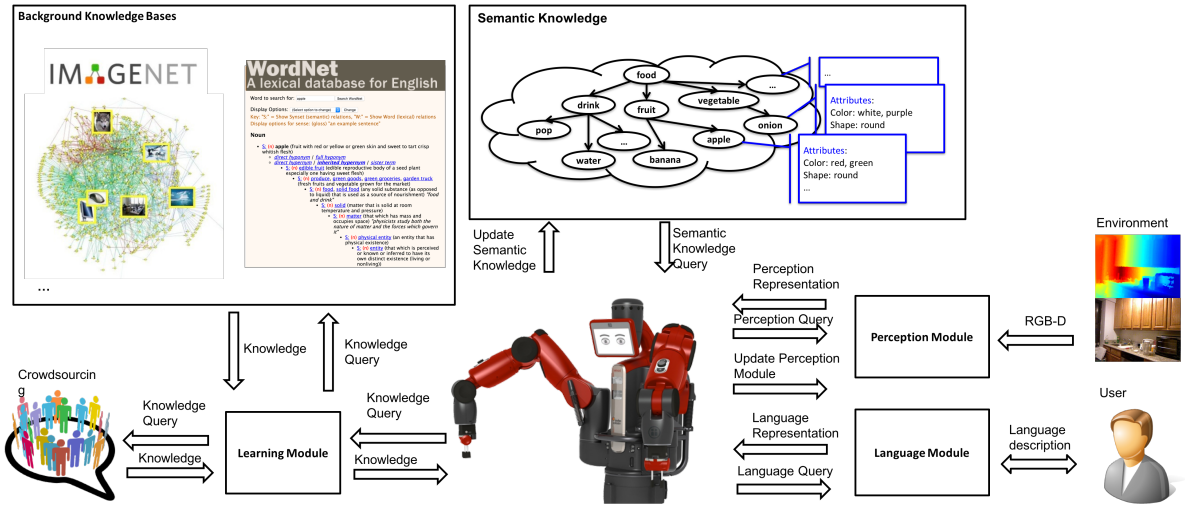


Figure 1.2: This thesis argues that a robot can learn new objects efficiently and accurately by integrating multiple knowledge sources.

description, established knowledge bases, and human teachers.

Figure 1.2 illustrates the idea. The robot, at the center of the diagram, has multiple resources from which it can acquire knowledge. For example, it could get well-structured and clean knowledge from established knowledge bases, such as ImageNet [50], which provides labeled images, and WordNet [69], which provides semantic knowledge of concepts, *et al.* Unfortunately, the online databases might be inadequate to answer all its questions. In this case, the robot could post the questions to humans, who might be the teachers or the workers for some crowdsourcing platforms. The robot has the access to various kinds of knowledge resources, the flexibility to choose resources to acquire knowledge, and the capability to combine all the information to learn objects more efficiently.

We also claim that the robot should keep a structured semantic representation of knowledge. Semantic knowledge, more specifically a hierarchy of concepts, lets the robot acquire future knowledge more efficiently. And future knowledge also enriches the robots' semantic hierarchy. Moreover, the semantic hierarchy of concepts can be easily shared with other robots around the

world.

Finally, we want to emphasize that robots learning process is trigged by its communication with human users and the environment, therefore, it involves understanding natural language descriptions and recognizing objects in working environments. Most existing never-ending learners, such as NELL [31] and NEIL [36], gather general knowledge existing on the web. In contrast, our robot has to learn new objects based on users' requirements. The involvement of humans makes the learning task less predictable and establishes a new direction for the Never-Ending Learning paradigm.

1.4 Thesis Outline

1.4.1 Attribute-Based Object Identification

In Chapter 2, we focus on how to identify objects based on language descriptions. While the computer vision community recently started to investigate the use of attributes for object recognition, these approaches do not consider the task settings typically observed in robotics, where a combination of appearance attributes and object names might be used in referral language to identify specific objects in a scene. In this chapter, we introduce an approach for identifying objects based on natural language containing appearance and name attributes. To learn the rich RGB-D features needed for classifying attributes, we extend recently introduced sparse coding techniques so as to learn attribute-dependent features automatically. We present a large dataset of attribute descriptions of objects in the RGB-D object dataset. Experiments on this dataset demonstrate the strong performance of our approach for language-based object identification. We also show that our attribute-dependent features provide significantly better generalization to previously unseen attribute values, thereby enabling more rapid learning of new attribute values.

The main research contributions of the chapter are: (1) establishing work as a pioneering effort on attribute learning for robots, and (2) Proposing a method to extract task-dependent features based on unsupervised feature learning techniques.

1.4.2 Learning to Identify New Names

In Chapter 3, we focus on a method of understanding new names and identifying objects based on new names without seeing any examples. Identifying objects based on language descriptions is an important capability for robots interacting with people in everyday environments. People naturally use attributes and names to refer to objects of interest. Due to the complexity of indoor environments and the fact that people refer to objects in various ways, a robot frequently encounters new objects or object names. To respond to such situations, it must be able to grow its object knowledge base continuously. In this chapter, we introduce a system that organizes objects and names in a semantic hierarchy. Then, the vector representation of words is learned to embed the hierarchy information. The vector representation allows measuring the similarity between words; therefore, it enables mapping novel objects and names to existing ones. Novel objects are inserted automatically into the knowledge base, where their exact location in the hierarchy is determined by asking a user questions. The questions are informed by the current hierarchy and the appearance of the object. Experiments demonstrate that the learned representation captures the meaning of names and is helpful for object identification with new names.

The main research contributions of the chapter are: (1) studying the novel problem of recognizing objects based on new object names without seeing any instances, and (2) proposing a metric learning method to learn hierarchy-embedded vector representation of names.

1.4.3 Building Semantic Hierarchies via Crowdsourcing

In Chapter 4, we focus on methods of building hierarchies of concepts using crowdsourcing. Hierarchies of concepts are useful in many applications, from navigation to organization of objects. Usually, a hierarchy is created in a centralized manner by employing a group of domain experts, a time-consuming and expensive process. The experts often design one single hierarchy to best explain the semantic relationships among the concepts, and they ignore the inherent uncertainty that may exist in the process. In this chapter, we propose a crowdsourcing system to build a hierarchy and furthermore capture the underlying uncertainty. Our system maintains a distribution over possible

hierarchies and actively selects questions to ask using an information gain criterion. We evaluate our method on simulated data and a set of real-world application domains. Experimental results show that our system is robust to noise, efficient in picking questions, cost-effective, and can build high-quality hierarchies.

The main research contributions of the chapter are: (1) proposing a novel method to create semantic hierarchies over a given set of words, (2) conducting extensive evaluations to demonstrate that the proposed method is robust, efficient and accurate in building various kinds of hierarchies, and 3) proposing a new model to enable a robot to infer user-dependent hierarchies as described in Appendix B.

1.4.4 Building and Deploying Task-Dependent Hierarchies: Theory Meets Practice

In Chapter 5, we perform an extensive user study of the proposed hierarchy learning systems. We propose a new evaluation method utilizing crowdsourcing to evaluate the quality of the built hierarchies. We use the crowdsourcing experiments to measure the performance of different hierarchies for an image retrieval task. Extensive evaluations demonstrate that our method can learn hierarchies more suitable for the task.

The main research contributions of the chapter are: designing a new evaluation experiment to estimate the quality of hierarchies for fulfilling tasks directly.

1.4.5 Never-Ending Object Learning for Robots

In Chapter 6, we propose a framework to enable a robot to learn objects over its lifetime. Learning to recognize objects based on names is a crucial capability for personal robots interacting with people in their environment. Recent recognition methods successfully learn to recognize objects in a train-once-then-test setting. These methods do not apply readily to robotic settings, where a robot might continuously encounter new objects and people might use new names to refer to them. In this work, we present a framework for Never-Ending Object Learning (NEOL). Our framework automatically learns to organize object names into a semantic hierarchy using crowdsourcing and background

knowledge bases. It then uses the hierarchy to improve the consistency and efficiency of annotating objects. It also adapts information from additional image data sets to learn object classifiers from a small number of training examples. We present experiments to test the performance of the adaptation method and demonstrate the full system in a never-ending object learning experiment.

The main research contributions of the chapter are: (1) presenting the first Never-Ending Object Learner for robots and (2) achieving both coverage and precision in the process of never-ending learning.

Chapter 2

ATTRIBUTE-BASED OBJECT IDENTIFICATION

2.1 Introduction

Identifying objects in complex scenes is a crucial capability in order for an autonomous robot to understand and interact with the physical world and be of use in everyday life scenarios. Over the last few years, the robotics community has made substantial progress in detection and 3D pose estimation of both known and unknown objects [94, 121]. The development of features and algorithms for combining color and depth information provided by RGB-D cameras further increased the accuracy of object detection [94]. So far, virtually all work on object instance recognition assumes that each object has a unique ID by which it is referenced. However, this is not how people would use language to identify objects in everyday settings [44]. Since not every object in an environment has a unique language identifier, people often use object name and additional *attributes* - such as color (blue), shape (round), size (large), material (metal), and others - to refer to specific objects. For instance, a person might say, "Bring me my coffee mug; it's the blue one." or "Pick up the Oreos on the kitchen counter." (see Figure 2.1). While the first statement requires a color attribute to identify the correct object, the object name "Oreos" is sufficient in the second statement.

The computer vision community recently started to investigate the use of attributes for object recognition [66]. Their work showed that it is possible to recognize new object types solely by their appearance attributes [123], and that using attributes improves recognition of fine-grained object classes such as bird species [59]. However, these approaches do not consider the task settings typically observed in robotics, where a combination of appearance attributes and object names might be used together to identify specific objects in a scene, rather than describing general object categories only. Recent work in semantic natural language processing introduced a joint model for language and visual attributes for the purpose of object identification [143]. Yet the main focus of

(a) Command could be "Bring me my coffee mug; it's the blue one."

(b) Command could be "Pick up the Oreos on the kitchen counter."

Figure 2.1: Object identification: Identify and visually recognize language attributes that refer to the desired object (marked by red rectangle).

the work was on language learning.

In this chapter, we introduce an approach for identifying objects based on appearance and name attributes (while people might use additional cues such as gestures and spatial attributes, this work focuses on intrinsic attributes only). Specifically, we consider the following *object identification task*: A robot perceives a set of segmented objects, is given a sentence describing attributes of one of the objects and has to identify the particular object being referenced. Attributes are grouped into different types: shape, color, material, and name. The attribute type name contains all words people might use to name an object. For instance, a name could be an object type, such as box, or a specific product, such as Mini Oreo. To learn rich RGB-D features needed for attribute classification, we build on recently introduced sparse coding techniques [17]. Our approach first learns codebooks from color and depth images captured by an RGB-D camera, and then it uses group lasso regularization to select the most relevant codewords for each type of attribute. These attribute-dependent codewords generalize much better than fully unsupervised learned codebooks, especially when only limited training samples are available.

To evaluate our approach, we collected an RGB-D attribute dataset for a subset of objects taken from the RGB-D object dataset developed by Lai et al. Attribute values are gathered via Amazon

Mechanical Turk. This dataset provides a rich selection of attribute words and names people use to describe objects. In addition to describing individual objects, we also collect a SCENE dataset, which contains multi-object scenes. The dataset simulates the task where a person might command a robot to pick up an object from multiple objects placed on a table. Experiments demonstrate that our learned appearance and name attributes are extremely well suited to identify objects.

This chapter is organized as follows. We present in Section 2.2 attribute types and our object attribute dataset. Then, in Section 2.3, we introduce our approach to attribute-dependent feature learning, followed by experimental results in Section 2.4. We conclude in Section 2.5.

2.2 Dataset

We developed a new RGB-D Object Attribute Dataset for training attribute classifiers. This dataset consists of 110 objects in 12 categories from the RGB-D Object Dataset [122]: *ball*, *coffee mug*, *food bag*, *food box*, *food can*, *garlic*, *instant noodle*, *marker*, *sponge*, *stapler*, *tomato* and *water bottle*. Figure 2.2 shows the 110 objects from our dataset. We collected labels for the attributes of objects using Amazon Mechanical Turk (AMT). In particular, we ask AMT workers to describe the color, shape, material, and name attributes of objects using simple but precise words or phrases. According to [44], those attributes are most frequently used in referral language.

We found that different workers tend to use the same words to describe the color, shape, and material attributes for the same objects. For these attribute types, we select the dominant words used to describe each attribute for each object instance. Overall, we collected eleven color attributes, three shape attributes, and six material attributes (see Table 2.1).

The name attribute annotations are inconsistent because people use different names to refer to the same object for different scenarios. For instance, some people say “bag of chips”, while and others say “Sun Chips” for the object instance of “Sun Chips”. So we associate *all* object names used by AMT workers with an object instance. As a result, the name attribute has 90 different values (words or phrases) and, for example, the “Sun Chips” object has several possible names, like “Sun Chips”, “bag of chips”, “bag” and so on. Table 2.1 shows some names used for objects in the food bag and instant noodles categories. This data is mainly used for the experiments described in

Figure 2.2: 110 objects in the RGB-D Object Attribute Dataset. Each image shown here belongs to a different object.

Sections 2.4.3 and 2.4.6.

Table 2.1: We collected attribute words used by people to refer to 110 objects in the RGB-D dataset. The words used for color, shape, material, and example name attributes in the Object Attribute Dataset are lists, as follows.

Attributes	Words
Color	red, orange, yellow, green, blue, purple, pink, brown, black, white, transparent
Shape	rectangle, cylinder, ellipse
Material	foam, paper, metal, plastic, food, ceramic
Name (food bag)	bag, bag of chips, barbecue chips, food bag, archer farms potato chips, cracker bag, package, bag of cracker, chicken biscuit, bag of pretzels, Oreo, snack, Snyder's pretzels, bag of Sun chips
Name (instant noodles)	instant noodles, ramen noodles, asian food, bag, Ichiban instant noodles

2.3 Method

In our object identification framework, the inputs are values for K attribute types, $A = \{a^1, \ldots, a^K\}$, and a set containing J segmented objects $\{I_1, \ldots, I_J\}$. The goal is to find the specific object j^* referred to by the attributes. We identify j^* by maximizing the likelihood of the attribute values A given the object I_{j^*}:

$$j^* = \underset{1 \leq j \leq J}{\operatorname{argmax}}\, p(A|I_j) = \prod_{k=1}^{K} p(a^k|I_j) \tag{2.1}$$

where $p(a^k|I_j)$ is the likelihood function. Here, we have factorized $p(A|I_j)$ by assuming that the attributes are independent given the object I_j.

We model the probability of values of each attribute type using a multinomial logistic regression. In particular, the probability of object I (we omit the subscript when possible) having attribute value

t is defined as

$$p(t|I,W) = \frac{\exp(f^t(I,W))}{\sum_{t'=1}^{T}\exp(f^{t'}(I,W))} \tag{2.2}$$

where W are the model parameters for the attribute type k learned from training data, and T is the number of attribute values belonging to the attribute type k (we used W and T instead of W^k and T^k for simplicity). The functions $f^t(I,W)$ are discriminative functions. It might be useful to think of $f^t(I,W)$ as a compatibility function that measures the compatibility of pairs of the attribute value t and the segmented object I. This is also called a "soft-max function" in the neural networks literature. The corresponding log likelihood is of the form $\log p(t|I,W) = f^t(I,W) - \log\sum_{t'=1}^{T}\exp(f^{t'}(I,W))$. We will detail the discriminative functions in the next section.

Accuracy of attribute recognition strongly depends on rich features extracted from the RGB and depth values of object segments. In our object identification framework, we first learn general feature codebooks in an unsupervised way [16, 17], and then we sparsify these codebooks to learn attribute-dependent features via group lasso optimization [150]. We first describe the general codebook learning approach.

2.3.1 Unsupervised Feature Learning

Our attribute dependent feature learning is built on hierarchical sparse coding [16, 17]. The key idea of sparse coding is to learn a codebook, which is a set of vectors, or codes, such that the data can be represented by a sparse, linear combination of codebook entries. In our case, the data are patches of pixel values sampled from RGB-D images. Our codebook learning algorithm uses K-SVD [2] to learn codebooks $D = [d_1, \cdots, d_m, \cdots, d_M]$ and the associated sparse codes $X = [x_1, \cdots, x_n, \cdots, x_N]$ from a matrix Y of observed data by minimizing the reconstruction error

$$\begin{aligned} \min_{D,X} \quad & \|Y - DX\|_F^2. \\ s.t. \quad & \|d_m\|_2 = 1, \ \ \forall m \\ & \|x_n\|_0 \leq Q, \ \ \forall n \end{aligned} \tag{2.3}$$

Here, the notation $\|A\|_F$ denotes the Frobenius norm for matrix A, the zero-norm $\|\cdot\|_0$ counts the non-zero entries in the sparse codes x_n, and Q is the sparsity level controlling the number of non-zero entries.

With the learned codebooks, sparse codes can be computed for new images using orthogonal matching pursuit or the more efficient batch tree orthogonal matching pursuit [16]. Spatial pyramid max pooling is then applied to the resulting sparse codes to generate object level features. A spatial pyramid partitions an image into multiple levels of spatial cells, and the features of each spatial cell are computed via max pooling, which simply takes the component-wise maxima over all sparse codes within a cell (see [17] for details).

2.3.2 Attribute Dependent Features via Codeword Selection

As we will show in the experiments, the features learned via hierarchical sparse coding give excellent results for attribute classification and object identification when learned on raw RGB and Depth image patches. However, a limitation of such rich features is that they might not generalize as well as task dependent features. For instance, imagine one wants to train a classifier for the color "red" by providing a small set of red example objects. In this case, overly general features might lead to shape specific codewords being used to learn a good classifier for the examples (overfitting). To avoid this, instead of learning only one, general codebook, we learn subsets of such a codebook containing only codewords useful for specific attribute types. Our approach sparsifies codebooks via group lasso [150]. We now describe this learning approach in the context of attribute classification.

We learn attribute classifiers using linear decision functions

$$f^t(I, W) = \sum_{s=1}^{S} \beta(I^s, D)^\top w^{s,t} + b^t \tag{2.4}$$

where $\beta(I^s, D)$ are pooled sparse code features over the spatial cell I^s, S is the number of spatial cells drawn from the object I, $w^{s,t}$ and b^t are weight vectors and a bias term, $W = [w^{11}, \cdots, w^{1T}, \cdots, w^{S1}, \cdots, w^{ST}]$, and T is the number of attribute values belonging to one attribute type. Here, $t \in \{1, \cdots, T\}$ denotes a specific attribute value for one attribute type. For instance, attribute values for the shape attribute are circular, cylindrical, ellipsoidal and rectangular.

Note that previous work has shown that linear classifiers are sufficient to obtain good performance for sparse code features [17].

We learn the model parameters from the training data collected from AMT. For each attribute type, training data consists of G pairs of the segmented objects I_g and their corresponding attribute values a_g. Here, a_g belongs to one of T attribute values (*e.g.*, one of the 13 color words for color attribute). We want to find the model parameters W by maximizing the log likelihood of training data

$$\sum_{g=1}^{G} \log p(a_g|I_g, W) - \lambda||W||_{21} \tag{2.5}$$

where $||W||_{21}$ is a regularization term, and its intensity is controlled by the parameter λ. Let w^i and w_j denote the i-th row and the j-column of the matrix W, respectively. The matrix norm $(2,1)$ is defined as $\|W\|_{21} = \sum_{m=1}^{M} \|w^m\|_2$. In our case, we have

$$||W||_{21} = \sum_{m=1}^{M} \parallel w_m^{11}, \cdots, w_m^{S1}, \cdots, w_m^{ST} \parallel_2 \tag{2.6}$$

where M is the size of the codebook. This regularization term is called group lasso; it accumulates the weights of the spatial cells and the attribute values for each codeword using 2-norm and then enforces 1-norm over them to drive the weight vectors of individual codewords toward zero. The group lasso thereby sparsifies the codebook and thus leads to an attribute-dependent codebook that is usually much smaller than the full codebook. If the group lasso drives an entire weight vector w_m to 0, the corresponding codeword no longer affects the decision boundary and has effectively been removed by the optimization. The $(2,1)$-norm regularized log likelihood is a concave function of W, so the optimal parameter settings can be found without local minima. We use the accelerated gradient descent algorithm to solve the above optimization problem; it has been proven to have a fast convergence rate [35, 132].

The intuition behind our codebook sparsification is that the full codebook learned by K-SVD consists of many types of codewords, while only a small number of codewords is relevant to a given attribute type. To demonstrate this intuition, we visualize the codebooks selected by our algorithm for color and shape attributes in Figure 2.3. As can be seen, the color attribute codebook contains

Figure 2.3: Sparsified codebooks for color (*left*) and shape (*right*) attributes. Our approach learned that solid color codewords are most relevant to classify colors, while depth codewords (grayscale) are most often selected for shape classification.

only color codewords, while the shape codebook is dominated by depth codewords (black and white) along with some color gradient codewords.

2.4 Experiments

2.4.1 Dataset

All datasets used in our experiments are available at

http://www.cs.washington.edu/rgbd-object-attributes

Training dataset–Object Attribute Dataset

We use the Object Attribute Dataset described in Section 2.2 to train the attribute classifier. Following the experimental setting in [17], we take video sequences captured from the $30°$ and $60°$ elevation angles as the training set and ones captured from the $45°$ angle as the test set (leave-sequence-out on the RGB-D dataset [122]). For training efficiency, only $1/4$ of all images/views are used for training, *i.e.*, around $2,500$ images for each classifier. All images used in the evaluation dataset,

(a) "Pick up the True Delight."

(b) "Pick up the brown Ichiban Noodle bags."

(c) "Pick up the bag of Mini Oreo."

(d) "Pick up the round plastic yellow marker with a black cap."

(e) "Pick up the basket ball."

(f) "Pick up the garlic with stem."

(g) "Pick up the rectangular one."

(h) "Pick up the chunky white box."

(i) "Pick up the Oreo box."

(j) "Pick up the tall yellow can."

Figure 2.4: Scenes and sentences collected from Amazon Mechanical Turk. Our system correctly identifies all objects except the ones in (f), (h), and (j). The attributes "stem", "chunky", and "tall" are relative or parts-based attributes not contained in the training data.

SCENE, and Section 2.4.3 are from sequences of the $45°$ angle.

Testing dataset–SCENE

To evaluate attribute-based object identification, we collect an additional dataset called SCENE, which contains $2,400$ scenes. Each scene has four different objects in it, one of which is the target object (that people refer to as the one marked by a red bounding box). We present each scene to AMT and ask workers to write a sentence, based on which robot can identify the target object. Figure 2.4 gives some example scenes along with the sentences collected via AMT.

Table 2.2: Statistics for the SCENE dataset.

	Overall	SCENE-D	SCENE-S
Raw Data	2,400	2,000	400
$Version1$	2,027	1,767	260
$Version2$	1,767	1,577	190

SCENE has two parts, SCENE-D and SCENE-S. The first part consists of $2,000$ scenes, each of which contains four images of four different objects randomly selected from all 110 objects. We call this set SCENE-D since the objects are typically from **d**ifferent categories. Panels (b), (c) and (i) in Figure 2.4 are examples of this part. The second part, SCENE-S, consists of 400 scenes containing 4 objects randomly picked from the **s**ame category. The remaining panels in Figure 2.4 are taken from SCENE-S.

To automatically extract attributes and names from the AMT sentences, we rely on WordNet [69] and the Stanford Part of Speech (POS) tagger [193]. First, we use POS to tag every word in a referral sentence. All adjectives are considered as appearance attributes, and noun phrases are treated as names of objects. However, the scene descriptions might contain attribute words that were not used in the training set. For each unknown adjective, we use WordNet to determine if it is a synonym or hyponym of a known attribute. Since not all new words can be mapped to any known words, the test scenes might contain information that our classifiers cannot use.

To evaluate how well sentences can be parsed automatically by our approach, we analyzed all $2,400$ sentences provided via AMT and checked which of them were sufficient to identify the referred object. This gave us a subset of clean data, called $Version-1$. As can be seen in Table 2.2, $2,027$ (or 84.5%) of the AMT sentences in the SCENE dataset are sufficient to uniquely identify the target object in the scene. Broken down into SCENE-D and SCENE-S, we get $1,767$ (88.4%) and 260 (65.0%) sufficient sentences, respectively. The lower percentage of correct sentences in SCENE-S is because it contains more complex scenes, and Mechanical Turkers tend to make more

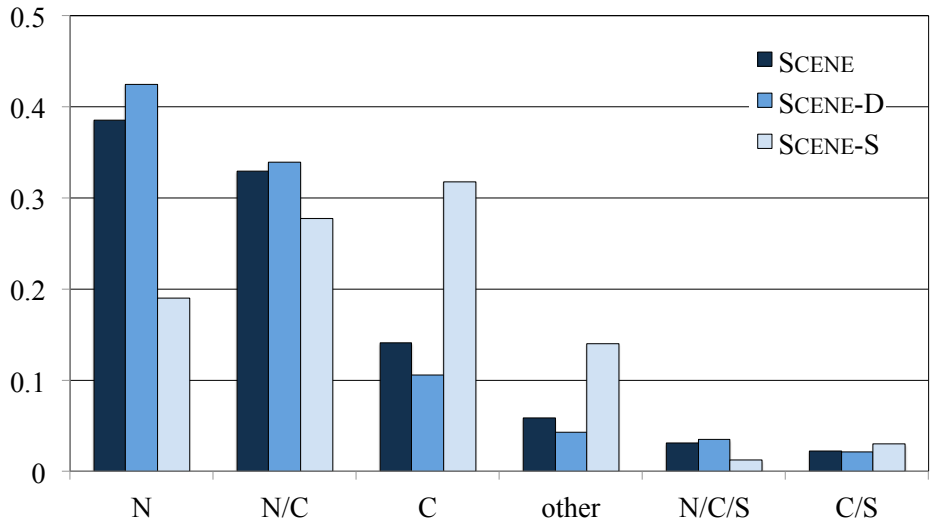


Figure 2.5: Frequency of different subsets of attribute types used by people to identify objects. **N** stands for name, **C** stands for color, **S** stands for shape and **M** is stands material. **Other** means other attribute types, such as relative or parts-based. Only the most frequently used attributes or attribute combinations are shown.

mistakes on these. We further checked among the sentences in $Version-1$ to determine how many of them our system could parse successfully to known attributes (we call this set $Version-2$). The numbers in Table 2.2 show that our language pipeline correctly parsed 89.2% ($1,577$ out of $1,767$) and 73.1% (190 out of 260) of the valid sentences in SCENE-D and SCENE-S, respectively. A careful analysis of the failure cases showed that for more difficult scenes, people use other types of attributes, such as relative attributes (darker, smaller) and localized attributes (having a white cap), which cannot yet be handled by our system.

We then evaluated which attribute types AMT workers used for object identification in the textsc-Scene dataset. Figure 2.5 shows percentages for the most frequently used combinations of attribute types among all tasks. For the textscScene-D set, Name is the most commonly used attribute. The second and third most frequent attributes are Name with Color (N/C), followed by Color only. The use of different attributes are not very surprising since Name and Color are very distinctive attributes, especially when the objects belong to different categories (it is interesting to note on Figure 2.6

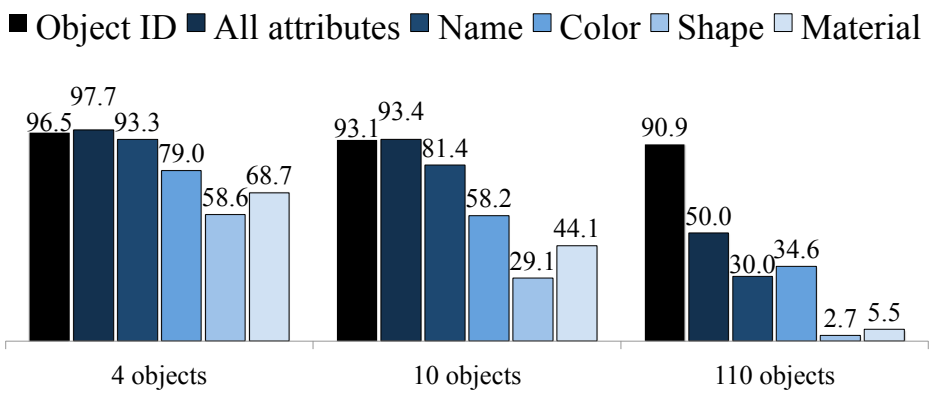


Figure 2.6: Object identification results using four types of attributes and their combination.

that Name and Color are also the two most discriminative attributes for our system). For SCENE-S, Name is not used as often as it is for textscScene-D. People tend to use Color and other attributes more often to identify specific objects.

2.4.2 Learning Setup

We learn general codebooks of size $1,000$ with sparsity level 5 on $1,000,000$ sampled 8×8 raw patches for both RGB and depth images. We remove the zero frequency component from raw patches by subtracting their means. With these learned codebooks, we compute sparse codes of each pixel (8×8 patch around it) using batch orthogonal matching pursuit with sparsity level 5, and we generate object level features by spatial pyramid max pooling over the whole images with 4×4, 2×2, and 1×1 partitions. The final feature vectors are the concatenation of the features over depth and color channels, resulting in a feature size of $42,000$ dimensions. The hyperparameters of sparse coding and multinomial logistic regression are optimized on the RGB-D object attribute training set (no test data was used for hyperparameter optimization). Average classification accuracy for individual attribute types i 97.9% (name), 89.1% (color), 94.7% (shape) and 97.0% (material).

2.4.3 Attribute Recognition for Object Identification

In this section, we evaluate the utility of attributes for object identification. The test data for object identification consists of $1,000$ scenes, each of which is generated by randomly picking $4-10$ different segmented object instances from the 45° angle test sequences. We consider three experimental settings.

In the first setting, we aim to identify the target object from a set of 4, 10, or all objects using all 4 attribute types provided by a Turker for the target object (*i.e.*, Turkers mention name, color, shape and material of the target object). We report the results of the different attribute types and their combination in Figure 2.6. First, we see that all four types of attributes are helpful for object identification, and that their combination outperforms each attribute by a large margin. For instance, for 10-object scenes, the combination of all four attributes achieves more than 10% higher accuracy than object Name and more than 20% higher than the appearance attributes (Shape, Color, and Material). Not surprisingly, the accuracy of object identification decreases with an increasing number of objects in the set. But the performance of the combination of all four types of attributes drops much less than the individual attribute. Figure 2.6 also shows the results of object instance recognition, in which each object gets a unique Object ID; (note that this is not the realistic setting for a natural language interface since not all objects in an environment can be named uniquely). It is very satisfying to see that our attribute-based identification slightly outperforms even this setting for 4 and 10 objects, indicating that learning multiple attribute classifiers provides higher performance than a single classifier even when trained on artificially provided IDs.

Note that the rightmost results represent an extreme case, assuming all 110 objects are placed in the same scene. Results show that we can achieve an accuracy of 50% using language attributes. The accuracy is significantly worse than that for object ID. The comparison suggests the potential benefit of introducing additional attribute types such as relative (darker, smaller) and spatial (the box on the left) attributes for more cluttered scenes.

The Gricean Maxims [44] suggest that people avoid redundancy in referral expressions and use only a subset of attributes for reference. In this second setting, we aim to identify the specific object

Table 2.3: Object identification accuracy (%) using different sets of attribute types.

	# of objects			
	4	6	8	10
All attributes	97.7	96.1	94.6	93.4
Minimal subset of attributes	91.0	88.8	87.3	86.1

from a set using a minimal subset of attributes, that is, only using attributes required to distinguish the target object. To get the minimal subsets, we try all the combinations of one, two, three, and four attribute types in turn and stop when the attribute combination is sufficient to distinguish the target object from the others (using ground truth attribute labels). We found that 87.6% scenes containing 10 objects can be identified based on only one attribute, 12.6% of them can be identified based on two attributes, and only 0.01% require three or more attributes. We report the results in Table 2.3. The results show that our approach obtains very high accuracy even in the minimal subset setting, suggesting the robustness of attribute-based object identification. Note that it is not surprising that the minimal subset of attributes works slightly worse than the combination of all four types of attributes since redundant attributes increase the robustness of object identification.

2.4.4 Scene-Based Object Identification

We now describe our evaluation of the SCENE dataset. We report the results on the raw data and the two validated subsets ($Version1$ and $Version2$) in Table 2.4. As can be seen, the attribute-based object identification framework achieves high accuracy even for the raw data. The accuracy on the verified data is even higher: 90.2% and 94.4% on the first type of test (objects are of different types). The accuracy decreases for SCENE-S, because the task is much harder than the first one. However, even in this most difficult setting, we still achieve 67.6% and 77.3% accuracy on the validated data and 64.3% on the raw AMT sentences (chance would be 25%). The overall performance on both types of tasks is 88.7% and 93.2% on the validated data and 83.7% on the raw data.

Table 2.4: Object identification results (Accuracy %) for SCENE, SCENE-D and SCENE-S.

	SCENE	SCENE-D	SCENE-S
Raw Data	83.7	85.6	64.3
$Version1$	88.7	90.2	67.6
$Version2$	93.2	94.4	77.3

2.4.5 Reproduce Human Behavior

From previous cognitive research [44], we know that people efficiently pick a subset of attributes to describe the target objects. They use minimum attributes when they are confident that a listener can differentiate the target object from other objects based on the selected attributes. We follow those rules and try to reproduce human behavior in accomplishing the identification task. Specifically, we want to choose the smallest attribute subset based on people's estimation about distinctiveness. The expected result of the experiment is to see the same frequency distribution over different attributes (like the frequency of Turkers' answers shown in Figure 2.5). This test would support our understanding of the identification task.

To reproduce human behavior, we first need to model the distinctiveness of attributes. Specifically, we want to know how much people believe an object o has a certain attribute value a. This could be computed using the attribute classifiers, which give $P(a|o)$. Based on this we compute the likelihood of our target object given attribute set $A = \{a_1, ..., a_K\}$ using the Bayes' rule, *i.e.*,

$$\begin{aligned} P(o|A) &= \frac{P(A|o)P(o)}{P(A)} \\ &= \frac{P(A|o)}{\sum_{o_i} P(A, o_i)} \\ &= \frac{P(A|o)}{\sum_{o_i} P(A|o_i)P(o_i)}. \end{aligned} \tag{2.7}$$

Using the naïve Bayes assumption that given object attributes are independent conditioned on

objects, and that each object is uniformly picked, Equation (2.7) will be rewritten as

$$P(o|A) = \frac{\Pi_k^K P(a_k|o)}{\sum_{o_i} \Pi_k^K P(a_k|o_i)}. \tag{2.8}$$

The likelihood of the target object represents people's confidence about using K attributes to make the target object distinguishable.

Second, we model how to create the smallest subset of attributes. We adopt a greedy algorithm, which is similar to the approach in [44]. Note that Dale and Reiter use an empirical estimation of the distinctiveness for each type of attributes and our work uses the distinctiveness given by the visual recognition system. Our distinctiveness measure could change according to different scenarios. For example, color is a very distinctive attribute according to Dale and Reiter. However, it should not be much less distinctive when the lighting condition of an image is quite poor. By comparing the performance of our method and the previous method, we could tell whether people also estimate distinctiveness of attributes based on the visual cue.

The greedy algorithm for generating a set of attributes works as follows: the algorithm first computes the confidence of success identification of the target object for the individual attribute. If any attribute generates confidence (Equation (2.8)) higher than a given threshold (set to 90% in our reproducing experiment), the algorithm will stop and use the most confident attribute for identification. Otherwise, two attributes will be used. The algorithm will keep adding attributes until the confidence level of correct identification is higher than the given threshold.

Following the greedy algorithm, we generate a subset of attributes that we believe people will use for each scene of the SCENE dataset. We compute the frequency of each attribute subset over the whole SCENE dataset, and we show the frequency of subsets frequently used by real people on the SCENE dataset (*i.e.*, **N**, **N/C**, **C**, **other**, **N/C/S** and **C/S**) in Figure 2.7.

Figure 2.7 shows that name and color are still frequently used. But they are not as frequent as when they are used by people in the real identification tasks. One possible explanation is that our strategy for selecting attributes assumes that people treat all attributes as being equally important without any preference. However, in reality people do have preferences for attributes. For example, people prefer to use names instead of materials even when they both are equally distinctive. This

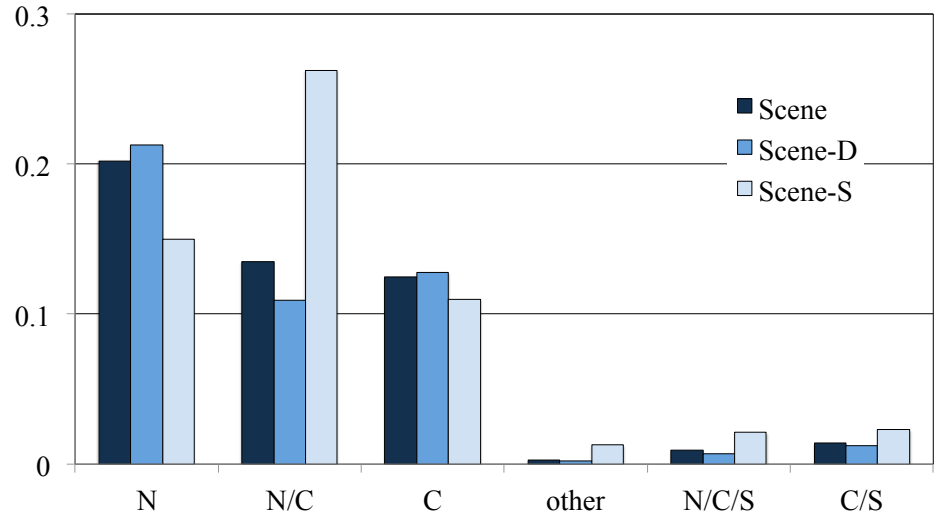


Figure 2.7: Frequency of different subsets of attribute types used by reproducing human behavior of identifying objects. **N** is short for name, **C** is short for color, **S** is short for shape and **M** is short for material. **other** means other attribute types such as relative or parts-based. Only the most frequently used attributes or attribute combinations are shown.

suggests that we should consider priors when generating attributes. Another possible reason is that our way of estimating distinctiveness using visual classification score is not very accurate. As we know the same vision task may be very simple for humans but very hard for computer vision classifiers. We will consider these two points when we build a better identification model in our future work.

2.4.6 Sparsified Codebooks for Transfer Learning

We perform the following experiments to investigate whether our attribute-dependent codebooks are more suitable for learning new attribute values compared to using the full codebooks. For each type of attribute, we leave two attribute values out as new (or target) attribute values and learn a sparsified codebook using the remaining attribute values. We then train models for the leave-out attribute values and test them on the test set for these attribute values. For example, in one setup we

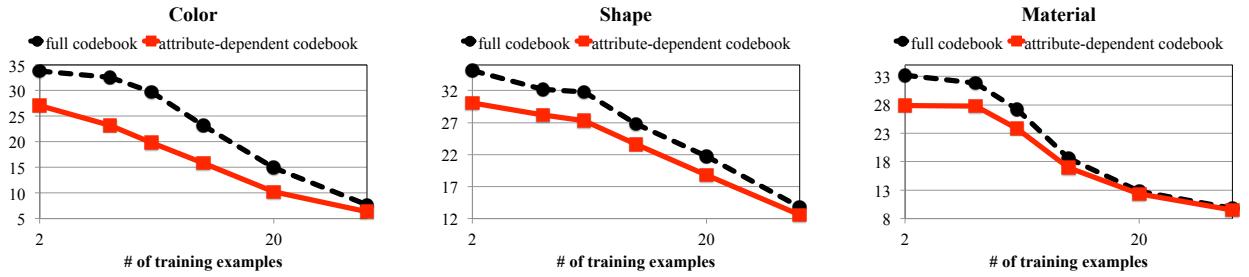


Figure 2.8: Error rates for learning new attribute values from different numbers of training examples (# of training samples are in log scale).

omit the colors blue and yellow, learn a sparsified color-dependent codebook using the other colors, and then use that codebook to learn classifiers for blue and yellow (see also Figure 2.3 for examples of sparsified codebooks for color and shape).

We compare the results obtained by the sparsified codebook and the full codebook in Figure 2.8. As can be seen, the sparsified codebooks significantly outperform the full codebooks on small numbers of training samples for color, shape, and material attributes. Unlike the full codebook, which contains a lot of redundant information for learning particular attribute types, the sparsified codebooks select codewords related to the particular attribute type; therefore, we can learn new attribute values especially with limited training samples. The sparsified codebooks result in much less chance to overfit to the training sets than the full codebooks and thus achieve better accuracy in this scenario. We also tried the sparsified and full codebooks for the object name attribute and found that these two approaches have comparable accuracy on the test set. The similar performance of the two codebooks is expected since the object name attribute is a highly mixed concept, and virtually all codewords are relevant to it.

2.5 Discussion

In this chapter, we present an attribute-based approach for object identification. We consider four types of attributes: shape, color, material, and name. We classify each type of attribute via

multinomial logistic regression. Our model learns perceptual features used for attribute classification from raw color and depth data. It incorporates sparse coding techniques for codebook learning and then uses group lasso regularization to select the most relevant codewords for each type of attribute.

To investigate the object identification task on realistic objects, we generate a large attribute data set by collecting descriptions for objects in an existing RGB-D object dataset [122]. Our experiments demonstrate that: 1) all types of attributes are helpful for object identification, and their combination works much better than the individual attribute, and 2) attribute-dependent sparsified codebooks significantly improve the accuracy over non-specific codebooks for learning new attribute values when only limited training samples are available. We believe that the capability of learning from smaller sets of examples will be particularly important in the context of teaching robots about objects and attributes and creates natural human-robot interfaces.

The work of this chapter has several limitations that deserve further study. Although treating different attribute types independently achieves excellent performance, considering them jointly might further improve performance. More complex combinations of attributes and gestures are also an important direction for future work. Attributes describing an object's spatial and physical relationship to other objects in a scene are also important. Those localized attributes are useful especially for fine-grained object identification. Last but not least, we only use a flat model for object names, while a more complex, hierarchical model could further improve results. We will present an improved object identification system - one that uses the hierarchy of names to recognize new objects based on new names - in the next chapter.

Chapter 3

LEARNING TO IDENTIFY NEW NAMES

3.1 Introduction

Identifying objects specified by a person is a crucial capability of household robots. When a person is referring to objects, he/she naturally refers to them using natural language to describe their attributes. For example, a person may say, "Here is a red Chex box"; "red" is a visual attribute (color), and "Chex box" is a name attribute. We discussed identifying objects based on attributes in the previous chapter; technique proposed in there assumes that all attribute values are known beforehand. However, due to the complexity of indoor environments and the many different names that can refer to objects, a robot will inevitably encounter unknown objects and names. This chapter studies techniques that enable robots to identify and learn objects in this challenging setting where objects are referred to by names.

One way of understanding new object names is to associate them with semantically related names learned by the robot. A natural approach to measuring the similarity between two words is to measure their distance defined by a taxonomy/hierarchy [124, 161, 210]. Figure 3.1 shows one example taxonomy/hierarchy. And according to the hierarchy, "Chex Box" is more similar to "Cereal Box" than "Apple", as the path between "Chex Box" and "Cereal Box" is shorter than the path between "Chex Box" and "Apple". The hierarchical organization of object names can then be applied to identify novel objects. For example, assume the robot has the name hierarchy shown in Figure 3.1 and a scene shown in Figure 3.2. It receives a new word "Chex Box" when a person says "Pick up the Chex box." It might have seen some other types of cereal boxes but not the Chex box. In this situation, it can still identify the correct object because "Chex" should be recognizable as a "Cereal Box." Hierarchies are useful for robot to recognize new objects and names.

However, names encountered by a robot may not have been included in an existing hierarchy.

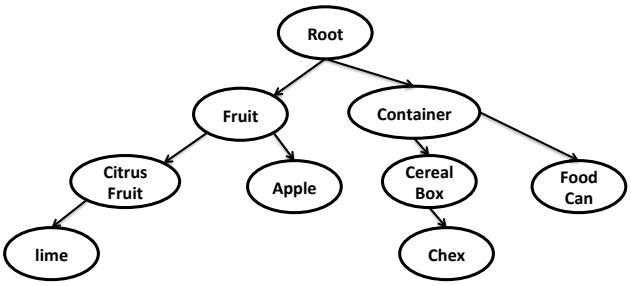


Figure 3.1: Part of an object name hierarchy.

Figure 3.2: Using the name hierarchy in Figure 3.1, the correct object in "Pick up the Chex Box" can be identified through the "Cereal Box" classifier even without ever having observed a Chex box before.

For example, the most widely used semantic hierarchy, WordNet [69], covers many category names like "cereal box". But it does not usually include instances (*e.g.*, "Chex") that are frequently used by people [183] to refer to objects. Moreover, people may use different words to refer to the same concept, *e.g.*, people might call a "mug" a "cup". It is thus necessary for an autonomous robot to automatically fulfill the following two tasks: 1) extend an existing hierarchy by inserting new instance names into the right places, and 2) associate synonyms with existing words in the hierarchy.

To insert an unknown name into an established hierarchy, we need to find a path of words in the hierarchy such that all words are hypernyms of and semantically similar to the target word.

We could use existing work [134, 196] to represent each word using a vector and computing the distance between vectors to measure the similarity between words. The vector representation of words has been proven to have good correlation with the human judgment of words' similarity [81] and is used widely in practice. However, few methods considered the hierarchical structure of words in learning the words' representation when this work was first published.

In this chapter, we propose a new method to learn a vector space representation of words that captures hierarchy information. In the hierarchy-embedded space, the distance between words is coherent with the distance within the name hierarchy. This let the robot use the similarity in the new space to find a path of words containing hypernyms of the new word from the existing hierarchy more accurately. We also propose a method to let users help the system find the exact meaning of a name and, therefore, extend the name hierarchy.

This chapter is organized as follows. After discussing related work in Section 3.2, we introduce the identification system in Section 3.3. We then propose a name learning method in Section 3.4. Section 3.5 discusses how users can interact with the system to find the exact meaning of names. We show the effectiveness and efficiency of the proposed system in Section 3.6 and we conclude our discussion in Section 3.7.

3.2 Related Work

Many efforts have been made for helping computers to learn the meaning of new words. One way is by using domain experts' knowledge. One famous example is the WordNet database [69], which organizes words in a hierarchy based on their semantic meaning. It relates words by their synonymy and hypernymy relationships. This relationship is useful for understanding words since if a word is included in the WordNet hierarchy, its meaning can be inferred from its ancestors and descendants. Although WordNet covers a large number of words, most of them are category names.

Another direction is to learn the meaning of words from documents. A word can be represented as a vector [62] generated from a large corpus of documents. The vector is computed based on the word frequency within the context of the target word, where context could be documents, paragraphs or other small windows within documents. The context-based vector representation has been

successfully applied to many natural language applications such as search query expansion [104], information retrieval [155], and automatic annotation of text [163]. Although the recent success of deep learning methods has improved the quality of vector representations [136, 157], most methods still do not consider structured semantic meaning, such as synonym or hypernym relationships. In this chapter, we will discuss a method to learn a vector representation that embeds the hierarchy information to make the representation more coherent with the experts' knowledge, *e.g.*, WordNet.

Our research did not uncover much work on embedding hierarchy information into the vector representation of words when we began. Recently, we have become aware of related work [199], which further demonstrates the importance of the work in this direction.

3.3 Indentification System

In this chapter, our goal is to build an identification system for RGB-D scenes such that a human can interact with the system by natural language. We train our system based on the attribute-based identification system described in Chapter 2. To be more specific, we keep the same appearance attributes as the previous system and elaborate the name attributes component. We use the same notations as the previous chapter: o for object, a for attribute, and I for object segmentations. In this section, we treat 'name' as a special attribute and use w to denote it.

In the previous chapter, names were treated independently. However, this treatment ignores the semantic relation between them. Here, our system considers the semantic relation between names and organizes them as a semantic hierarchy $\mathcal{H}$. Each node n in the hierarchy is a category like "Fruit", "Container", "Cereal Box", and others. Each node is associated with all synonyms corresponding to that category. For example, the node "Cereal Box" may be related to "Food Boxes" and "Cereal Box". Note that this organization of names is similar to WordNet synsets. Hence, we can adopt WordNet to initialize the hierarchy $\mathcal{H}$.

We then build name classifiers using the hierarchy. For every non-leaf node with more than one child, we train a multinomial logistic regression over all the objects belonging to different child nodes of this node. Here, we extract Hierarchical Matching Pursuit features over object segments. Note that each classifier actually predicts a conditional probability $P(n_i|o, n_j)$, where n_j is the

parent of n_i. To get the probability of $P(w|o)$, where w is a name contained in a node n_w of the tree $\mathcal{H}$, we need to consider all the predictions given by node classifiers along the path from the root to n_w denoted as $\boldsymbol{p}^w = [n_1^w \rightarrow \cdots n_{(L-1)}^w \rightarrow n_L^w]$ with $n_L^w = n_w$. Using the product rule and conditional independence encoded by the tree-structure, we have the following

$$P(w|o) = \prod_{l=1}^{L-1} P(n_{l+1}^w|o, n_l^w). \tag{3.1}$$

If a person uses a name that is new to the system, it is more challenging to find the correct path. We will handle this case in the next section.

3.4 Method

To handle new names, our identification system will associate them with existing ones in the hierarchy so as to find a path for name prediction. Here, we propose a method based on learning vector representation of names.

Our system starts with a set of names organized in a tree-structured name hierarchy $\mathcal{H}$. Each node in $\mathcal{H}$ contains at least one name word w_i . Given a new name w, the goal of our method is to find a path of names starting from the root node and ending at the node to which the new word belongs. The new word belongs to a node such that: 1) if the new word is a synonym of an existing word, the new word will be added to the node of its synonym, and 2) if the new word is a hyponym (specialization) of a leaf node, a new leaf node will first be added as a child of the current leaf node, and the new name will then be associated with the new leaf node. In this chapter, we assume that the system has already knowns all object categories; therefore, we do not consider adding new category nodes. We will discuss how to relax this constraint in the next chapter.

3.4.1 New Name Inference

For a name that is new to $\mathcal{H}$, we will find the node into which the new name should be inserted. The target node should contain words with similar semantic meaning similar to the new name. One naïve solution is to measure the similarity between the new name and all the nodes in $\mathcal{H}$, then select

the node with the highest similarity to the new name. However, this approach could cause semantic drift. To avoid semantic drift, we use the path as a regularization: the new name should be similar not only to one node, but also to all its ascendant nodes.

Next, we will describe how to measure similarity between nodes. Motivated by the success of previous work by Socher et al., we represent every word w as a d-dimensional vector $\boldsymbol{v}^w \in \mathbb{R}^d$ using a vector space model and measure the similarity between words using Euclidean distance in the space. $\boldsymbol{v}^w$ is given by the co-occurence with other words within a large document corpus [175]. To measure the distance between the new name w and a node n associated with M names $\{\boldsymbol{v}^{n^1_w}, \cdots, \boldsymbol{v}^{n^M_w}\}$, we use the average of M words distances. This is the same as representing a node in the same d-dimension space as

$$\boldsymbol{v}^n = \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{v}^{n^m_w}.$$

As can be seen, a node can be represented in the same vector space for names. So we will treat a node the same as a name in the following section.

3.4.2 Hierarchy Embedded Vector Representation

The original vector representation does not explicitly consider the hierarchy relation between words. Therefore, the distance between words might not reflect the distance embedded by the hierarchy. In this section, we will learn a new vector representation, that can embed hierarchy knowledge.

Given a new word w, a hierarchy $\mathcal{H}$, and the ground truth path $\boldsymbol{p} = [n_1 \rightarrow \cdots n_l \cdots \rightarrow n_L]$ that leads to a node n_L associated with w, we want the ground truth path to have the best score. Specifically, we define a decision function $\psi : (w, \mathcal{H}, \boldsymbol{p}) \rightarrow \mathbb{R}$ to measure the match score between a path $\boldsymbol{p}$ and a word w given the hierarchy $\mathcal{H}$, such that

$$\boldsymbol{p}^w = \arg\max_{\boldsymbol{p}} \psi(w, \mathcal{H}, \boldsymbol{p}), \tag{3.2}$$

where $\boldsymbol{p}^w$ is the ground truth path. In the following, we replace a word with its vector representation $\boldsymbol{v}^w$ and omit w for simplicity. For $\boldsymbol{v}^{n_l}$, we use $\boldsymbol{v}_l$ when there is no confusion.

For a path $\boldsymbol{p}$ with L words, we compute the decision score of the path as the sum of scores of all nodes in it

$$\psi(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p}) = \sum_{l=1}^{L} \frac{\phi(\boldsymbol{v}, \boldsymbol{v}_l)}{L}, \tag{3.3}$$

where $\phi(\boldsymbol{v}, \boldsymbol{v}_l)$ measures the match score between words $\boldsymbol{v}$ and $\boldsymbol{v}_l$. The intuition behind Equation (3.3) is that if $\boldsymbol{p}$ is the real path, $\boldsymbol{v}$ is semantically close to all $\boldsymbol{v}_l$ within the path, resulting in a high score for each $\phi(\boldsymbol{v}, \boldsymbol{v}_l)$. The resulting total score $\psi(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p})$ for the path will then be high, as well. The length of the path L is used to normalize the score so as to remove the bias toward long paths.

A natural choice of $\phi(\boldsymbol{v}, \boldsymbol{v}_l)$ is the negative distance between $\boldsymbol{v}$ and $\boldsymbol{v}_l$ as

$$\phi(\boldsymbol{v}, \boldsymbol{v}_l) = -(\boldsymbol{v} - \boldsymbol{v}_l)^T(\boldsymbol{v} - \boldsymbol{v}_l).$$

However, the original vector space does not explicitly consider the structural semantic information embedded in the name hierarchy. The correct path might thus not maximize the score function defined in Equation (3.3).

To embed hierarchy information into the score function, we parametrize the score function as

$$\psi_A(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p}) = \sum_{l=1}^{L} \frac{\phi_A(\boldsymbol{v}, \boldsymbol{v}_l)}{L}, \tag{3.4}$$

where

$$\phi_A(\boldsymbol{v}, \boldsymbol{v}_l) = -(\boldsymbol{v} - \boldsymbol{v}_l)^T A^T A(\boldsymbol{v} - \boldsymbol{v}_l). \tag{3.5}$$

Here $A \in \mathbb{R}^{d \times d}$ is a linear transformation mapping $\boldsymbol{v}$ into a new space. $A^T\boldsymbol{v}$ is the new representation of word w. $\phi_A(\boldsymbol{v}, \boldsymbol{v}_l)$ measures the negative distance between words in the new vector space. Let $W = A^T A$, and we can rewrite Equation (3.5) as

$$\phi_W(\boldsymbol{v}, \boldsymbol{v}_l) = -(\boldsymbol{v} - \boldsymbol{v}_l)^T W(\boldsymbol{v} - \boldsymbol{v}_l), \tag{3.6}$$

where W uniquely defines the new space and is usually called a metric.

Learning the parameterized score function can be formulated as a structure prediction problem [187]. To be consistent with the hierarchy information, the parameterized score function has to satisfy the constraints

$$\psi_W(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p}^w) \geq \psi_W(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p}), \forall \boldsymbol{v}, \boldsymbol{p}$$

where $\boldsymbol{p}^w$ is the ground truth path for $\boldsymbol{v}$ provided for the training data. Thus, the correct path should have a higher score than any other path $\boldsymbol{p}$. We define a shorthand $\delta_W \psi(\boldsymbol{v}, \boldsymbol{p}) \equiv \psi_W(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p}^w) - \psi_W(\boldsymbol{v}, \mathcal{H}, \boldsymbol{p})$, giving us the set of constraints as

$$\delta_W \psi(\boldsymbol{v}, \boldsymbol{p}) \geq 0, \forall \boldsymbol{v}, \boldsymbol{p}. \tag{3.7}$$

If the set of constraints in Equation (3.7) is feasible, there typically will be more than one solution W. To make the solution unique and avoid overfitting [187], we generalize the idea of max margin and select a W of low rank by minimizing its trace

$$\begin{aligned} &\min_W \ \text{tr}(W) \\ &\text{s.t. } \delta_W \psi(\boldsymbol{v}, \boldsymbol{p}) \geq 1, \forall \boldsymbol{v}, \boldsymbol{p} \\ &W \succeq 0. \end{aligned}$$

The condition $W \succeq 0$ enforces W to be Positive Semi-Definite (PSD) so as to be a valid metric and decomposable as $A^T A$.

Different predictions, or paths $\boldsymbol{p}$ for $\boldsymbol{p}^w$, should be penalized with different loss: if two paths diverge at the higher level of the tree, the loss should be high since two concepts are too far away from each other. We use tree loss $\Delta(\boldsymbol{p}^w, \boldsymbol{p})$ to measure the loss of mis-predicting the real path $\boldsymbol{p}^w$ as $\boldsymbol{p}$, which is the deepest level in a tree such that $\boldsymbol{p}$ and $\boldsymbol{p}^w$ are the same. Then the constraint set becomes

$$\delta_W \psi(\boldsymbol{v}, \boldsymbol{p}) \geq \Delta(\boldsymbol{p}^w, \boldsymbol{p}), \forall \boldsymbol{v}, \boldsymbol{p}$$

So the final optimization problem is the following convex optimization

$$\min_{W} \text{tr}(W) + C \sum_{i=1}^{N} \xi_i, \tag{3.8}$$
$$\text{s.t. } \delta_W \psi(\boldsymbol{v}^i, \boldsymbol{p}) \geq \Delta(\boldsymbol{p}^i, \boldsymbol{p}) - \xi_i, \forall i, \boldsymbol{p}$$
$$W \succeq 0, \xi_i \geq 0$$

where we also introduce the slack variables ξ_i, and i indexes all training examples.

Note that the training examples for this learning problem can be gathered from the known hierarchy $\mathcal{H}$, which could be a part of the WordNet hierarchy. For each non-root word in $\mathcal{H}$, we can find a unique path from the root. The pair of a node and its path are one training example. In total, we can collect $N = |\mathcal{H}| - 1$ training pairs from the hierarchy.

3.4.3 Optimization

The optimization problem in Equation (3.8) is convex and, therefore, has a global solution. However, the number of constraints in Equation (3.8) grows quadratically as $|\mathcal{H}|$ increases. Efficiency will thus be an important issue to solving the problem. Fortunately, only part of the constraints define the feasible set of solutions for the constraint optimization problem, so we can use the cutting-plane method [102], which efficiently solves constraint problems because it uses only active constraints.

We adapt the 1-Slack cutting plane algorithm used by McFee and Lanckriet to our problem. In the original optimization problem (Equation (3.8)), there are N-Slack variables. The 1-Slack method combines a batch of active constraints specified as $(\boldsymbol{p}_b^1, \cdots, \boldsymbol{p}_b^N)$ into one single constraint. All new constraints share the same slack variable ξ. It thereby reduces the number of constraints and slack variables and makes very large optimization problems solvable.

The algorithm is summarized in Algorithm 1, which alternates between updating W and updating the active constraint set $\mathcal{W}$. Line 5 of Algorithm 1 updates W with constraints specified by $\mathcal{W}$ by gradient descent. The gradient of the object function in Equation (3.8) over W is computed as

$$\frac{\partial f}{\partial W} = I - \frac{C}{N} \sum_{i=1}^{N} \delta_W \psi(\boldsymbol{v}^i, \boldsymbol{p}_*^i),$$

where $*$ indicates the single batch of constraints $(\boldsymbol{p}_b^1, \cdots, \boldsymbol{p}_b^N)$ with the largest gap between $\frac{1}{N}\sum_{i=1}^{N}\psi(\boldsymbol{v}^i, \boldsymbol{p}_b^i)$ and $\frac{1}{N}\sum_{i=1}^{N}\Delta(\boldsymbol{p}^i, \boldsymbol{p}_b^i)$. After the gradient descent update of W, the algorithm projects W back onto the set of PSD matrices by spectral decomposition.

After updating W, the algorithm finds the $\boldsymbol{p}_b^i$ maximizing the gap between $\Delta(\boldsymbol{p}^i, \boldsymbol{p}^i)$ and $\delta_W\psi(\boldsymbol{v}^i, \boldsymbol{p}^i)$ for the i-th word (line 8). $(\boldsymbol{p}_b^1, \cdots, \boldsymbol{p}_b^N)$ are put together as the b-th new batch, which will be added into the active constraint set $\mathcal{W}$.

The cutting-plane method converges when the gap between $\frac{1}{N}\sum_{i=1}^{N}\psi(\boldsymbol{v}^i, \boldsymbol{p}_b^i)$ and $\frac{1}{N}\sum_{i=1}^{N}\Delta(\boldsymbol{p}^i, \boldsymbol{p}_b^i)$ is smaller than a given threshold ϵ (line 4). Theoretical results guarantee that it converges quickly [102]. In practice (see Section 3.6), we also found that it converges after only few iterations. Algorithm 1 returns the metric W parameterizing the path score function given as input.

3.5 Interaction with Humans

When using of the system, the path $\boldsymbol{p} = [n_1 \rightarrow \cdots n_L]$ determined for a new word w might not be entirely correct. Inserting w into the hierarchy solely based on the most likely path $\boldsymbol{p}$ would introduce substantial noise and semantic drift. To avoid this, our system interacts with a human by asking questions to find the correct place for w. The "best place" here means either w is a synonym of n_L, or w is a hyponym of n_L and n_L is a leaf node in $\mathcal{H}$. The second case indicates w is a new instance name.

3.5.1 Interaction with Humans

Our system interacts with people by asking questions. After finding the most likely path $\boldsymbol{p} = [n_1 \rightarrow \cdots n_L]$, the system asks a question for the last node n_L to determine its relationship with the new word. There are three possible relationships between w and n_L: w is a synonym, hyponym or neither. If w is a synonym of n_L, it will be associated with n_L. If w is a hyponym of a leaf node n_L, w will be inserted into the hierarchy as a new child node of n_L. If n_L is neither case of w, n_L should not be part of the correct path $\boldsymbol{p}^w$. The system then finds the next best path and repeats the same procedure. The correct path will eventually be found since we assume a name is either a synonym

Algorithm 1 Algorithm for learning hierarchy-embedded vector space.

1: **Input**:

Words representation $\{\boldsymbol{v}^1, \cdots, \boldsymbol{v}^N\}$;

rankings $\{\boldsymbol{p}^1, \cdots, \boldsymbol{p}^N\}$;

Slack trade-off C;

Convergence tolerance ϵ

2: **Output**:

Metric W

3: $\mathcal{W} \leftarrow \emptyset, \xi \leftarrow 0$

4: **while** $\epsilon + \frac{1}{N}\sum_{i=1}^{N} \delta_W \psi(\boldsymbol{v}^i, \boldsymbol{p}_b^i) \geq \frac{1}{N}\sum_{i=1}^{N} \Delta(\boldsymbol{p}^i, \boldsymbol{p}_b^i) - \xi$

do

5: update metric

$$
\begin{aligned}
W &= \arg\min_{W} \ \mathrm{tr}(W) + C\xi \\
\text{s.t. } \frac{1}{N}\sum_{i=1}^{N} \delta_W \psi(\boldsymbol{v}^i, \boldsymbol{p}_b^i) &\geq \frac{1}{N}\sum_{i=1}^{N} \Delta(\boldsymbol{p}^i, \boldsymbol{p}_b^i) - \xi \quad (3.9) \\
&\forall\ (\boldsymbol{p}_b^1, \cdots, \boldsymbol{p}_b^N) \in \mathcal{W}
\end{aligned}
$$

6: **for** $1 \rightarrow$ N **do**

7: $\boldsymbol{p}_b^i = \arg\max_{\boldsymbol{p}} \Delta(\boldsymbol{p}^i, \boldsymbol{p}) - \delta_W \psi(\boldsymbol{v}^i, \boldsymbol{p})$

8: $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\boldsymbol{p}_b^1, \cdots, \boldsymbol{p}_b^N)\}$

of an existing word or a new instance name.

3.5.2 Incorporating Perception

In the attribute-based identification setting, a user uses an object name together with several appearance attributes to specify the target object. An intuitive way of deciding which node to ask about could thus be based on information gain, using the posterior uncertainty over the objects in

the scene

$$n^* = \arg\min_n H[P(o|n, a^1, \cdots, a^k)], \tag{3.10}$$

where $H[\cdot]$ is the entropy. Unfortunately, Equation (3.10) cannot be estimated robustly. So our system estimates the likelihood $P(n, a^1, \cdots, a^k|o)$ instead of the posterior. Since a person attempts to uniquely specify an object in the scene, we know that only the target object has a high overall score of

$$P(w|o)\Pi_{k=1}^{K} P(a^k|o), \tag{3.11}$$

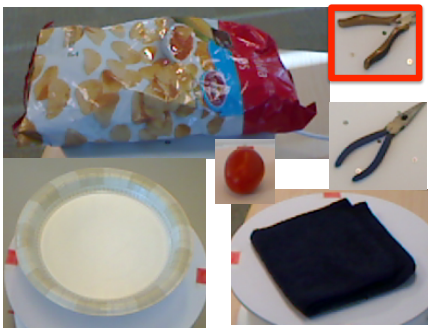
while the rest of the objects have far lower scores. Base on this fact, we can infer the relationship between n_L and w: we replace w in Equation (3.11) with n_L to compute the score of Equation (3.11) for each object in the scene. If no object has a high score, the system infers that w is neither a synonym nor a hyponym for n_L and rejects the hypothesis n_L without asking any question. If there are several objects with high scores, it infers that n_L cannot make the target object unique, which implies that w might be a hyponym of n_L, where several objects of the type corresponding to n_L are in the scene. We use a threshold on the score to distinguish these cases.

3.6 Experiments

We evaluated the name learning and object identification system with three sets of experiments on the object attribute dataset [183]. Our results suggest that: 1) name learning significantly outperforms baselines in terms of accuracy of associating new names with the existing name hierarchy (Section 3.6.2), 2) name learning significantly boosts the object identification system's accuracy, particularly when new names are being used (Section 3.6.3), and 3) name learning significantly reduces the human effort required to correctly insert new names into the existing hierarchy (Section 3.6.4).

3.6.1 Data Set

We use all 300 objects from the RGB-D object dataset [122] in our experiments. 150 objects were used to train attribute and name classifiers. All 300 objects are used for testing, *i.e.*, 150 new



(a) A scene given to student 1. The student says "Pick up the yellow clipper."

(b) A scene given to student 2. The student says "Pick up the Pepsi."

Figure 3.3: Two example scenes used in the experiments.

objects in the testing stage. Name and appearance annotations for these objects were collected using Amazon Mechanical Turk. These attribute labels are available online at our project website[1].

We initialize hierarchy $\mathcal{H}$ using the WordNet hierarchy, shown in Figure 3.4. In the figure, all leaf nodes are in blue, with the number of object instances in each leaf node in parentheses. We extracted features for object images using Multi-Path Hierarchical Matching Pursuit, a feature learning approach proposed in [18]. Our system then builds attribute classifiers for color, shape, and material, and an object name classifier for each internal node in the hierarchy using a multinomial logistic regression implemented in LibSVM [33].

We generated 600 "scenes", each of which contains 6 randomly picked objects arranged in image panels. On average, half of the objects have never appeared in the training set. For each scene, we marked the target object with a red rectangle. Figure 3.3 shows two example scenes. The yellow "pliers" is the target object in Figure 3.3 (a), and the "Pepsi" can is the target object in Figure 3.3 (b). We asked two students at the University of Washington to identify the target object using one sentence for each scene. Each student took charge of 300 scenes.

[1] http://www.cs.washington.edu/rgbd-object-attributes-dataset/

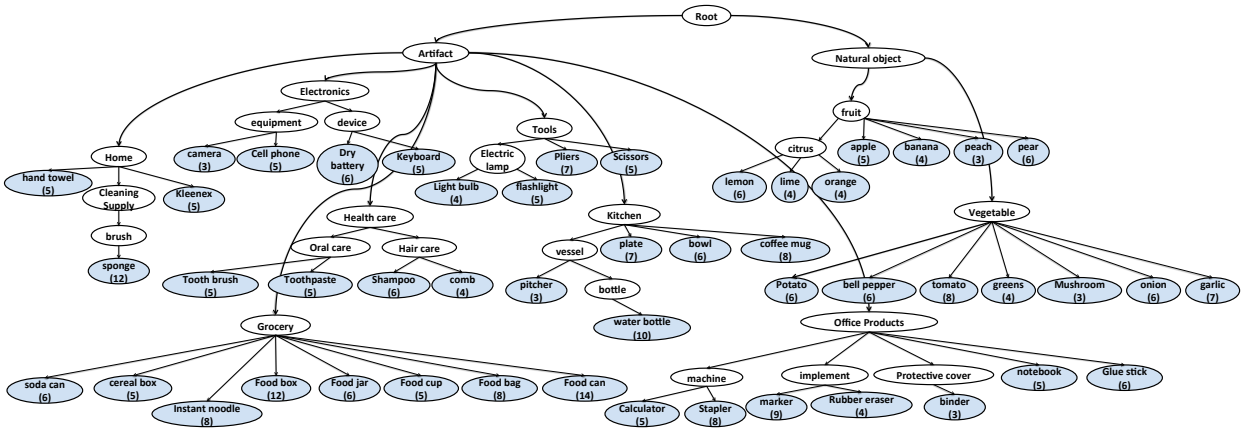


Figure 3.4: Hierarchy used in this chapter. The numbers in parentheses represent the number of different object instances within each leaf node.

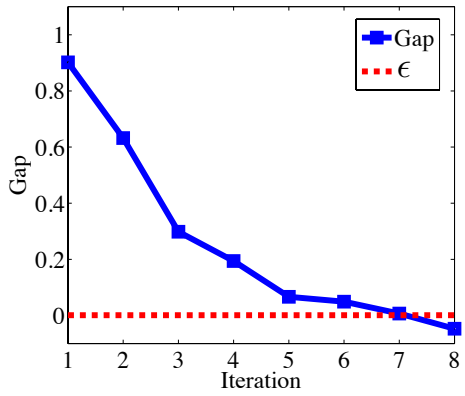
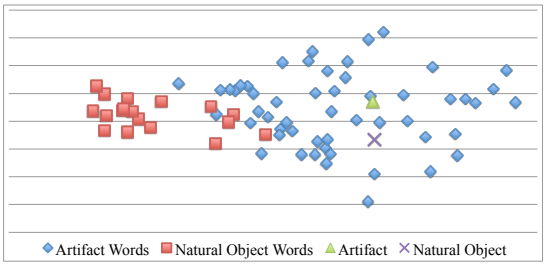


Figure 3.5: Convergence rate of the cutting-plane method.

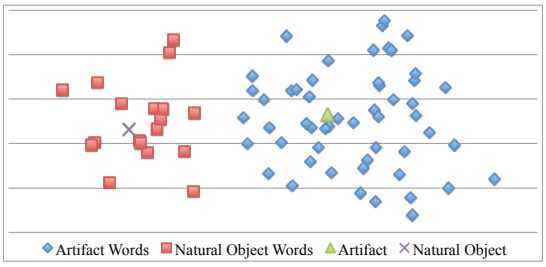
3.6.2 Name Learning

We first examined the convergence rate of the cutting-plane method for learning the tree-based word representation. For a name hierarchy with 72 words (Figure 3.4), there are $O(72^2)$ constraints in total. Figure 3.5 shows that the proposed cutting-plane algorithm converges to the given tolerance ϵ (set to be $1e^{-5}$ in our experiment) after only 8 iterations. This indicates that the cutting-plane method is efficient and can be scaled to even larger data sets.

We next illustrate that the proposed name learning method captures name hierarchy information.



(a) Word vector representation without name learning.



(b) Word vector representation after name learning.

Figure 3.6: 2-D projection of all 72 word vectors.

Recall that each word is originally represented by a vector $\boldsymbol{v}$ extracted using a large corpus of web documents. Figure 3.6 (a) shows the 2-D projection (via PCA) of the original 72 word vectors. All words belonging to the class of natural objects are marked as red squares, and words belonging to the class of artifacts as blue diamonds. The words "Artifact" and "Natural Object" are highlighted using a purple cross and a green triangle so that one can see how other words are related to their ancestors using the original feature representation.

After the metric learning, each $\boldsymbol{v}$ is transformed to a new vector $A\boldsymbol{v}$, where A is found via Cholesky factorization ($W = A^T A$). The learned feature vectors are visualized in Figure 3.6 (b).

In Figure 3.6, we see that Artifact words and Natural Object Words are not well separated, and the word "Natural Object" is far away from other Natural Object words. This implies that using the original representation will result in various mistakes when reasoning about novel words, even at the highest level of the hierarchy. On the other hand, our method can separate the two different subcategories, as shown in Figure 3.6 (b). In the new vector space, the name of the category is located at the center of all words of the same category. This demonstrates that our method does learn a metric embedding the hierarchy information. This would enable our approach to identify nodes for new words more accurately.

3.6.3 Object Identification

Since our system can associate an unknown name with an existing name in the hierarchy, it can identify objects based on new object names. We test whether the ability to understand a new name can boost identification accuracy.

In this task, we test on identifying objects from scenes. Each scene has 6 objects, one of which is the target object. We also collect one command for each scene such that the target object can be identified based on the command. One example scene is given in Figure 3.3 (a). To identify an object based on the natural language command, we first extract name and appearance attributes using the Stanford parser [110] from the command. We then use the learned vector representation to determine a name path $\boldsymbol{p} = (w_{p_1}, \cdots, w_{p_L})$. Next, we compute the likelihood of both appearance attributes and the name path given each object in the scene and decide the target object according to Equation (3.11).

We compare the identification accuracies under three different settings of handling new names. 1) Ignoring an unknown name. In this case, only appearance attributes will be used for identification (IGNORE). 2) Mapping a new name to a name path (VEC_ORI) using the original vector representation. 3) Mapping a new name to a path using our hierarchy-adapted word vectors (VEC_HIE).

Table 3.1 summarizes the results. As can be seen from the table, using the original vectors to map new names does not achieve higher identification accuracy than ignoring unknown names. On the other hand, adapting the word vectors to the name hierarchy improves the identification accuracy by an average margin of 7%. We also evaluate only on identification tasks in which a new object name is used, and the improvement is about 11%. Overall, the whole identification system achieves 74% accuracy using the proposed name learning method. This is remarkable since 50% of the objects in each scene are not in the training set, including many cases in which even the object of interest and the name used by the person are novel.

As a side note, the improvements for Student2's tasks are more significant than for Student1's. An inspection of the annotations revealed that Student2 used more names than appearance attributes to refer to objects, thereby providing more opportunities for our hierarchy to help with the identification

Table 3.1: Identification accuracy on 600 scenes with 6 objects.

Methods		IGNORE	VEC_ORI	VEC_HIE
Accuracy(%)	Student 1	70.67	68.00	**73.67**
	Student 2	63.67	66.70	**75.00**
	AVE	67.17	67.35	**74.34**

task.

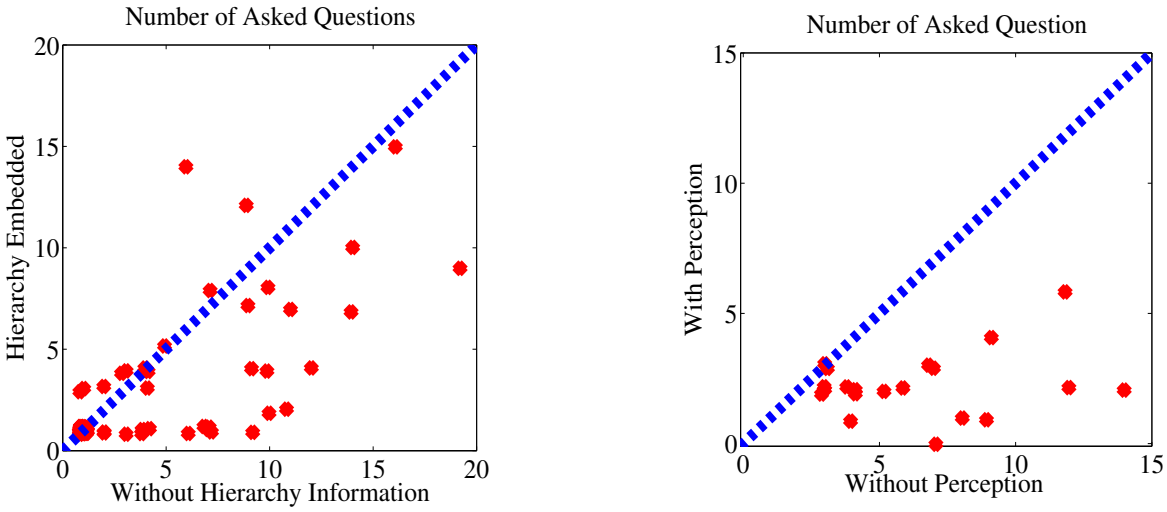
3.6.4 Locating New Names via Questions

Our name learning method can map a new name into an existing node in the name hierarchy more accurately than the baseline method. However, it still generates some false predictions. To correct these mistake, our system interacts with humans and asks clarification questions to figure out the correct node for the name.

In this experiment, we will test on inserting all the new names used by the two students into the WordNet hierarchy. We count the number of questions asked to find the correct path for the new names as the evaluation criterion. Three methods are compared: Method1 uses the original word vector representation, while Method2 used the hierarchy-embedded vector representation output by our metric learning method (both methods are described in Section 3.5.1). Method3 was the same as Method2 but with extra perception information, as described in Section 3.5.2.

The comparison between Method1 and Method2 is shown as the scatter plot in Figure 3.7 (a). For most names, the identification system with the learned representation asks fewer questions than the one without name learning. The result suggests that our name learning captures the hierarchy semantic of names and could be helpful to better understand the new names.

We also checked whether the perception information helps to reduce the number of questions further. Figure 3.7 (b) shows the scatter plot comparing Method2 and Method3 for hard name tasks, which require more than 1 question by Method2. As can be seen, using perceptual information



(a) Method1 vs. Method2

(b) Method2 vs. Method3

Figure 3.7: Scatter plot of number of questions asked until the correct location of an unknown name is determined.

dramatically decreases the number of necessary questions.

To further shed light on the difference between the three methods, we intensively examined which questions each method asked for two hard names. In Example 1, Student1 was given the scene in Figure 3.3 (a). She said "Give me the yellow clipper." The new name "clipper" was a synonym for "pliers" in this identification context. 10 questions were asked by Method1 in total while 9 questions were asked by Method2. "Food can" was asked by Method2 because clippers can be used to open food cans and appear in a similar document context frequently, resulting in similar word frequency vectors. Using perception information, Method3 can detect that there is no food can, lightbulb, electric lamp or scissors in the given scene. So, only 2 questions were asked before finding out that "clipper" is a synonym for "pliers".

The second example scene is shown in Figure 3.3 (b). The student given the task said, "Pick up the Pepsi." The new name "Pepsi" is a new instance name of "soda can". Example 2 shows the behavior of the different methods. The identification system asked 14 questions to find out the right path for "Pepsi" using Method1. Method1 asked about many unrelated names, such as "Kleenex"

	Method1	Method2	Method3
1	food cup	food can	grocery
2	grocery	grocery	pliers
3	artifact	artifact	
4	vessel	home product	
5	kitchen supply	lightbulb	
6	flashlight	electric lamp	
7	electric lamp	tool	
8	tool	scissors	
9	scissors	pliers	
10	pliers		

Example 1: Questions asked for the new name *clipper*.

	Method1	Method2	Method3
1	potato	onion	grocery
2	vegetable	vegetable	instant noodles
3	natural object	natural object	
4	kleenex	food can	
5	home product	grocery	
6	artifact	instant noodles	
7	coffee mug	soda can	
8	kitchen supply		
9	instant noodles		
10	grocery		
11	cereal box		
12	food cup		
13	food box		
14	soda can		

Example 2: Questions asked for the new name *pepsi*.

and “home product”, while Method2 quickly found out that “Pepsi” is more related to “grocery” by using the hierarchy information. Only 7 questions were asked in this case. Using attribute and object classifiers, the system filtered out “onion”, “vegetable”, “natural object” and “food can”, which did not exist in the scene. It also found that “Pepsi” is not a synonym for “soda can” because there is still uncertainty in the scene when using only “soda can” to refer to the target. Our system (Method 3) found that “Pepsi” is a new instance of “soda can” after asking 2 questions.

3.7 Conclusion

A robot operating in an indoor environment and interacting with people will frequently face new objects and new names used by people to refer to new objects. Therefore, it must be able to learn new objects and names as they occur during task execution. Toward this goal, we propose a new object recognition system for identifying objects based on natural language referrals with new names. The system organizes names into a semantic hierarchy of concepts. Each node of the hierarchy contains several names of similar meaning. When the system encounters a new name, it infers the node to which it should associate the new name. The target node is either a synonym or hypernym of the new name. Our system could use the classifier of the existing node to recognize the target objects. The system needs to find out the target node in the hierarchy such that the new name is semantically similar to the node and all its ancestor nodes. We propose a metric learning method to measure the similarities between words. The method can capture the semantic hierarchy information.

We perform extensive experiments to validate the system. The experimental results show that using the hierarchy-embedded word vector representation achieves significantly better reasoning about unknown names than using the purely text-based word vectors. The experiments also demonstrate that the overall system achieves 74% identification accuracy for scenes containing six objects, half of which are unknown. Our experiments also show that the proposed approach can determine the correct location of the new name in the hierarchy by asking the user only a small number of questions. Further, combining perceptual information with name reasoning significantly reduces the number of necessary questions.

One limitation of the current system is that it allows the adding of the leaf nodes only into an existing hierarchy, assuming that the hierarchy could be initialized with an existing hierarchy (such as WordNet). However, robots may need to insert an internal node into the WordNet hierarchy, or even build a hierarchy from scratch. How to create hierarchies from scratch will be the topic of the next chapter.

Chapter 4

BUILDING SEMANTIC HIERARCHIES VIA CROWDSOURCING

4.1 Introduction

Hierarchies of concepts and objects are useful across many real-world applications and scientific domains. Online shopping portals, *e.g.*, Amazon [4] and BestBuy [9], use product catalogs to organize their products into a hierarchy, aiming to simplify the task of search and navigation for their customers. Sharing the goal of organizing objects and information, hierarchies are prevalent in many other domains, such as in libraries to organize books [57] or web portals to organize documents by topics. Concept hierarchies also serve as a natural semantic prior over concepts, helpful in a wide range of Artificial Intelligence (AI) domains, such as natural language processing [13], computer vision [50, 121] and robotics [184].

Task-dependent hierarchies are expensive and time-consuming to construct. They are usually built in a centralized manner by a group of domain experts. This process makes it infeasible to create separate hierarchies for each specific domain. On the other hand, in the absence of such specific hierarchies, many applications use a general-purpose, pre-built hierarchy (for example, WordNet [69]) that may be too abstract or inappropriate for specific needs. An important question in this context is thus, *How can we cost-efficiently build task-dependent hierarchies without requiring domain experts?*

Attempts to build hierarchies using fully automated methods [11] have failed to capture the relationships between concepts as perceived by people. The resulting hierarchies perform poorly when deployed in real-world systems. With the recent popularity of crowdsourcing platforms, such as AMT, efforts have been made to employ non-expert workers (the crowd) at scale and low cost to build hierarchies guided by human knowledge. Chilton et al. propose the CASCADE workflow, which converts the process of building a hierarchy into the task of multi-label annotation for objects.

However, acquiring multi-label annotations for objects is expensive and could prove uninformative for creating hierarchies. This leads to the second question, *How can we actively select simple and useful questions that are most informative to the system while minimizing the cost?*

Most existing methods (including CASCADE) as well as methods employing domain experts usually generate only a single hierarchy that aims to best explain the data or the relationships among the concepts. This ignores the natural ambiguity and uncertainty that may exist in the semantic relationships among the concepts, leading to the third question, *How can we develop probabilistic methods that can account for this uncertainty in the process of building the hierarchy?*

In this chapter, we propose a novel system for building task-dependent hierarchies. The proposed system aims to tackle the three questions posed above. It gathers knowledge from crowdsourcing workers to build hierarchies. However, differs significantly from existing crowdsourcing-based methods regarding the type of questions asked. It directly asks questions regarding the structure of the hierarchy. To maintain the uncertainty over hierarchies as well as capture the noise introduced by non-expert workers' answers, it maintains a distribution over hierarchies instead of keeping the one best hierarchy. We deploy a compact model to parameterize the distribution and provide efficient methods to do inference. We also develop an efficient algorithm to update the model parameters based on the answers given by the crowdsourcing workers and prove convergence guarantees for the method. Further, we show several extensions of the model, such as asking informative questions using an active query method, inserting new nodes into the hierarchy, and identifying synonyms. Finally, we provide extensive empirical evaluations of the approach on synthetic problems, as well as on real-world domains with data collected from AMT workers, to demonstrate the broad applicability of our system.

The rest of this chapter is organized as follows. We discuss related work in Section 4.2. In Section 4.3, we describe how to model distributions over hierarchies and how to do inference efficiently. We present a method for updating the model and prove convergence guarantees for the method in Section 4.4. In Section 4.5, we discuss several implementation details and extensions of the algorithm. We present extensive evaluations in Section 4.6. Finally, we discuss conclusions and directions for future work in Section 4.7. We include our proofs and more experimental results in

the Appendix A.

4.2 Related Work

Hierarchies for Artificial Intelligence

Semantic hierarchies of concepts have been widely deployed for solving many AI tasks, *e.g.*, supporting various Natural Language Processing (NLP) tasks, such as disambiguating word sense in text retrieval [202, 164], information extraction [13], machine translation [111], and others. Meanwhile, hierarchies of object classes [221, 140, 209, 152] are also used as prior knowledge in the Computer Vision (CV) community for transferring knowledge between categories. The applications of object class hierarchies in CV include improving object categorization [165], scaling image classification [50, 52, 121], improving efficiency of image annotation [55], *etc.* Most of this work uses established hierarchies in their methods and does not consider building hierarchies suitable for their tasks. However, the quality of the hierarchies can influence the performance of these methods significantly.

Building Semantic Hierarchies

Building hierarchies of good quality is, therefore, an important research topic. The traditional way to create hierarchy is to hire a small group of experts to build the hierarchy in a centralized manner [69], which is expensive and time-consuming. Therefore, researchers have developed autonomous or semi-autonomous methods to do this task. Many of these methods tackle the problem by creating hierarchical clusterings of concepts. For instance, Sivic et al. and Bart et al. use visual feature similarities to perform clustering. However, visually similar concepts are not necessarily similar in semantics. Hence, the hierarchies built purely based on vision information do not match the semantics in many cases. There are also text-based methods [11, 12], which define the similar between pairs of words based on their co-occurrence in a large corpus of text. However, the existing techniques produce low-quality hierarchies since they lack the common sense and ability to understand language.

Ontology Learning System

Hierarchy creation is also related to ontology learning from the web [30, 208, 31]. The goal of ontology learning is to extract entities and relationships, that explain the world. Extracting hierarchy relationships (taxonomy) is a task in ontology learning [69, 180]. Some methods [180] focus on extending or refining established hierarchies using other resources, e.g. Wikipedia. However, their output is not varied enough as it is limited by the coverage of the resources. Several other methods [86, 174, 90] use lexical patterns/templates to mine hypernym-hyponym relations from text corpora. However, these methods rely on accurate extraction of lexical patterns and therefore do not generalize well to new domains. Other popular type of methods [29, 73, 128, 114] use general text corpus to define the hypernym-hyponym relation to build hierarchies. These methods usually do not perform well because they lack common sense [208].

It is worth noting that these ontology learning techniques focus mostly on coverage rather than accuracy. The knowledge, including hierarchies, extracted by these approaches are typically not very accurate. Many famous systems, *e.g.*, the Never-Ending Language Learner [31], still require humans to clean up the systems regularly to keep the semantic drifts in the systems under control. Moreover, the knowledge of categories or relations in the web is usually long-tail. For some rare or emerging categories, it is difficult to gather enough statistics to extract useful knowledge.

Knowing that humans are good at using common sense and learning new concepts, the involvement of humans in ontology learning remains highly necessary and desirable [208]. The new trend of ontology learning of rare categories is to involve humans in the early stage [156] of building the systems, rather than relying on them to perform post processing on the knowledge bases.

Crowdsourcing Structure Knowledge

The popularity of crowdsourcing platforms has made cheap human resources available for building hierarchies. For example, CASCADE [37] uses multi-label annotations for items and deploys label co-occurrence to generate a hierarchy. DELUGE [23] improves the multi-label annotation step in CASCADE using decision theory and machine learning to reduce the labeling effort. However, for

both pipelines, co-occurrence of labels does not necessarily imply a connection in the hierarchy. Furthermore, both methods can build only a single hierarchy and do not consider the uncertainty naturally existing in hierarchies.

Mortensen et al. use crowdsourcing to verify an existing ontology. Their empirical results demonstrate that non-expert workers can verify structures within a hierarchy built by domain experts. Inspired by their insights, it is possible to gather information about the hierarchy structure by asking simple true-or-false questions about the "ascendant-descendant" relationship between two concepts. In our work, we propose a novel method of hierarchy creation based on asking such questions and fusing information from the answers together.

4.3 Modeling Distribution over Hierarchies

The goal of our approach is to learn a hierarchy over a domain of concepts, using input from non-expert crowdsourcing workers. Estimating hierarchies through crowdsourcing is challenging since answers given by workers are inherently noisy. Further, even if every worker gives her/his best possible answer, concept relationships might be ambiguous, and there may be no single hierarchy that consistently explains all the workers' answers. We deal with these problems by using a Bayesian framework to estimate probability distributions over hierarchies rather than determining a single, best guess. This allows our approach to represent uncertainty due to noisy, missing, and possibly inconsistent information. Our system interacts with crowdsourcing workers iteratively while estimating the distribution over hierarchies. At each iteration, the system picks a question related to the relationship between two concepts in the hierarchy, presents it to multiple workers on a crowdsourcing platform, and then uses the answers to update the distribution. The system keeps asking questions until a stopping criterion is reached. In this work, we set a threshold for the number of questions asked.

4.3.1 *Modelling Distributions over Hierarchies*

The key challenge for estimating distributions over hierarchies is the huge number of possible hierarchies, or trees, making it intractable to directly represent the distribution as a multinomial. Consider the number of possible hierarchies of N concepts is $(N+1)^{N-1}$ (we add a fixed root node to the concept set), which results in 1,296 trees for 5 concepts, but already $2.3579e+09$ trees for only 10 concepts.

We will now describe how to represent and estimate distributions over such a large number of trees. Assume that there are N concept nodes indexed from 1 to N, a fixed root node indexed by 0, and a set of possible directed edges $\mathcal{E} = \{e_{0,1}, \ldots, e_{i,j}, \ldots, e_{N,N}\}$ indexed by (i,j), where $i \neq j$. A hierarchy $T \subset \mathcal{E}$ is a set of N edges, which form a valid tree rooted at the 0-th node (we use the terms hierarchy and tree interchangeably). All valid hierarchies form the sample space $\mathcal{T} = \{T_1, \ldots, T_M\}$. The prior distribution π^0 over $\mathcal{T}$ is set to be the uniform distribution.

Due to the intractable number of trees, we use a compact model to represent distributions over trees

$$P(T|W) = \frac{\prod_{e_{i,j} \in T} W_{i,j}}{Z(W)}, \tag{4.1}$$

where $W_{i,j}$ is a non-negative weight for the edge $e_{i,j}$, and $Z(W) = \sum_{T' \in \mathcal{T}} \prod_{e_{i,j} \in T'} W_{i,j}$ is the partition function.

4.3.2 *Inference*

Using the compact model defined in (4.1), inference is very efficient. We list the three most useful results in this section.

Computation of Partition Function

First, the partition function $Z(W)$ can be analytically computed utilizing the Matrix-Tree Theorem [197]. We define the Laplacian matrix $L(W) \in \mathcal{R}^{N \times N}$ of W, for $h, m = 1 \ldots N$

$$L_{h,m}(W) = \begin{cases} \sum_{h'=1}^{N} W_{h',m}, \text{ if } h = m \\ -W_{h,m}, \text{ otherwise,} \end{cases} \tag{4.2}$$

and another matrix

$$\hat{L}(W) = L(W) + \text{diag}(W_{0,\cdot}), \tag{4.3}$$

where $\text{diag}(\boldsymbol{v})$ is the diagonal matrix with the vector $\boldsymbol{v}$ on its diagonal. Using the Matrix-Tree Theorem [197] and its preposition [113], the partition function $Z(W)$ is given by $Z(W) = |\hat{L}(W)|$.

Computation of Marginals

Second, we can also analytically compute the marginal probabilities over the edges, *i.e.*, $P(e_{i,j})$. It is true that the log partition-function and the marginals have the following relation

$$P(e_{i,j}) = \frac{1}{W_{i,j}} \frac{\partial \log Z(W)}{\partial W_{i,j}}.$$

Using the fact that $Z(W) = |\hat{L}(W)|$, it can be shown that

$$P(e_{0,m}) = W_{0,m} \left[\hat{L}^{-1}(W)\right]_{m,m},$$

and, for $h > 0$, the marginals are given by

$$P(e_{h,m}) = W_{h,m} \left[\hat{L}^{-1}(W)\right]_{m,m} - (1 - \delta_{h,m}) W_{h,m} \left[\hat{L}^{-1}(W)\right]_{m,h},$$

where $\delta_{h,m}$ is the Kronecker delta.

Finding the MAP Tree

Third, finding the tree with the maximum probability equals

$$T^* = \arg\max_{T \in \mathcal{T}} \prod_{e_{i,j} \in T} W_{i,j} = \arg\min \sum_{e_{i,j} \in T} -\log(W_{i,j}).$$

This is the same problem as finding the minimum spanning tree in a directed graph, which can be implemented efficiently using Edmond's Algorithm [38, 60].

4.4 Approach: Updating Distributions over Hierarchies

Our system maintains a posterior distribution over hierarchies. Given a sequence of questions regarding the structure of the target hierarchy $q^1, \ldots, q^{(t)}, \ldots$, along with the answers $a^1, \ldots, a^{(t)}, \ldots$, the posterior $P(T|W^{(t)})$ at time t is obtained by Bayesian inference

$$P(T|W^{(t)}) \propto P(T|W^{(t-1)}) f(a^{(t)}|T). \tag{4.4}$$

Hereby, $f(a^{(t)}|T)$ is the likelihood of obtaining answer $a^{(t)}$ given a tree T, specified below to simplify the notation.

So far, we have not made any assumptions about the form of questions asked. Since our system works with non-expert workers, the questions should be as simple as possible. As discussed above, we resort to questions that only specify the relationship between pairs of concepts. We will discuss different options in the following sections.

4.4.1 Edge Questions

Since a hierarchy is specified by a set of edges, one way to ask questions would be to ask workers about immediate parent-child relationships between concepts, which we call *edge questions*. Answers to edge questions are highly informative since they provide direct information about whether there is an edge between two concepts in the target hierarchy.

Let $e_{i,j}$ denote the question of whether there is an edge between node i and j, and $a_{i,j} \in \{0, 1\}$ denote the answer for $e_{i,j}$. $a_{i,j} = 1$ indicates that a worker believes there is an edge from node i to j; otherwise, there is no edge. The likelihood function for edge questions is defined as follows

$$f(a_{i,j}|T) = \begin{cases} (1-\gamma)^{a_{i,j}}\gamma^{1-a_{i,j}}, \text{ if } e_{i,j} \in T \\ \gamma^{a_{i,j}}(1-\gamma)^{1-a_{i,j}}, \text{ otherwise} \end{cases}, \tag{4.5}$$

where γ is the noise rate for wrong answers. Substituting (4.5) into (4.4) leads to an analytic form to update edge weights

$$W_{i',j'}^{(t)} = \begin{cases} W_{i',j'}^{(t-1)}(\frac{1-\gamma}{\gamma})^{(2a_{i,j}-1)}, \text{ if } i' = i \wedge j' = j \\ W_{i',j'}^{(t-1)}, \text{ otherwise} \end{cases}. \tag{4.6}$$

An edge question only affects weights for that edge. Unfortunately, correctly answering such questions is difficult and requires global knowledge of the complete set (and granularity) of concepts. For instance, while the statement "Is *orange* a direct child of *fruit* in a food item hierarchy?" might be true for some concept sets, it is not correct in a hierarchy that also contains the more specific concept *citrus fruit* since it separates *orange* from *fruit* (see also Figure 4.4).

4.4.2 Path Questions

To avoid the shortcomings of edge questions, our system resorts to asking less informative questions relating to general, ascendant-descendant relationships between concepts. These *path questions* provide information only about the existence of directed paths between two concepts and are thus, crucially, independent of the set of available concepts. For instance, the path question "Is *orange* a type of *fruit*?" is true independent of the existence of the concept *citrus fruit*. While such path questions are easier to answer, they are more challenging to use when estimating the distribution over hierarchies.

Let $p_{i,j}$ denote a path question and $a_{i,j} \in \{0, 1\}$ be the answer for $p_{i,j}$. $a_{i,j} = 1$ indicates that a

worker believes there is a path from node i to j. The likelihood function is

$$f(a_{i,j}|T) = \begin{cases} (1-\gamma)^{a_{i,j}}\gamma^{1-a_{i,j}}, \text{ if } p_{i,j} \in T \\ \gamma^{a_{i,j}}(1-\gamma)^{1-a_{i,j}}, \text{ otherwise} \end{cases}, \tag{4.7}$$

where $p_{i,j} \in T$ simply checks whether the path $p_{i,j}$ is contained in the tree T.

Unfortunately, the likelihood function for path questions is not conjugate of the prior. Therefore, there is no analytic form to update weights. Instead, we update the weight matrix by performing approximate inference. To be more specific, we find a W^* by minimizing the KL-divergence [117] between $P(T|W^*)$ and the true posterior

$$W^* = \arg\min_W KL(P(T|W^{(t)})\|P(T|W)) \tag{4.8}$$

It can be shown that minimizing the KL-divergence can be achieved by minimizing the following loss function

$$L(W) = -\sum_{T\in\mathcal{T}} P(T|W^{(t)}) \log P(T|W). \tag{4.9}$$

Directly computing (4.9) involves enumerating all trees in $\mathcal{T}$ and is therefore intractable. Instead, we use a Monte Carlo method to estimate (4.9) and minimize the estimated loss to update W. To be more specific, we will generate i.i.d. samples $\widetilde{\mathcal{T}} = (T_1, \ldots, T_m)$ from $P(T|W^{(t)})$, which defines an empirical distribution $\tilde{\pi}^{(t)}$ of the samples, with estimated loss

$$L_{\tilde{\pi}}(W) = -\sum_{T\in\widetilde{\mathcal{T}}} \tilde{\pi}(T) \log P(T|W), \tag{4.10}$$

the negative log-likelihood of $P(T|W)$ under the samples.

Sampling Hierarchies from the Posterior

Given a weight matrix W, sampling hierarchies from the distribution defined in (4.1) can be achieved efficiently, for example, using a loop-avoiding random walk on a graph with W as the adjacency matrix [207]. Therefore, we can sample hierarchies from the prior $P(T|W^{(t-1)})$. Noticing that the posterior defined in (4.4) is a weighted version of $P(T|W^{(t-1)})$, we can generate samples for the empirical distribution $\tilde{\pi}$ via importance sampling, that is, by reweighing the samples from $P(T|W^{(t-1)})$ with the likelihood function $f(a^{(t)}|T)$ as importance weights.

Regularization

Since we get samples only from $P(T|W^{(t)})$, the estimate of (4.10) can be inaccurate. To avoid overfitting to the sample, we add an ℓ_1-regularization term to the objective function. We also optimize $\Lambda = \log W$ rather than W so to simplify notation. The final objective is

$$L^{\boldsymbol{\beta}}_{\tilde{\pi}}(\Lambda^{(t)}) = -\sum_{T\in\mathcal{T}} \tilde{\pi}^{(t)}(T) \log P(T|\Lambda^{(t)}) + \sum_{i,j} \beta|\lambda_{i,j}|. \tag{4.11}$$

Optimization Algorithm

We iteratively adjust Λ to minimize (4.11). At each iteration, the algorithm adds Δ to the original Λ, resulting in $\Lambda' = \Lambda + \Delta$. We optimize Δ to minimize an upper bound on the change in $L^{\beta}_{\tilde{\pi}}$, given by

$$\begin{aligned} L^{\boldsymbol{\beta}}_{\tilde{\pi}}(\Lambda') - L^{\boldsymbol{\beta}}_{\tilde{\pi}}(\Lambda) \leq \sum_{i,j} [&-\delta_{i,j}\tilde{P}(e_{i,j}) + \frac{1}{N} P(e_{i,j}|\Lambda)(e^{N\delta_{i,j}} - 1) \\ &+ \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|)] + C, \end{aligned} \tag{4.12}$$

where $\tilde{P}(e_{i,j}) = \sum_{T\in\mathcal{T}:e_{i,j}\in T} \tilde{\pi}(T)$ is the empirical marginal probability of $e_{i,j}$, and C is a constant w.r.t. $\delta_{i,j}$. The derivation is presented in Appendix A.

Minimizing the upper bound in (4.12) can be done by analysing the sign of $\lambda_{i,j} + \delta_{i,j}$. By some calculus, it can be seen that the $\delta_{i,j}$ minimizing (4.12) must occur when $\delta_{i,j} = -\lambda_{i,j}$, or when $\delta_{i,j}$ is either

$$\begin{aligned} &\frac{1}{N} \log \frac{(P(e_{i,j}) - \beta)}{P(e_{i,j}|\Lambda)}, \text{ if } \lambda_{i,j} + \delta_{i,j} \geq 0, \text{or} \\ &\frac{1}{N} \log \frac{(P(e_{i,j}) + \beta)}{P(e_{i,j}|\Lambda)}, \text{ if } \lambda_{i,j} + \delta_{i,j} \leq 0. \end{aligned}$$

This leaves three choices for each $\delta_{i,j}$ – we try out each and pick the one leading to the best bound. This can be done independently per $\delta_{i,j}$ since the objective in (4.12) is separable. The full algorithm to optimize Λ based on a query answer is given in Algorithm 2.

Algorithm 2 Weight Updating Algorithm

Input: $W^{(t-1)}$, an answer $a^{(t)}$, thr for stopping criterion
Non-negative regularization parameters β
Output: $W^{(t)}$ that minimizes (4.11)
Generate samples $T'_1, \ldots, T'_m$ from $P(T|W^{(t-1)})$
Use importance sampling to get empirical distribution $\tilde{\pi}$
Initialize $\Lambda^0 = \mathbf{0}$, $l = 1$.
repeat
    For each (i, j), set $\delta_{i,j} = \arg\min$ (4.12);
    Update $\Lambda^{(l)} = \Lambda^{(l-1)} + \Delta$;
    $l = l + 1$;
until $|\Delta| \leq thr$
return $W^{(t)} = \exp(\Lambda^{(l)})$

Theoretical Guarantee

Even though we minimize only a sequence of upper bounds, we prove (see Appendix) that Algorithm 2 in fact converges to the true maximum likelihood solution

Theorem 1. *Assume β is strictly positive. Then Algorithm 2 produces a sequence $\Lambda^{(1)}, \Lambda^{(2)}, \ldots$ such that*

$$\lim_{\ell \to \infty} L^{\beta}_{\tilde{\pi}}(\Lambda^{(\ell)}) = \min_{\Lambda} L^{\beta}_{\tilde{\pi}}(\Lambda).$$

Let $\hat{\Lambda}$ be the solution to Algorithm 2. We next show that if we generate enough samples m, the loss of the estimate $\hat{\Lambda}$ under the true posterior will not be much higher than that obtained by any distribution of the form (4.1).

Theorem 2. *Suppose m samples $\tilde{\pi}$ are obtained from any tree distribution π. Let $\hat{\Lambda}$ minimize the regularized log loss $L^{\beta}_{\tilde{\pi}}(\Lambda)$ with $\beta = \sqrt{\log(N/\delta)/(m)}$. Then, for every Λ it holds with probability at least $1 - \delta$ that*

$$L_{\pi}(\hat{\Lambda}) \leq L_{\pi}(\Lambda) + 2\|\Lambda\|_1 \sqrt{\log(N/\delta)/m}$$

Theorem 2 shows that the difference in performance between the density estimate computed by minimizing w.r.t. $L^{\beta}_{\tilde{\pi}}$ and w.r.t. the best approximation to the true posterior becomes small rapidly as the number of samples m increases.

4.5 Implementation

4.5.1 Active Query Selection

At each interaction with workers, the system needs to pick a question to ask. The naive approach would be to pick questions randomly. However random questions are usually not very informative since they get mostly *No* answers, while *Yes* answers are more informative about the structure. Instead of randomly picking question, we propose to select the question p^* that maximizes information gain over the current hierarchy distribution $\pi^{(t)}$

$$p^* = \arg\max_{p_{i,j}} \min(H(\tilde{\pi}^{(t+1)}_{p_{i,j},1}), H(\tilde{\pi}^{(t+1)}_{p_{i,j},0})) \tag{4.13}$$

where $H(\cdot)$ is the entropy and $\tilde{\pi}^{(t+1)}_{p_{i,j},a_{i,j}}$ is the posterior distribution over trees after knowing the answer for $p_{i,j}$ as $a_{i,j}$. Note that this criterion chooses the question with the highest information gain using the *less* informative answer, which we found to be more robust than using the expectation over answers. Since we cannot compute the entropy exactly due to the size of the sampling space, we reuse the trees from the empirical distribution $\tilde{\pi}$ to estimate information gain.

Another intuition is that we do not want to ask the same question too many times if there is only a little uncertainty over the workers' answers. One way of achieving this goal is to set up a maximum number of selections such that no question could be selected more than the maximum number of times. However, the maximum number must be question-dependent. To better model the question-dependent uncertainty, we model the distribution of the probability of getting a positive answer from a worker using a Beta distribution for each path question. Its parameters are initialized as $(1, 1)$ and updated as more answers to the path questions are collected from workers. We compute the entropy of the Beta distribution and combine it with (4.13) to evaluate the information main in a path question.

4.5.2 Adding New Nodes to the Hierarchy

New concepts will sometimes be introduced to a domain. In this case, how should the new concept be inserted into an existing hierarchy? A wasteful way would be to re-build the whole hierarchy distribution from scratch using the pipeline described in the previous sections. Alternatively, one might consider adding a row and column to the current weight matrix and initializing all new entries with the same value. Unfortunately, uniform weights do not result in uniform edge probabilities and thus do not correctly represent the uninformed prior over the new node's location.

We instead propose a method to estimate the weight matrix W that correctly reflects uncertainty over the current tree and the location of the new node. After building a hierarchy of $N+1$ nodes, the learned weight matrix is denoted as W_N. Now we want to insert a new node into the hierarchy. Using basic rules of probability, it is true that

$$P(T'|W_N) = \sum_T P(T'|T)P(T|W_N), \tag{4.14}$$

where T' is a hierarchy of $N+1$ nodes, and T has N nodes. $P(T'|T)$ is the transition probability from T to T'. Since there is no prior information about the position of the new node, $P(T'|T)$ is set to a uniform prior over all possible hierarchies of $N+1$ nodes by inserting one node into any possible location in T. Equation (4.14) represents a sample distribution that preserves the information of the previous distribution while not assuming anything about the new node. The new weight matrix with $N+1$, denoted as W_{N+1}, can be initialized as $W_{N+1}^{(0)}$ such that it represents a distribution that is close to that defined in (4.14). This can be implemented by minimizing the KL-divergence between the two distributions

$$W_{N+1}^{(0)} = \arg\min_{W_{N+1}} KL(P(T'|W_N)\|P(T'|W_{N+1})), \tag{4.15}$$

which indeed can be optimized using the same technique described in Section 4.4.2.

4.5.3 Identifying Synonyms

In some cases, two distinct names might represent the same concepts. This usually suggests that one node is a synonym for another. For example, "drink" is a synonym for "beverage". Because they have the same semantic meaning, there will be a loop between "drink" and "beverage" in the dependence relation. The loop breaks the assumption in our model that the target structure over nodes is a tree. We want to identify possible candidate synonym pairs, ask humans to verify whether the candidates are synonyms, then merge the real synonym nodes into a single node.

It is worth noting that there are many ways to detecting synonyms in the natural language processing literature, *e.g.*, [195, 214]. These methods rely on some similarity measure between a pair of words to detect synonym candidates. We are not trying to compete with these methods; rather, we are simply suggesting a way to identify synonyms to make our framework self-contained.

We propose a new way of measuring the similarity score for a pair of nodes i and j based on the answers to the path questions given by the crowdsourcing workers. We design the score based on the observation of inter-changeability of synonyms. If node j is a synonym for i, $a_{i,j'} = 1$ indicates $a_{j,j'} = 1$. We use these notations

$$P(a_{i,\bar{\boldsymbol{j}}}) = [P(a_{i,1}), \ldots, P(a_{i,j'}), \ldots, P(a_{i,N}))], \text{if } j' \neq i, j' \neq j,$$

and

$$P(a_{\bar{\boldsymbol{j}},i}) = [P(a_{1,i}), \ldots, P(a_{j',i}), \ldots, P(a_{N,i}))], \text{if } j' \neq i, j' \neq j.$$

Here $P(a_{i,j})$ is the probability of getting 1 for the path question $q_{i,j}$ based on workers' answers. $P(a_{i,\bar{\boldsymbol{j}}})$ and $P(a_{j,\bar{\boldsymbol{i}}})$ will have strong linear correlation with each other and vice versa. It is also expected that both $P(a_{i,j})$ and $P(a_{j,i})$ are high. Therefore, we define a simple score function to measure the similarity of node i and j as

$$\texttt{score}(i,j) = \texttt{cos}(P(a_{i,\bar{\boldsymbol{j}}}), P(a_{j,\bar{\boldsymbol{i}}})) \times \texttt{cos}(P(a_{\bar{\boldsymbol{j}},i}), P(a_{\bar{\boldsymbol{i}},j})) \times P(a_{i,j}) \times P(a_{j,i}), \quad (4.16)$$

where $\texttt{cos}(\cdot,\cdot)$ measures the cosine similarity between two vectors. We can use the maximum likelihood estimation for $P(a_{i,j})$.

We use (4.16) to rank all pairs of nodes in the hierarchy and ask workers to confirm whether two nodes are synonyms. We will merge true synonyms into a new node and update the weight matrix W accordingly.

For example, we want to merge node i and j into one node i'. To update W properly, we will need to know $P(e_{i',k})$ for the new node i'. There is an edge from i' to k if and only if there exists an edge from i or j to k. Because the two events are exclusive, we can derive that

$$P(e_{i',k}) = P(e_{i,k}) + P(e_{j,k}).$$

To estimate $P(e_{k,i'})$, the exclusive property does not exist. Fortunately, we can still use the Inclusion-Exclusion Principle to compute it as

$$P(e_{k,i'}) = P(e_{k,i}) + P(e_{k,j}) - P(e_{k,i}, e_{k,j}).$$

We prove the following proposition of the Matrix-Tree Theorem that $P(e_{k,i}, e_{k,j})$ can be computed analytically.

Proposition 1. *Given that $e_{i,j}$ and $e_{i,k}$ are two edges, the marginal likelihood $P(e_{i,j}, e_{i,k})$ is given as follows*

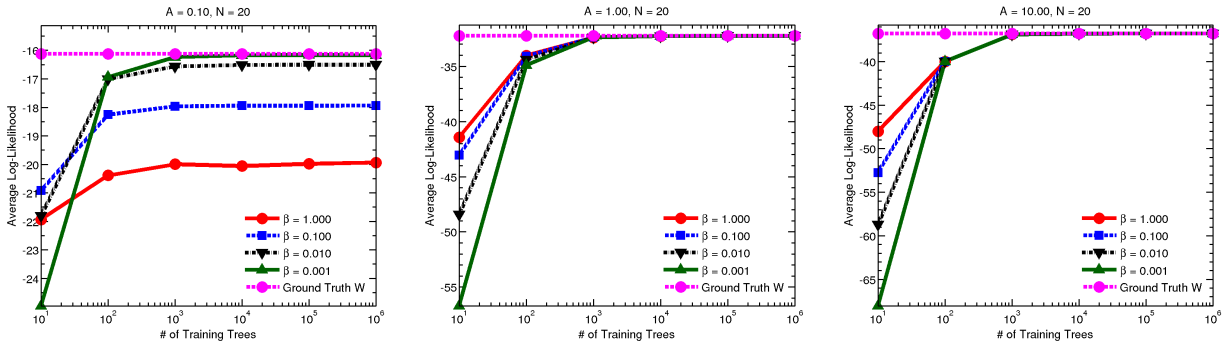
$$P(e_{i,j}, e_{i,k}) = P(e_{i,j})P(e_{i,k}) -$$
$$W_{i,j}W_{i,k} \begin{cases} [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{j,l}, \ \textit{if } i = 0 \\ -\left([\hat{L}^{-1}]_{i,i}\right)^2 + [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{k,i} - [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{j,i} + [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{k,j}, \ \textit{o.w.} \end{cases}$$

where $P(e_{i,j})$ is the marginal likelihood of edge $e_{i,j}$, and $\hat{L}$ is defined in equation(4.3).

Given the new marginal likelihood of edges, we can update the model parameter W using the same method we described in Section 4.4.2.

4.6 Experiments

We empirically evaluate our approach according to four criteria: (1) the performance of the proposed algorithms using simulation data, (2) a comparison with related methods using real-word data, (3)



(a) High concentration. (b) Medium concentration. (c) Low concentration.

Figure 4.1: Weight estimation performance for hierarchies with 20 nodes. X-axis is the number of samples given to the algorithm, and β is the regularization coefficient.

the ability to build hierarchies for diverse application domains;, and (4) the ability to build better task-dependent hierarchies.

4.6.1 Simulation

We start with experiments using simulation data to analyze the empirical performance and robustness of the proposed methods. The simulations allow us to know the ground truth trees, usually hard to learn for real-world experiments. Additionally, we can control the parameters and hence get insights into the algorithm. Moreover, the simulations also let us control the noise rate. Therefore, we can observe how robust our method is against different levels of noise.

Sample-Based Weight Estimation

Our first experiment evaluates the ability of Algorithm 2 to estimate a weight matrix based on samples generated from a distribution over trees. We first sample a "ground truth" weight matrix W; we next sample trees according to that weight matrix; we then estimate the weight matrix W^* using Algorithm 2; finally, we evaluate the quality of the estimated matrix. To evaluate, we sample an additional test set of trees from $P(T|W)$ and compute the average log-likelihood of these

trees given W^*, where $P(T|W^*)$ is defined in (4.1). For calibration, we also compute the average log-likelihood of these trees under the ground truth specified by $P(T|W)$.

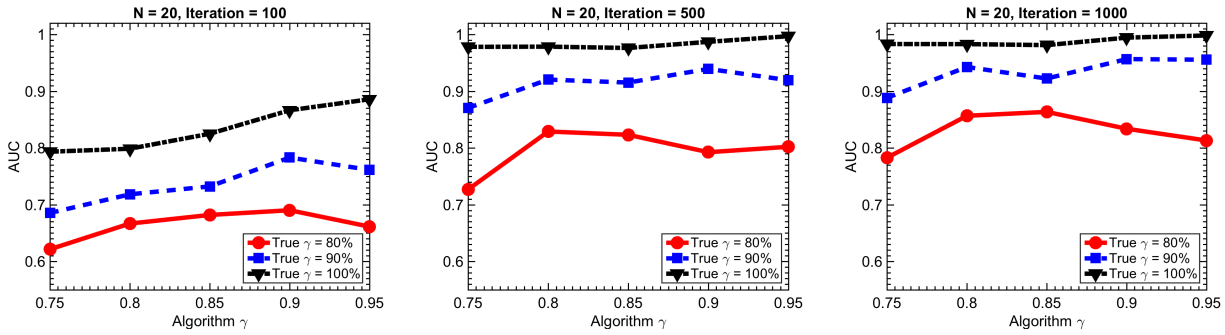
To sample W for trees with N nodes, we independently sample each column of W from an N-order Dirichlet distribution. Note that the "concentration" of W decides the complexity of the problem. High concentration indicates greatest probability mass on a small set of trees, while low concentration means the opposite. So, we vary the concentration to test the performance of the algorithm under different problem complexities. This is achieved by varying parameters A, where the parameters of the Dirichlet distribution is $A \times \mathbf{1}^T$, and $\mathbf{1} \in \mathcal{R}^N$. We test out three values of $A = (1, 10, 100)$. To give an intuition for the complexity of the investigated distributions, when sampling 1 million trees with 20 nodes according to one of the weight samples from a Dirichlet distribution with $A = 100$, we typically get about $900,000$ distinct trees. When we sample 1 million trees according to the W sample from a Dirichlet distribution with $A = 1$, there are only about $8,000$ unique trees.

Figure 4.1 shows the performance for $N = 20$ nodes using different values for the regularization coefficient β and sample size m. Each data point in the plots is an average over 100 runs on different weights sampled from a Dirichlet distribution. The red line on top is the ground truth log-likelihood. As can be seen, the algorithm always converges to the ground truth as the number of samples increases. With an appropriate setting for β, the proposed method requires about $10,000$ samples to achieve a log-likelihood that is close to the ground truth. We also tested different tree sizes N and got similar performance results (see Appendix A). Overall, we found that $\beta = 0.001$ works robustly across different N values and used that number in all the following experiments.

Sequential Update to Recover the Hierarchy

We next study the overall pipeline for recovering the correct hierarchy. Our pipeline works as follows. It selects a path question at each iteration to ask a crowdsourcing worker. It then updates the weight matrix W according to the answer using the equation (4.10). It repeatedly asks questions and updates W until the distribution converges.

To evaluate the ability of our technique to recover the correct hierarchy, we artificially generate



(a) Iteration 100 (b) Iteration 500 (c) Iteration 1000

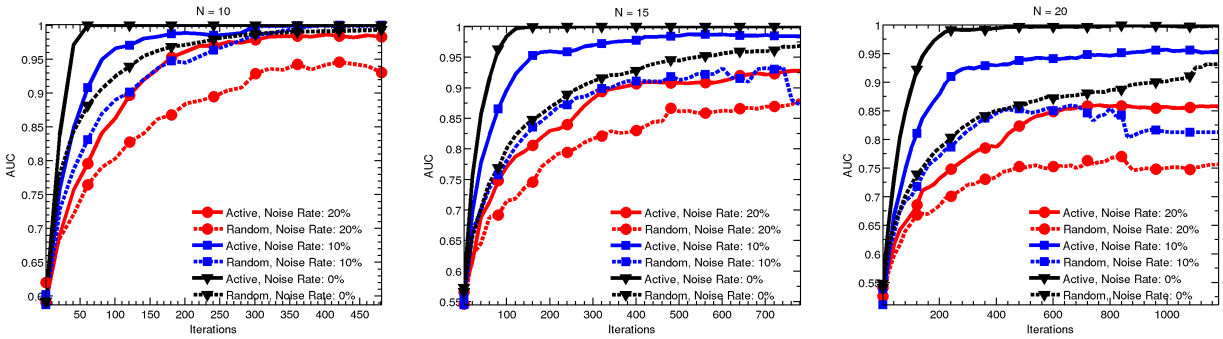
Figure 4.2: The proposed approach is robust with respect to noise rate γ.

ground truth hierarchies of different sizes. To simulate a worker's response to a query, we first check whether the query path is part of the ground truth tree and then flip the answer randomly using a pre-set noise rate γ.

To evaluate how well our approach estimates the ground truth tree, we use the marginal likelihood of the tree edges to compute the Area Under the Curve (AUC) using different thresholds on the likelihood to predict the existence or absence of an edge in the ground truth tree. The marginal likelihood of an edge, $P(e_{i,j}|W) = \sum_{T \in \mathcal{T}, e_{i,j} \in T} P(T|W)$, can be computed in closed form based on the conclusion of the Matrix Theorem [197]. We also tested different evaluation measures, such as the similarity between the MAP tree and the ground truth tree, and found them all to behave similarly.

Robustness to a Mismatched γ. Our model requires the noise rate γ, which is not known. In this experiment, we vary γ values used by the algorithm against three ground truth noise rates: 0%, 10%, and 20%. Figure 4.2 shows the performance of the proposed method on simulated trees with $N = 20$ nodes. The performance shows that using γ in the range 5% to 20% performs as good as knowing the true noise rate.

In general a larger γ gives a better convergence rate. Thus, we decided to set $\gamma = 10\%$ in all the following experiments.



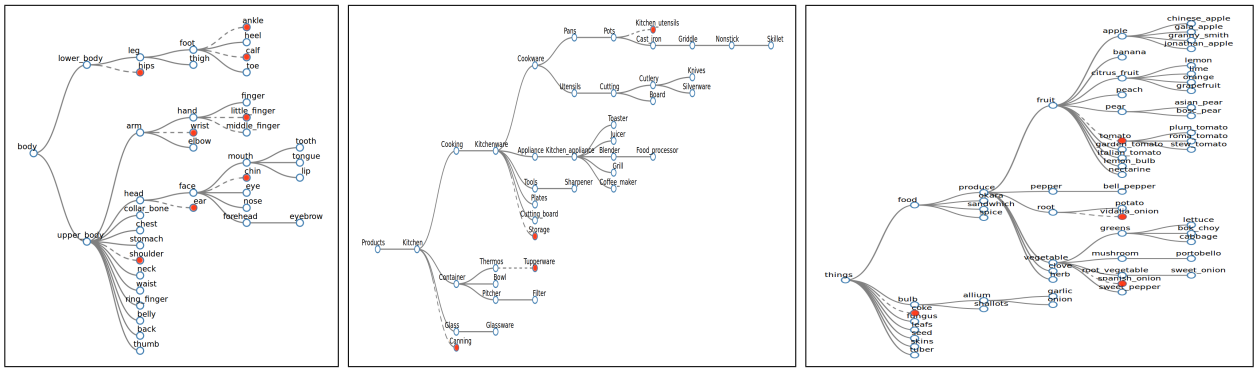
(a) $N = 10$ (b) $N = 15$ (c) $N = 20$

Figure 4.3: Experimental results comparing active query (solid lines) and random query (dashed lines) strategies for tree sizes ranging from 5 nodes (left), 10 nodes (center), and 15 (right), using three different noise rates (0, 10, 20) for answers to questions.

Active vs. Random Queries. We deploy an active query approach to select the most informative questions. We compare the performance of our active query strategy with two baseline methods: *Random* (picking uniformly random questions) and *RoundRobin* (round-robin over random orders).

Differently sized trees of nodes $\{10, 15, 20\}$ are tested to see how the method performs as the problem gets larger. We also test different noise rates, - including 0%, 10%, and 20% - to verify the robustness of the method. The number of samples for updating the weight matrix is fixed to $20,000$ across all experiments. For each setting, 20 different random ground truth trees are generated. The average results are reported in Figure 4.3. The X-axis is the number of questions asked, and the Y-axis is the AUC. $AUC = 1$ means that all edges of the ground truth tree have higher probabilities than any other edges according to the estimated distribution over trees.

As can be seen in the figures, active queries always recover the hierarchy more efficiently than their random counterparts (*i.e.*, queries are generated by randomly choosing a pair of nodes) when the noise rate is within a reasonable rage. If there is no noise in the answers, our approach always recovers the ground truth hierarchy despite the sample-based weight update. Note that, in the noise-free setting, the exact hierarchy can be perfectly recovered by querying all N^2 pairs of concepts. While the random strategy typically requires about twice that number to recover the



(a) Body parts (b) Amazon kitchen products (c) Food item

Figure 4.4: MAP hierarchies representing body parts, Amazon kitchen products, and food items (left to right). Red nodes indicate items for which the parent edge has high uncertainty (marginal probability below 0.75).

hierarchy, our proposed active query strategy always recovers the ground truth tree using less than N^2 samples.

As N gets larger, the difference between active and random queries becomes more significant. While our active strategy always recovers the ground truth tree, the random query strategy does not converge for trees of size 15 if the answers are noisy. This is due to insufficient samples when updating the weights and the fact that random queries are frequently answered with *No*, which provides little information. The active approach, on the other hand, uses the current tree distribution to determine the most informative query and generates more peaked distributions over trees, which can be estimated more robustly with our sampling technique. As an indication of this, for trees of size 15 and noise-free answers, 141 out of the 200 first active queries are answered with “yes”, while this is the case for only 54 random queries.

4.6.2 Real-World Experiments

Applications to Various Domains

These experiments provide examples demonstrating that our method can be applied to different tasks using AMT. The following pipeline is followed for all three application domains: collect the set of concepts; design the path question to ask; collect multiple answers for all possible path questions; estimate hierarchies using our approach. Collecting answers for all possible questions enabled us to test different settings and methods without collecting new data for each experiment. AMT is used to gather answers for path questions. The process for different domains is almost identical: We ask workers to answer "true-or-false" questions regarding a path in a hierarchy. Our method can consider the noise rate of workers. We estimate this by gathering answers from 8 workers for each question, then we take the majority vote as the answer and use all answers to determine the noise ratio for that question. Note that noise ratios not only capture the inherent noise in using AMT, but also the uncertainty of people about the relationship between concepts. Five different path questions are put into one Human Intelligence Task (HIT). Each HIT costs $0.04; average time for a worker to finish one HIT is about 4 seconds.

The process of building the hierarchies is divided into two consecutive phases. In the first phase, a distribution is built using a subset of the concepts. In the second phase, we use the process of inserting new concepts into the hierarchy until all concepts are represented. For the body part dataset, we randomly chose 10 concepts belonging to the first phase. For online Amazon shopping and RGBD object data, the initial set is decided by thresholding the frequency of the words used by workers to tag images (15 nodes for Amazon objects and 23 nodes for RGBD objects). The learned MAP hierarchies are shown in Figure 4.4.

Representing Body Parts. Here, we want to build a hierarchy to visualize the "is a part of" relationship between body parts. The set of body part words are collected using Google search. An example path question would be "Is *ear* a part of the *upper body*?". After asking 2,000 questions, its MAP tree is shown in the left panel of Figure 4.4. As can be seen, its overall structure agrees very

well with people's common sense of the human body structure. Some of the nodes in the tree are shown in red, indicating edges whose marginal probability is below 0.75. These edges also reflect people's uncertainty in the concept hierarchy. For example, it is not obvious whether *ear* should be part of *head* or *face*, the second most likely placement. Similarly, it is not clear whether *ankle* should be part of *foot* or *leg*, and whether *wrist* should be part of *arm* or *hand*.

An obvious mistake made by the system is that *ring finger* and *thumb* are connected to the *upper body* rather than the *hand*. This is caused by questions such as, "Is the *ring finger* part of the *arm*?", which only 1 out of 8 workers answered with yes. Hence, the concept of *ring finger* or *thumb* is not placed into a position below *arm*.

Online Shopping Catalogue. The second task is to arrange kitchen products taken from the Amazon website. There are some existing kitchen hierarchies, for example, the Amazon hierarchy [4]. However, the words used by Amazon (*e.g.*, "Tools-and-Gadgets") might prove confusing for customers. Therefore, we collected the set of words used by workers when searching for products. We give AMT workers images of products and ask them to write down words they would like to see in navigating kitchen products. Some basic preprocessing is done to merge plural and singular forms of the same words, remove obviously wrong words, and remove tags used less than 5 times by workers because they might be a particular person's nickname. We also remove the word "set" because it is used by workers to refer to a "collection of things" (*e.g.*, pots set, knives set) but may not be related to the type of products shown in the pictures. The path questions have the form, "Would you try to find *pots* under the category of *kitchenware*?" The learned MAP tree is shown in the middle panel of Figure 4.4.

Food Item Names. This experiment investigates learning a hierarchy of food items used in a robotics setting [122]; the goal is to learn names people use in a natural setting to refer to objects. Here, AMT workers were shown images from the RGBD object dataset [122] and asked to provide names they would use to refer to these objects. Some basic pre-processing was done to remove noisy tags and highly infrequent words. The path questions for this domain are of the form, "Is it

correct to say all *apples* are *fruits*?".

The MAP tree is shown in the right panel of Figure 4.4. Again, while the tree most often captures the correct hierarchy, high uncertainty items provide interesting insights. For instance, *tomato* is classified as *fruit* in the MAP tree but also has a significant probability of being a *vegetable*, indicating people's uncertainty, or disagreement, about this concept. Meanwhile, the crowd of workers could uncover very non-obvious relationships, such as *allium* is a kind of *bulb*.

Comparison to Existing Work

We now compare our method to the most relevant systems – DELUGE [23] and CASCADE [37] – which also use crowdsourcing to build hierarchies. CASCADE builds hierarchies based on multi-label categorization, and DELUGE improves the multi-label classification performance of CASCADE. We will thus compare to DELUGE.

We apply both methods to the evaluation dataset used by DELUGE [23]. This dataset has 33 labels that are part of the fine-grained entity tags [130]. The goal is to build hierarchies over the 33 labels with an extra root node. We use the WordNet hierarchy as the ground truth hierarchy to evaluate the performance of different methods. The evaluation metric is AUC, described in the previous section.

DELUGE builds hierarchies using the following process. It uses each label as a keyword to query many items from an established knowledge base, *e.g.*, Freebase [19]. Next, it randomly selects a subset of 100 items and labels them with multiple labels using crowdsourcing. It builds a hierarchy using hierarchical clustering with the label co-occurrence as the measurement of label similarity.

To classify items into multi-labels, both CASCADE and DELUGE ask crowdsourcing workers to vote for questions, which are binary judgments about whether an item belongs to a category (label). CASCADE selects questions in a round-robin manner, while DELUGE makes active query selections based on an information gain criterion. DELUGE also considers the label correlation when aggregating the votes and, therefore manages to do multi-label annotation more efficiently than CASCADE. We use the code and parameter settings provided by the authors of DELUGE in this experiment.

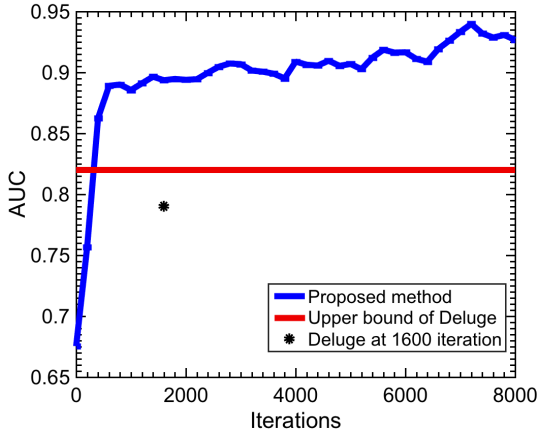
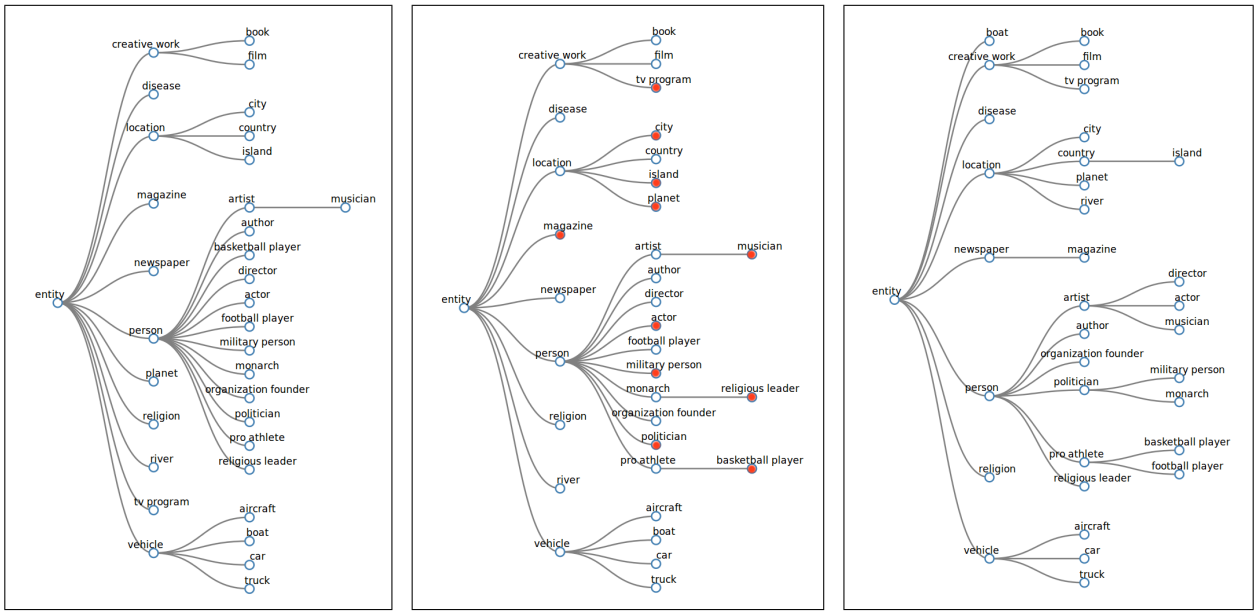


Figure 4.5: Our method performs significantly better than DELUGE on the same benchmark dataset.

We compare the performance of our method to DELUGE when different amounts of votes are asked (see Figure 4.5). For our method, we plot the AUC score for every ten iterations. For DELUGE, we plot two critical points: 1) using 1,600 votes, and 2) using 49,500 votes. For the first critical point, we pick 1,600 votes as suggested by the authors because DELUGE's performance saturates after that many votes. For the second critical point, we compute the results of using all the votes collected in the DELUGE data set to see the best performance of DELUGE. We set an upper bound vote for our approach. It will stop after asking for 8,000 votes because its performance becomes flat after that.

Using 1,600 votes, our method achieves a value of 0.82, which is slightly better than DELUGE, with an AUC of 0.79. However, DELUGE does not improve significantly beyond that point, reaching an AUC of 0.82 after 49,500 votes. Our approach, on the other hand, keeps on improving its accuracy and reaches an AUC of 0.97 after only 6,000 queries. This indicates that, using our approach, non-expert workers can achieve performance levels very close to those achieved by experts (AUC = 1).

We also visualize different hierarchies. Figure 4.6 (a) shows the ground truth hierarchy from WordNet; Figure 4.6 (b) shows the MAP hierarchy built by our method; and Figure 4.6 (c) shows the hierarchy built by DELUGE. The hierarchy output using our method is more similar to the WordNet



(a) WordNet hierarchy. (b) Hierarchy generated by our method (c) Hierarchy generated by DELUGE

Figure 4.6: Different hierarchies over the same set of words. For the hierarchy created by our method, we use the red dots to highlight the nodes with high uncertainty.

hierarchy than it is to the hierarchy created by DELUGE. This difference is largely due to DELUGE's use of co-occurrence of labels to define the hypernym-hyponym relation between words. However, the co-occurrence does not directly imply the connect hierarchy. In comparison, our method directly asks questions regarding the hierarchical structure about whether there is a path between a pair of nodes.

To summarize, our method and DELUGE significantly differ with respect to 3 aspects:

1. DELUGE relies on a knowledge base to get the seeding instances of concepts. For new concepts or rare concepts in knowledge bases, DELUGE would not apply. Our method does not depend on any prior knowledge and, therefore can be used readily for any concepts.

2. DELUGE builds hierarchies based on co-occurrence of labels. However, co-occurrence does not imply an "is a type of" relation directly. The results heavily depend on the seeding

instances from the knowledge bases. For example, the most famous islands in existing knowledge bases are countries. Therefore, DELUGE will produce a direct edge from "county" to "island" in the hierarchy, which does not make common sense. On the other hand, our method directly asks questions regarding the structure of the hierarchy. Our active query policy also targets at building hierarchies quickly.

3. DELUGE produces one single hierarchy for a set of words and, therefore ignores the uncertainty in building semantics hierarchies. Our method considers the uncertainty and captures even more common sense knowledge.

4.7 Conclusion

This chapter introduces an approach for learning hierarchies over concepts using crowdsourcing. Our approach fuses simple questions that can be answered by non-experts without global knowledge of the concept domain to build a globally coherent hierarchy. To deal with the inherent noise in crowdsourced information and with people's possible disagreements over hierarchical relationships, we develop a Bayesian framework for estimating posterior distributions over hierarchies. When new answers become available, these distributions are updated efficiently using a sampling-based approximation for the intractably large set of possible hierarchies. The Bayesian treatment also allows us to generate active queries that are most informative given the current uncertainty. New concepts can be added to the hierarchy at any time, automatically triggering queries that enable the correct placement of these concepts. Our approach can also detect synonyms with high accuracy. It is also worth noting that our approach lends itself naturally to manual correction of errors in an estimated hierarchy: by setting the weights to be inconsistent with a manual annotation to zero, the posterior over trees automatically adjusts to respect this constraint.

We investigated several aspects of our framework and demonstrated that it can recover quite robust hierarchies for real-world concepts using AMT. Importantly, by reasoning about uncertainty over hierarchies, our approach can unveil confusion of non-experts over concepts, such as whether a tomato is a fruit or a vegetable. We believe that these abilities are extremely useful for applications

where hierarchies should reflect the knowledge or expectations of regular users rather than domain experts. Example applications could be search engines for products or restaurants, or robots that interact with people who use various terms to relate to similar objects in the world. Investigating one such use case will be discussed in detail in the next chapter.

Chapter 5

EVALUATING TASK-DEPENDENT TAXONOMIES FOR NAVIGATION

5.1 Introduction

Taxonomies are useful across many real-world applications and scientific domains. Online shopping portals such as Amazon use catalogs to organize their products in order to simplify the task of navigation and product search for their users. In scientific domains, many machine learning algorithms and intelligent agents, including robots, also use taxonomies to improve their performance on fundamental tasks such as object recognition [221, 140, 209, 152], natural language understanding [202, 164, 13, 111], and others. Task-dependent taxonomies, *i.e.*, taxonomies adapted to specific application/task or to a specific cohort of users, are crucial for these applications and tasks to perform efficiently [55]. Task-dependent taxonomies are usually created by hiring domain experts, an expensive process that is infeasible for every task. Alternatively, there are methods for creating taxonomies autonomously [11, 12]. However, these methods lack the semantics/common sense of humans and often produce taxonomies that are inefficient in terms of their end-use. Hence, many applications [121, 50, 221] are limited to using generic taxonomies such as WordNet [69].

Crowdsourcing based techniques. Recently, several crowdsourcing-based techniques [37, 23, 185] (with well-designed algorithms and prepared questions) have demonstrated that non-experts also have the potential to build task-dependent taxonomies. Chilton et al. and Bragg et al. introduce techniques for creating taxonomies based on the co-occurrence of multi-label object annotation. Sun et al. take a Bayesian approach and propose an active learning algorithm for building taxonomies. Their approach can capture uncertainty by producing a distribution over these taxonomies instead of producing one single taxonomy as output. Most of the work [23, 185] evaluates the quality of output taxonomies using accuracy with respect to some gold-standard knowledge base such as WordNet [69]. However, no quantitative evaluation of the end-to-end deployment of taxonomies

compares the effectiveness of different taxonomies for tasks for which they are eventually used.

Research questions. The primary goal of our work is to understand whether crowdsourcing provides an effective solution to create taxonomies of knowledge. To this end, we seek to answer the following 3 research questions: 1) do task-dependent taxonomies built by crowdsourcing techniques improve end-to-end performance compared to using generic taxonomies built by experts?, and 2) does the uncertainty captured by probabilistic approaches further improve the efficiency of performing the navigation tasks? Answering these questions will in turn shed light on the third, more general research question: 3) *Can crowdsourcing-based techniques provide practical solutions to complex structural learning tasks such as building taxonomies?*

Our approach. In this chapter, we present the first quantitative study to measure the effectiveness of different taxonomies in context of an end-to-end application. Specifically, we compare the task-dependent taxonomies built by crowdsourcing techniques to a generic taxonomy built by experts. To start, we use a crowdsourcing based technique [185] to create a taxonomy using the AMT. Then, to perform the user study for quantitative evaluation, we design randomized behavioral experiments on AMT for a navigation task resembling many real-world applications, such as conducting a product search in online shopping portals. Participants in the user study differ from AMT workers who helped build the taxonomy we are evaluating. As part of our user study, we record various quantitative metrics such as the time of navigation, the number of clicks performed, and the search path taken by a participant to locate the target.

Our results. Our findings from the user study show that: 1) task-dependent taxonomies built by crowdsourcing techniques can reduce the navigation time by up to 20%, and 2) the uncertainty/distribution over the taxonomies can be exploited to help users perform navigation tasks more efficiently. These results affirmatively answer our research questions and confirm that crowdsourcing provides an effective solution to building task-dependent taxonomies. This, in turn, demonstrates that crowdsourcing can provide a practical solution for learning complex structure using responses to simple questions from non-experts. Furthermore, our design of behavioral experiments serves as an example for other researchers who want to use crowdsourcing to conduct user studies for evaluating the performance of complex systems.

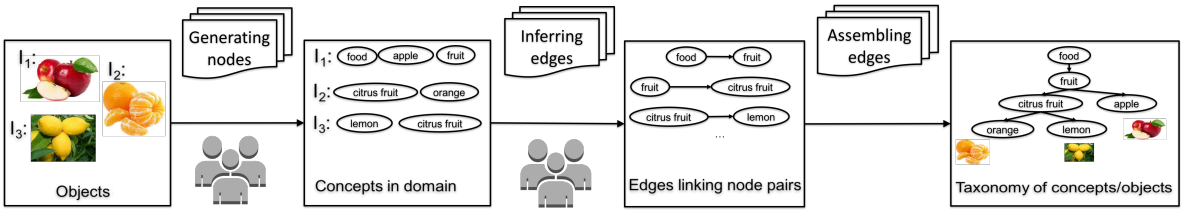


Figure 5.1: Illustration of the workflow of building task-dependent taxonomies via crowdsourcing. Collecting tags and inferring tags are done via crowdsourcing. Assembling edges to get the final taxonomy is performed by the algorithm.

The rest of this chapter is organized as follows: We first introduce the background of building taxonomies via crowdsourcing in Section 5.2. Next, we will present the data collection step and the experimental setup (Sections 5.3, 5.4). Then, we will discuss the results from the user study 5.6. Finally, we will discuss related work before conclude the chapter.

5.2 Building Taxonomies via Crowdsourcing

We begin by reviewing background information on crowdsourcing-based techniques to build taxonomies. The next section will use one of these techniques to build the taxonomy on our dataset for evaluation via a user study.

5.2.1 Different Components to Specify a Taxonomy

A task/domain is specified by a set of objects. A taxonomy represents a knowledge base by organizing the objects into a hierarchical tree structure. Two main components are necessary to specify a taxonomy completely: the set of nodes in the tree, and the set of edges connecting these nodes. Figure 5.1 illustrates the workflow involved in building a taxonomy on a toy example, as discussed further below.

Generating nodes. A natural approach of node generation is to represent the objects related to the task/domain for which the taxonomy is being built by a set of concepts or tags. These tags correspond to the nodes in the taxonomy tree. Each node is associated with a set of objects that is

tagged by the node or any of its descendant nodes. Most crowdsourcing-based methods [37, 185] begin by collecting a set of tags for these objects in hand (the first step in Figure 5.1). For example, CASCADE [37] shows a set of objects via their text descriptions to the crowdsourcing workers and asks them to provide tags they will use to refer to these objects. Sun et al. take a similar approach. However, instead of showing the text descriptions, the authors show visual images of the objects and ask the workers to annotate them with tags.

Inferring edges. The next step is to specify the directed edges that connect these nodes (the second step in Figure 5.1). In the hierarchical tree of the taxonomy, each node is more general than any of its descendant nodes. Inferring edges in the tree is a challenging problem as it requires having global knowledge of the complete set of nodes. In the crowdsourcing setting, non-expert workers would not possess such a global knowledge and are better at answering questions that can be answered with local/partial knowledge. So the key to designing crowdsourcing based algorithms for building taxonomies is to break down the problem requiring global knowledge into simpler problems/questions requiring only local knowledge to answer. The algorithms then fuse answers to these simple questions to generate the final global structure of the taxonomies (the last step in Figure 5.1).

5.2.2 Decomposing the Structure Learning Problem

We now categorize existing crowdsourcing techniques building taxonomies into two main categories based on how they decompose the global structure learning problem into simpler problems and questions.

Indirect questions to infer edges. CASCADE[37] and DELUGE[23] are representative techniques of this category, which use local questions without direct connection to the structure of the taxonomies. Chilton et al. introduce CASCADE, a workflow for creating taxonomies based on the co-occurrence of multi-label annotation of items. CASCADE takes a set of objects and the corresponding set of tags to generate simple questions/microtasks as follows: each microtask consists of one object and one tag. CASCADE then asks a worker to vote on whether the object fits the tag. In the fusion step, it creates nested categories representing the parent-child relationship of

these tags. DELUGE improves the efficiency of this annotation process to reduce the number of questions required to learn a taxonomy with the same accuracy.

Direct questions to infer edges. Sun et al. take a Bayesian approach and propose an active learning algorithm for building hierarchies/taxonomies. Their approach directly asks questions related to the structure of the taxonomy. Specifically, they ask *path questions*, *e.g.*, "Should there be a path from 'apple' to 'food' in the target taxonomy?", stated informally as "Is 'apple' a type of 'food'?". These questions can be answered by the crowdsourcing workers using only local/partial knowledge. Sun et al. provide a probabilistic model to efficiently maintain the distribution over all possible tree structures to represent the fused knowledge from the answers obtained from workers. At each iteration, the model updates the distribution based on the answer to a path question. Intuitively, if a worker gives a positive answer to a path, his/her method will increase the probability of the taxonomy trees with that path; otherwise, the probabilities of the corresponding trees will be reduced.

5.2.3 Technique Used for Evaluation

In this chapter, we use Sun et al.'s method for building task-dependent taxonomies and then compare them to the generic taxonomy of WordNet. The key reasons for using this technique are two-fold. First of all, the experimental study performed by Sun et al. shows that this Bayesian approach is empirically more cost-efficient compared to CASCADEand DELUGE, for which the cost quantifies the number of questions asked to the participants in order to produce a taxonomy of desired accuracy. Second, this approach outputs a distribution over taxonomies that in turn enables various operations: 1) using MAP (maximum a posteriori probability) inference to compute the most representative taxonomy, and 2) estimating the marginal likelihood of edges efficiently. These marginal likelihoods in turn capture the uncertainty that naturally exists in semantic taxonomies, *e.g.,* the answer to the question "Is 'candy' a type of 'dessert'?" or "Is 'candy' a type of 'sugar'?" is inconsistent depending on the participants' beliefs. Our second research question concerns the use of these uncertainties, and whether it can improve the efficiency of performing navigation tasks relative to the use of a single representative taxonomy.

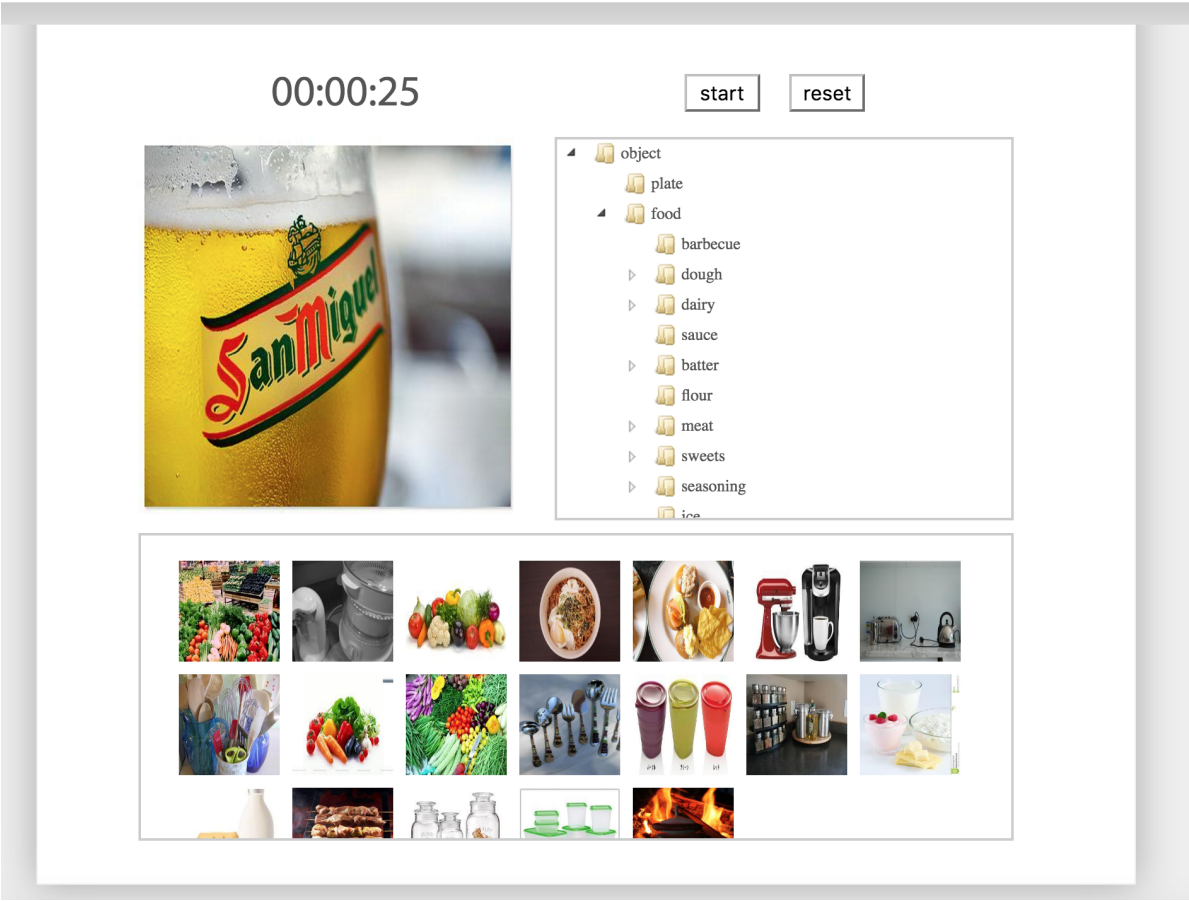


Figure 5.2: This figure shows the interface provided to the AMT workers to perform the navigation tasks: (i) the target object is shown in the *top-left* panel; (ii) the *top-right* panel shows the taxonomy – a navigation system; and (iii) the *bottom* panel shows a subset of the images associated with the current node clicked by the user.

5.3 Application Domain and Navigation Tasks

In this section, we discuss the application domain, the process of data collection for this domain, and the navigation tasks we use for evaluation.

5.3.1 Application Domain

In this chapter, we consider the application domain of *kitchen*. This domain involves objects used in the kitchen in everyday life such as food items, kitchen appliances, cooking utensils, *etc*. We chose this domain for 2 reasons: 1) the navigation tasks in this domain represent simple everyday scenarios, ensuring that AMT workers are familiar with the domain to perform the navigation tasks for evaluation, and 2) it is a rich domain that captures the intricacies of real-world applications, such as online shopping portals.

Collecting objects for our domain. We begin by collecting the most frequently used keywords in the kitchen domain. We extract the words/phrases representing the most commonly used kitchen concepts from the WikiHow [206] articles using the Stanford parser [110]. We then do a two-step filtering process as follows: first, we keep the top 100 frequently occurring keywords as candidates; then, we filter out the candidates that are not in WordNet[1]. This filtering process yields a set of about 70 *seeding* keywords — these are the concepts that represent the kitchen domain. We then use these keywords to search for images via Google's image search engine to get exemplary images. For each keyword, we collected about 50 images, giving us a total of 3500 images. These images represent the objects for the kitchen domain, *c.f.*, "Objects" in Figure 5.1. In the next section, we describe the different taxonomies that we build to organize these objects.

5.3.2 Navigation Task

Figure 5.2 illustrates a navigation task for the kitchen domain using one of the taxonomies we evaluate. This is the same interface provided to AMT workers during our user studies. The goal of a navigation task is to localize a target object from the set of objects in the domain — the target object is shown in the top-left panel of Figure 5.2 and randomly picked from all candidate objects. All candidate objects are organized using a taxonomy represented by nodes and edges as discussed in section 5.2. The user has access to this taxonomy as the *navigation system*, shown in the top-right

[1]This second filtering step is to have a reasonable WordNet taxonomy that can be used as a baseline taxonomy for the comparison.

panel in Figure 5.2. Recall that each node in the taxonomy is associated with a set of objects that is tagged by this node or any of its descendant nodes. When a user clicks on a node, a random subset of the images belonging to that node (or any of its children) appears (the bottom panel of Figure 5.2). The user can search through these images to find the target. Or the user can use these images as an indicator of the kind of objects expected to appear under the current node, then click finer-grained nodes beneath the current node to look for the target. During the navigation task, the user can also backtrack the search path by clicking on a non-descendant node. The task finishes when the user finds the target object in the bottom panel and clicks on it. This navigation task resembles many real-world applications. For example, many online shopping portals, such as Amazon [4] and BestBuy [9], organize their products in a taxonomy to help their customers navigate and search efficiently.

Metrics. Our primary goal is to quantitatively measure the efficiency of the taxonomies as a navigation system. The natural metric is the time a user spends per task, *i.e.*, time to localize the target object using a given taxonomy. A shorter time suggests that the system is more efficient at guiding users to find the target object. Another metric of interest is the number of clicks performed before a user localizes the target object. Intuitively, this number captures whether a taxonomy matches the common sense. The number of backtracking steps done by a user while navigating the taxonomy tree is another metric we measure. A large number of backtracking steps indicates that the taxonomy might confuse its users.

5.4 Taxonomies Used for Evaluation

In this section, we present the specific taxonomies that are used for evaluation in our work. We compare three different taxonomies: 1) the WordNet taxonomy [69], 2) MAP (the most representative) taxonomy created by [185], and 3) the multifaceted taxonomy created by [185] to capture uncertainty.

5.4.1 WordNet taxonomy

WordNet, a large taxonomy containing over 150,000 words, groups words that are roughly synonymous into *synsets*. These synsets are then connected to each other based on semantic relations, represented as a Directed Acyclic Graph (DAG). To generate a WordNet-based taxonomy for our domain, we extract a taxonomy from WordNet covering the 70 seeding keywords representing the concepts for kitchen domain as described below.

First, we locate the nodes in WordNet's DAG corresponding to the 70 keywords. Although all 70 keywords are in WordNet (note that we filtered the keywords to keep only the ones that are in WordNet), it is still a non-trivial task to locate the nodes in the DAG that are relevant to the kitchen domain because the same word may have different meanings *i.e.*, it could correpond to different synsets/nodes in WordNet. Therefore, whenever a keyword is mapped to multiple nodes in WordNet, we use the following heuristic to decide which synset/node in the WordNet has the semantics related to the kitchen domain. The heuristic will pick a synset if it has 'food', 'instrumentality' or 'appliance' as its ancestor synsets; otherwise, it picks the synset based on priority ordering defined by the match of the target keyword in the *lemmas* (semantic sense) of the sysets. It is worth noting that the crowdsourcing-based technique we use [185] faces a similar problem of polysemy in collecting tags in the first step, *c.f.*, Figure 5.1. For example, 'apple' could represent the fruit or the corporation. However, crowdsourcing based techniques resolve this naturally as the tag collection process involves showing images of the objects to the workers.

Then, after locating the nodes in the DAG for the 70 seeding keywords, we greedily find the minimum spanning tree in this DAG to cover these 70 nodes. To find this tree, the algorithm initializes a forest with 70 nodes. At each iteration, it picks a pair of nodes, adds their lowest common ancestor into the forest, and connects these two nodes with the new ancestor node. The algorithm then repeats this process until the forest becomes a single-rooted tree. After executing this algorithm, we have a total of 96 nodes. Figure 5.5 (a) shows this WordNet taxonomy corresponding to the 96 nodes; the 70 nodes corresponding to the seeding keywords of kitchen domain are marked in *blue*.

5.4.2 *Crowdsourcing-Based Taxonomies*

Next, we use the method of Sun et al. for building crowdsourcing-based taxonomies. Following the workflow in Figure 5.1, we first collect tags for the object images corresponding to the 70 seeding keywords representing the kitchen domain.[2] We show the 70 representative object images to the AMT workers and ask them to provide 3 distinct tags for each image they see. Furthermore, each image is shown to 3 distinct workers. At the end, we aggregate all the tags, and we keep only the tags that are used by more than 5 workers to refer to these objects. We combine the newly collected tags with the original 70 keywords to get the final set of 104 tags/nodes. We then run the sequential version of the method proposed in [185] to learn the taxonomy over the set of these 104 nodes. As discussed in the previous section, the output of the method is a distribution over all taxonomy trees with 104 nodes.

MAP taxonomy. This is the taxonomy obtained by doing MAP (maximum a posteriori probability) inference and is the most representative taxonomy [185]. Figure 5.5 (b) shows this MAP taxonomy corresponding to the 104 nodes; the 70 nodes corresponding to the seeding keywords of kitchen domain are marked in *blue*.

Multifaceted taxonomy. The MAP taxonomy ignores the uncertainty that naturally exists in semantic taxonomies. One reason for this uncertainty is the multifaceted classification [159] problem as there are different criteria that can be considered by humans to classify and categorize objects. The most common example is the online shopping portals, where products can be categorized by price, type, maker and so on. In order to incorporate this uncertainty, we create an augmented taxonomy to utilize the uncertainty to create more options for users to localize the target object in the taxonomy [171]. We begin by computing the marginal likelihood of the edges from the distribution of the taxonomies obtained above — their Bayesian approach provides a computationally tractable method to compute these likelihoods. We then keep only the edges that have a marginal likelihood of more than 0.1. Next, we append the MAP taxonomy tree with these additional edges. As an

[2]Note that we have 50 images per seeding keyword. However, in order to make this tag collection process cost-efficient, we randomly selected 5 images per keyword, then grouped these 5 images into one big image, and used this as the representative object image to collect tags.

example, *c.f.*, Figure 5.5, we obtain the edges from 'sugar' to 'candy' and 'dessert' to 'candy' among these high likelihood edges. The MAP taxonomy tree (Figure 5.5 (b)) already has an edge from 'dessert' to 'candy'. Hence, we append this tree with edge from 'sugar' to 'candy'. It is important to note that the resulting structure after augmenting these edges is not a tree but a DAG. For the user studies, we consider showing it to users as a tree for easier interpretation and navigation. To convert this DAG to a tree, we do the following: For each node x with $n > 1$ parent nodes represented by the set π_x, we replicate the sub-tree rooted at x for n times and add this sub-tree as a child node to each of the parent nodes in π_x. This treatment to the DAG gives us a taxonomy with replicated nodes (as shown in Figure 5.5 (c)) and we call this taxonomy a Multifaceted taxonomy. In Figure 5.5 (c), the set of nodes that was replicated and added to the MAP taxonomy at different positions are highlighted in *red*.

5.5 Structural Comparison of Taxonomies

In this section, we compare the three taxonomies based on the various structural and qualitative aspects a discussed below.

Number of nodes. The MAP taxonomy (*c.f.*, Figure 5.5 (b)) contains 104 nodes compared to 96 nodes in the WordNet taxonomy (Figure 5.5 (a)). Note that both these taxonomies contain the same set of 70 seeding keywords we begin with. A careful examination reveals that the MAP taxonomy includes many *instance names* that specify some seeding keywords. For example, 'fish' is a seeding keyword, and 'salmon' is a particular type of 'fish'. The reason for seeing these instance names is that we show images corresponding to the 'fish' object to collect the tags, and upon seeing 'salmon' food dish in the image, some workers may annotate the image with the tag 'salmon'. These specific instance names do not exist in the WordNet taxonomy since we generate the taxonomy by adding only ancestor nodes of the seeding keywords in the DAG of the WordNet. Furthermore, if we remove all nodes related to these specific instance names from the MAP taxonomy, we are left with 91 nodes — a number slightly lower than the size of the WordNet taxonomy, *i.e.*, 96 nodes. This suggests that non-experts might use a more compact set of tags/concepts to refer to the same collection of objects for a certain application domain.

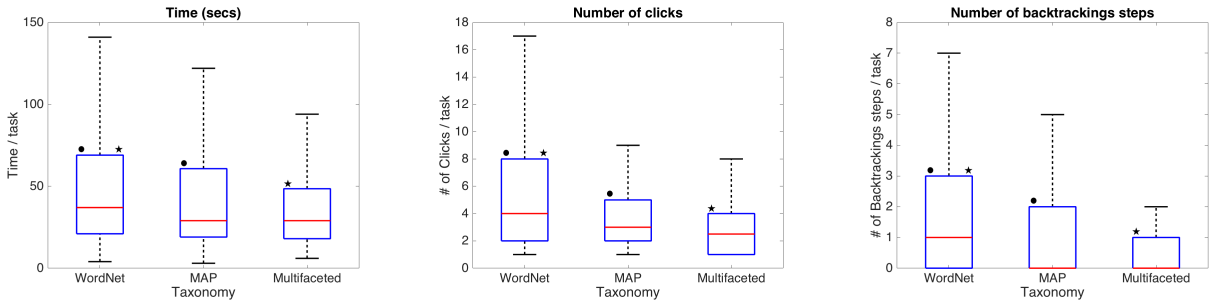
Depth/width of taxonomy. The MAP and the Multifaceted taxonomies both have a depth of 6 and a width of 16. On the other hand, the WordNet taxonomy has a depth of 8 and a width of 8. These numbers suggest that non-experts tend to build shallower but wider trees, while experts might prefer creating deeper but narrower trees.

Words in taxonomy. Taxonomies built via crowdsourcing have more familiar keywords/concepts associated with the nodes. The WordNet taxonomy has more obscure/infrequently occurring keywords such as 'nutriment', 'instrumentality', 'concoction', and others. These keywords in the WordNet taxonomy might be unfamiliar to ordinary users, in particular non-native English speakers.

Uncertainty. The taxonomy in Figure 5.5 (c) captures some interesting information about the uncertainty of the edges. One example is the 'blender' node. As per the MAP taxonomy in Figure 5.5 (b), 'blender' is a type of 'kitchen appliance'. However, in the Multifaceted taxonomy, a 'blender' could also be a kind of 'container'. This uncertainty captures the notion of multifaceted classification: humans use different attributes to classify the same objects/ concepts. Some users classify 'blender' according to its essential functionality as an appliance, while others believe that'blender' can be used as a 'container'. Keeping two 'blender' nodes in one taxonomy thus potentially provides users with more choices in navigating the taxonomy to find the target objects. A user who believes 'blender' is a 'container' will not find the target object directly in the MAP taxonomy, leading more clicks and backtracking steps. But the user can quickly locate 'blender' in the Multifaceted taxonomy. The taxonomy in Figure 5.5 (c) also illustrates other sorts of uncertain edges. For example, some users are confused about the relation between 'dough' (flour mixed with less water) and 'batter' (flour mixed with more water). In the Multifaceted taxonomy, we can see the edges from 'dough' to 'batter' and from 'batter' to 'dough'. These kind of cycles suggest that these two nodes are synonyms.

5.6 User Study for Quantitative Evaluation

We performed our user study on AMT by recruiting workers to perform navigation tasks. Each worker participating in the study is allowed to accomplish the navigation task only two times: one task uses the WordNet taxonomy, and the other task uses either the MAP taxonomy or the



(a) Time (in second) per task by taxonomy types. $p^{\bullet} < 0.001$. $p^{\star} < 0.0001$.

(b) Number of clicks per task by taxonomy types. $p^{\bullet} < 0.001$. $p^{\star} < 0.01$.

(c) Number of backtrackings per task by taxonomy types. $p^{\bullet} < 0.0001$. $p^{\star} = 0.0001$.

Figure 5.3: Boxplots showing the evaluation metrics by type of taxonomy. "$\bullet$" compares WordNet with the MAP taxonomy; "$\star$" compares WordNet with the Multifaceted taxonomy.

Multifaceted taxonomy. The order of these two tasks is chosen randomly. We do not allow workers to perform tasks with both the MAP and the Multifaceted taxonomies as they might learn one taxonomy from another, thereby affecting the validity of the study. We ran these online experiments on AMT for 3 days restricted to US workers with more than 95% approval rate, and over 800 distinct workers participated in the study. Considering only the tasks successfully accomplished by the workers, we have the following number of completed tasks: 730 using the WordNet taxonomy, 343 using the MAP taxonomy, and 436 using the Multifaceted taxonomy.

5.6.1 Task-Dependent vs. Generic Taxonomies

We first compare the efficiency of using task-dependent taxonomies to the generic taxonomy to answer the first research question (*viz.*, *Do task-dependent taxonomies built by crowdsourcing techniques improve the end-to-end efficiency compared to generic taxonomies built by experts?*) Among the three taxonomies we use, both the MAP and the Multifaceted are task-dependent taxonomies, and the WordNet taxonomy is generic.

Table 5.1 shows the summary statistics and the average performance of tasks for three metrics,

Table 5.1: The means $\pm$ standard deviations (STD) of time, the number of clicks, and the backtracking steps for finding one target.

Metrics	Taxonomies		
	WordNet	MAP	MF
`time(s)`	60.2 ± 76.3	49.8 ± 57.2	47.9 ± 63.2
`# of clicks`	6.0 ± 6.7	4.4 ± 5.2	4.0 ± 5.7
`# of bts`	2.3 ± 3.3	1.3 ± 2.7	1.1 ± 3.0

including: time, number of clicks, and number of backtrackings per task, where backtracking indicates that a user navigated back up through the hierarchy after discovering a potential dead end. Figure 5.3 shows the distribution of the different metrics for the different types of taxonomies. Significance testing involves unpaired t-tests.

Results show that using the WordNet taxonomy performs significantly worse than using both the MAP and the Multifaceted taxonomies regarding all three metrics ($p < 0.05$). Among the three taxonomies, the Multifaceted taxonomy performs the best. It outperforms the WordNet taxonomy by 20% with respect to time, by 33% with respect to the number of clicks, and by 49% with respect to the number of backtrackings.

These above findings show that non-expert workers at AMT can build task-dependent taxonomies. The task-dependent taxonomies are designed for the kitchen application domain and can therefore be deployed by navigation tasks to achieve efficient navigation performance. On the other hand, the generic taxonomy does not consider any task information. Although it is created by experts and is correct in its content, it does not help to achieve efficient navigation performance in the desired target domain.

5.6.2 MAP vs. Multifaceted Taxonomies

We examined the tasks using MAP or Multifaceted taxonomies and tried to answer the second question (*viz.*, *Does the uncertainty captured by probabilistic approaches help?*). To determine

Table 5.2: The means $\pm$ standard deviations (STD) of time, the number of clicks and the backtracking steps for finding a target: the targets are from the nodes with uncertainty.

Metrics	Taxonomies	
	MAP	MF
`time(s)`	49.1 ± 44.9	41.3 ± 44.6
`# of clicks`	4.3 ± 4.9	2.8 ± 2.5
`# of bts`	1.6 ± 2.5	0.8 ± 1.4

the effect of using uncertainty information, we segmented all the 70 nodes into two groups based on whether there are replicated nodes in the Multifaceted taxonomy. We did this because: 1) the MAP and Multifaceted taxonomies are different only on these nodes, and 2) these nodes have high uncertainty, letting us evaluate the difference between using and not using uncertainty information in the taxonomies.

First, we compare the performance of the taxonomies on the group of nodes with uncertainty. The evaluation metrics of the two taxonomies are summarized in Table 5.2. The average time, the number of clicks, and the number of backtrackings using the Multifaceted taxonomy are consistently lower than using the MAP taxonomy. Among the three metrics, both the numbers of clicks and backtracking results are significantly different between the two taxonomies, with both p values < 0.01. The difference on the time metric is as significant as the other two metrics given the p value of 0.218. We hypothesize that the time does not exactly reflect the time taken by users to navigate through the taxonomy because it also includes time spent for searching for target images.

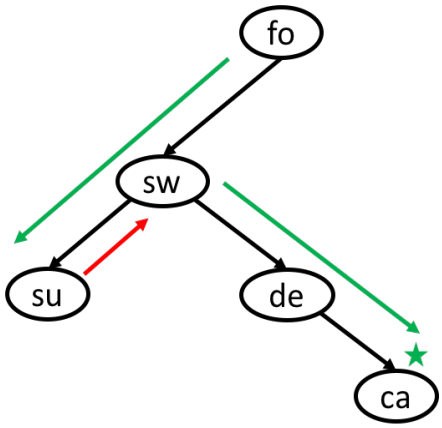
Next, we report the backtracking rates of each node with hierarchy uncertainty in Table 5.3. The backtracking rates tell us how often a user has to backtrack at least once to localize the target object. We rank the nodes by their entropy, given in the last column. To compute the entropy, we use the approach in [185] to determine the marginal likelihoods of all possible edges (the probabilities of all the edges ending at the same node are automatically normalized). The distribution of a node tells us which nodes are likely to be the parent of that node. We then compute the entropy of the

Table 5.3: The backtracking rates (%) of nodes with uncertainty when different taxonomies are used. "w/un" means the group of nodes with uncertainty, and "wo/un" indicates the group of nodes without uncertainty.

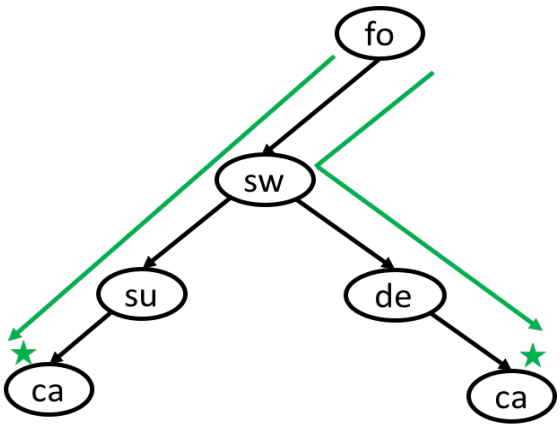
Node	BT-MAP(%)	BT-MF(%)	Entropy
sauce	44	17	0.70
flour	100	8	0.69
salad	0	18	0.69
pie	33	20	0.69
grill	67	100	0.69
stove	60	88	0.69
blender	75	89	0.69
microwave	100	90	0.69
pot	100	14	0.69
fork	0	0	0.69
honey	43	29	0.69
juice	20	33	0.69
candy	67	17	0.69
chicken	50	25	0.69
chocolate	50	0	0.69
batter	80	60	0.64
dough	33	20	0.64
w/un	54 ± 31	37 ± 34	0.69 ± 0.02
wo/un	37 ± 34	36 ± 27	0.00 ± 0.00

distribution to represent the uncertainty of the node.

The individual results show that using the MultiFaceted hierarchy MF clearly outperforms the MAP hierarchy on searches relating to these nodes, as confirmed by a reduction of the average backtracking rate from 54% to 37% ($\tilde{3}0\%$ marginal improvement). Although the paired t-test is not significant (p = 0.06), using the Multifaceted taxonomy outperforms the MAP taxonomy for



(a) The sub-tree for *candy* from the MAP taxonomy.



(b) The sub-tree for *candy* from the Multifaceted taxonomy.

Figure 5.4: Illustration of the search paths taken by users to search for *candy* using different taxonomies. We use the following short notations: *fo* for *food*, *sw* for *sweet*, *su* for *sugar*, *de* for *dessert*, and *ca* for *candy*. The green edges indicate the path-taken by the users; the red edge indicates backtracking; the stars indicate the targets.

most nodes. This is not surprising since MF provides alternative paths for uncertain nodes, thereby avoiding the backtracking required by the MAP taxonomy. The last row in Table 5.3 also shows that on nodes with high entropy, users backtrack more often than on nodes without uncertainty. This suggests that the uncertainty over nodes in the taxonomy does capture the uncertainty users have when they perform navigation tasks.

We examined the common patterns of user searching for target nodes with uncertainty using the MAP and Multifaceted taxonomies. Figure 5.4 shows how users look for *candy* using different taxonomies. We show the parts of the taxonomies relevant to the target node *candy*, and highlight the path taken by the users to look for *candy*. When doing so, about half of the users first go through *sugar* from *sweet* because they might believe that *candy* is a type of *sugar*. The other users take the path from *sweet* to *dessert*. The MAP taxonomy does not consider the uncertainty information and puts *candy* only as a child of *dessert*. Users going in the direction of *sugar* could

not find the target in the MAP taxonomy and therefore have to backtrack and search the other alternative path, from *dessert* to *candy*. On the other hand, the Multifaceted taxonomy realizes that there is uncertainty over the location of *candy*. Therefore, it puts candy under both possible locations. Hence, the users could find *candy* by taking either path in the Multifaceted taxonomy. Using uncertainty reduces the backtracking rate for sugar from 67% to 17%.

5.7 Related Work

Crowdsourcing for user studies. Crowdsourcing platforms provide easy, on-demand and low-cost access to a large pool of diverse participants, thereby opening up opportunities to conduct large-scale online user studies [109, 142]. In the computer science community, crowdsourcing-based user studies have been widely deployed to evaluate the performance of algorithms in various application domains, such as graphical perception design [87, 146], web search [3, 32], recommendation systems [141], computer vision [56], *etc.*

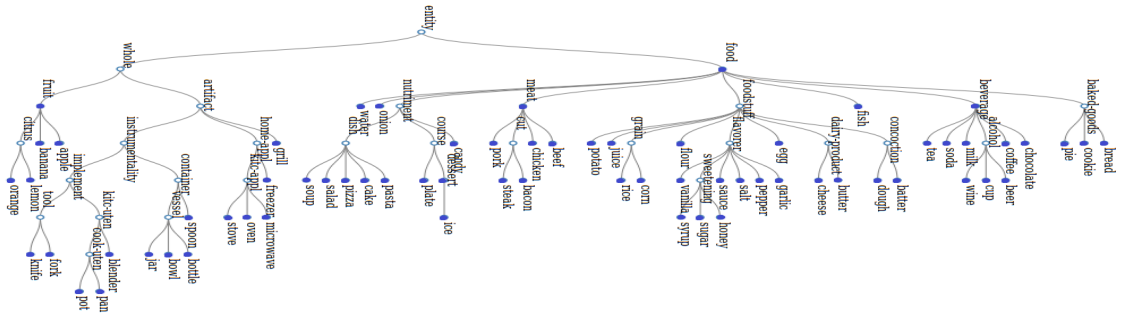
Evaluating ontologies. Taxonomy is usually a major component of an ontology; hence, the methodologies for evaluating ontologies significantly overlap with the evaluation of taxonomies. When a ground-truth ontology or knowledge base is available, precision-recall-based metrics inspired from information retrieval evaluations have been used [138]. However, these methods are inadequate otherwise. On the other hand, data-driven methods [158] evaluate target ontologies to match the knowledge included in some existing corpus or test data set. However, they do not quantify the quality/utility of a target ontology in terms their efficiency of performing the desired task. One orthogonal direction for evaluating taxonomies is to study the *cognitive feasibility* of the taxonomy by measuring participants' reactions to the relations entailed by the structure of the taxonomy [64]. Building on the ideas of Evermann and Fang, [149] proposed a crowdsourcing-based user study to evaluate/verify target ontologies. However, [149, 64] rely on microtask methods of ontology verification wherein participants answer simple binary questions, such as "Is every 'heart' an 'organ'?".

Task-driven evaluation of taxonomies. Task-driven evaluations [25] directly measure the end-to-end performance of different taxonomies. These methods [25] define simple microtasks, *e.g.*,

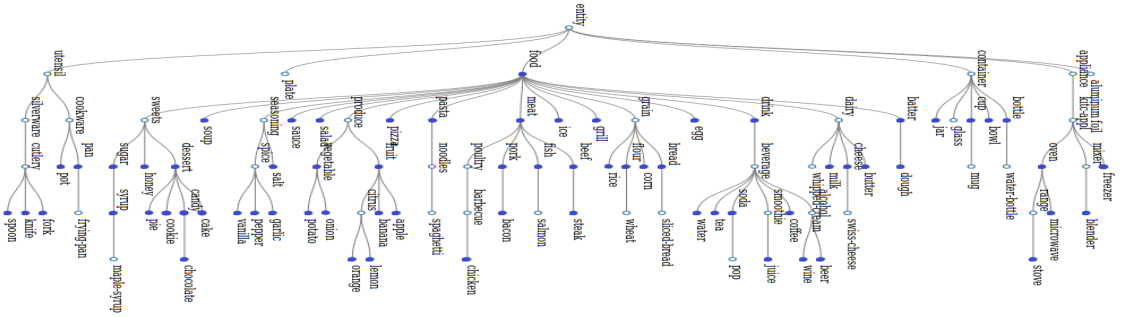
binary questions to evaluate the accuracy and coverage of the taxonomy content. The downside of these methods is that only local knowledge of the target taxonomy usually suffices to answer these questions. Our approach is a type of task-driven evaluation method. However, unlike the traditional task-driven methods that focus on assessing the content via simple questions, our approach focuses on evaluating the efficiency of using a taxonomy for doing *inference*: We rely on a more sophisticated navigation task to assess the taxonomy as a whole, representing a knowledge base in a hierarchical structure.

5.8 Conclusion

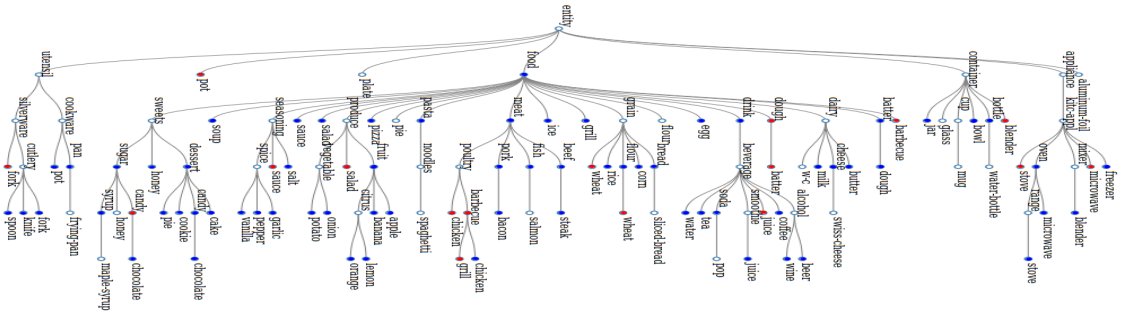
In this chapter, we evaluated crowdsourcing-based techniques for building task-dependent taxonomies. Ours experiments show that task-dependent taxonomies built by crowdsourcing techniques significantly improve the efficiency of navigating and searching for target objects compared to using a generic taxonomy created by experts. The results also show that using multifaceted taxonomies to capture uncertainties over node positions reduces the number of clicks/backtracking steps performed by users to find objects. These results demonstrate that crowdsourcing-based techniques can be deployed for complex structural learning tasks such as building semantic taxonomies.



(a) WordNet taxonomy for kitchen domain.



(b) MAP taxonomy built by crowdsourcing techniques [185] for kitchen domain.



(c) Multifaceted taxonomy capturing uncertainity built by crowdsourcing techniques [185] for kitchen domain.

Figure 5.5: Three taxonomies for kitchen domain used for evaluation. In (a) the WordNet and (b) the MAP taxonomy, the 70 nodes corresponding to the seeding keywords of kitchen domain are marked in *Blue*. In (c) the Multifaceted taxonomy, a small set of nodes with high uncertainty are replicated and added to the MAP taxonomy at different positions (highlighted in *Red*). The following abbreviations are used in these taxonomies: 'kitc-appl' for 'kitchen appliance', 'cook-uten' for 'cooking utensil', 'w-c' for 'whipped cream', 'home-appl' for 'home appliance', and 'kitc-uten' for 'kitchen utensil'.

Chapter 6

NEVER-ENDING OBJECT LEARNING

6.1 Introduction

Humans are capable of learning continuously: we can learn to recognize new objects and to associate new names with existing objects. To be useful over the long term, open-ended deployments, personal robots must be able to do the same. The predominate approach to object recognition is to train object classifiers on a predetermined set of objects with a predetermined set of names and then to test their performance on new instances of the same set of objects and names [50, 122, 115]. This train-once-then-test approach is inadequate for personal robots, since it is infeasible to assume that all objects, and all possible names that people refer to them by, are known in advance. So for future robots to be useful, they require the ability to learn not only new objects, but also new names for existing objects over time [183].

To see the potential problem of not learning new object names, we asked workers on Amazon Mechanical Turk (AMT) to name objects in a popular RGB-D dataset [122]. This dataset contains objects that are organized according to WordNet categories (*i.e.*, with a pre-specified set of names). We found that only 75% of the names given by the workers are included in WordNet. Thus, even if a robot correctly apply labels to 100% of the objects in their WordNet categories, it would correctly name only 75% of the objects to their AMT workers' simply because it lacks the ability to associate new names to existing ones. People do not use only category names to refer to objects [106]. For instance, a can of "Pepsi Cola" might be referred to as "Pepsi", "pop", "soda", "can", or "Pepsi can".

To deal with the challenge of learning novel objects and names, we introduce `NEOL`, a framework for Never-Ending Object Learning. `NEOL` learns objects and their names continuously – adding and improving classifiers after every object encounter – rather than learning via the train-once-then-test

approach.

There are two major challenges to developing a framework for never-ending object learning. First, how should objects and their corresponding names be organized? A naïve approach would be to train an independent classifier for every object name, using instances with that name as positive examples and instances of all other object names as negative examples. However, this approach ignores the important relationships between names. For instance, since apples are also fruits, every object named "apple" should also be a positive training example for the name "fruit". Many false negative labelings will occur if the naïve approach is deployed that could be especially harmful for a never-ending learning paradigm, where training data are quite limited. We decide to organize names and objects according to semantic hierarchies of concepts, which have been shown to be useful in building classifiers for object recognition and improving recognition accuracy [50, 55, 121]. Most prior work uses existing hierarchies, which may not be optimal for their tasks. To deal with this challenge, NEOL automatically learns a hierarchy over object names using crowdsourcing, inserting new names as they occur.

The second challenge of never-ending object learning is that the robot needs to cope with inconsistent, sparse, and noisy annotations. This phenomenon is related to the fact that each object can be referred to by different names, and it is difficult for a human teacher to label all possible names of an object. As a result, a robot knows only that an object can be referred to as "fruit" but is uncertain about the specific type of the fruit. Although there are methods [41, 27, 186] for explicitly considering the problem of partial labelling, they cannot yet achieve human-level annotation accuracy. Alternatively, crowdsourcing platforms provide affordable human labour to fill in the missing annotations efficiently [23]. However, as a robot sees more objects and names, relying only on human annotations becomes economically infeasible. Moreover, crowdsourcing workers often provide noisy answers. As will be shown in an experiment in Section 6.3, noisy answers could be harmful to a never-ending learning system. We will deploy the hierarchy maintained by the system to efficiently annotate objects and reduce labelling noise.

Our contributions. We introduce NEOL, a framework for robots to learn new objects and new object names over time. We extend our previous crowdsourcing-based method for building semantic

hierarchies of names using existing knowledge bases. We propose an efficient and robust way to fill in missing annotations of objects using the hierarchy and crowdsourcing. Moreover, we design a robust method that uses both in-domain RGB-D images and out-domain images to accurately recognize objects by names. Extensive evaluations show that NEOL can achieve robust results even given challenging settings.

This chapter is structured as follows: Section 6.2 presents a summary of related work. Section 6.3 introduces the never-ending object learning method, NEOL. Experimental results are provided in Section 6.4. Finally, we conclude in Section 6.5.

6.2 Related Work

Image annotation is often a multi-label problem [198, 204]. Many multi-label works [99, 217] assume that each training image in the training set has a complete label assignment. However, this assumption is typically not true [186]. When people teach a robot an object, they usually mention only a subset of positive labels for the object. A robot can use the object as a negative example of the labels that are not mentioned by the human. However, this naïve approach introduces false negative labels that could cause ambiguity when learning objects. One approach to tackle this problem is to fill in the missing labels automatically, as was down in WELL [186] and MLR-GL [27] in the multi-label literature. But these methods cannot to reach human-level annotation accuracy. Another promising direction is to include humans in the loop by having crowdsourcing workers give multi-label annotations to objects [23]. But, existing methods do not consider the structure information of labels, *i.e.*, hierarchies of the given labels. In this work, we combine hierarchy with crowdsourcing to efficiently annotate objects with missing labels.

6.3 Never-Ending Object Learning

In our setting, a robot is initialized without any knowledge of the objects in its working environment. Its goal is to recognize objects based on names and visual appearance. To be able to do so, it gradually learns both objects and their names from humans. A person will show the robot one object at a time and teach the robot several names that can be used to refer to the object. The robot

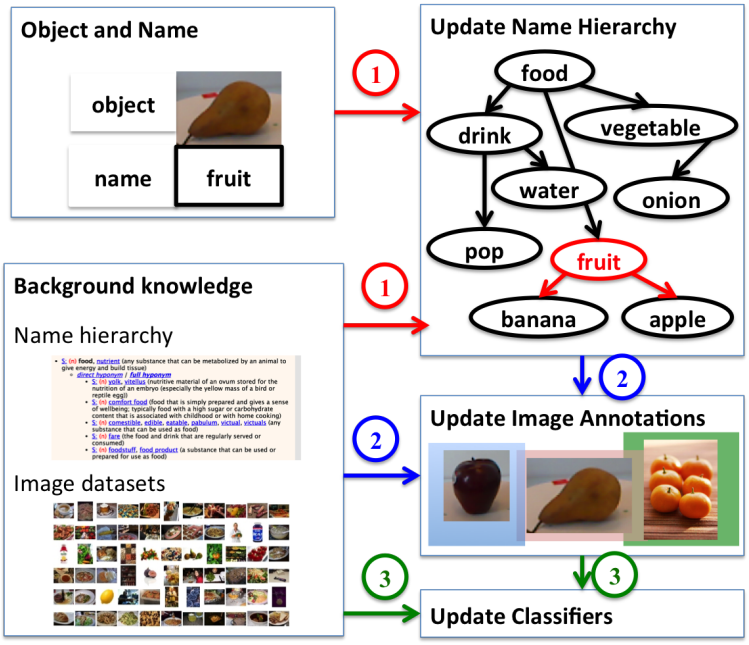


Figure 6.1: Pipeline of NEOL to learn one new object. NEOL takes the object and name pair as an input, then gives object classifiers, object annotations and a semantic hierarchy of names as output. ① indicates updating the hierarchy, ② corresponds to update object annotations, and ③ leverages background image datasets to re-train classifiers.

remembers the names, observes the object from several views, and then learns to recognize objects of similar kinds.

Figure 6.1 summarizes the major technical components of NEOL. We make three design choices in this pipeline:

How to organize objects and names. The robot is going to learn many names continuously. It could take a naïve approach by treating all names independently. However, this approach ignores the correlation between names, which could be useful to speed up the learning process [166]. As inspired by previous work [121, 50], we decide to organize names into a semantic hierarchy. All objects that can be referred to by a certain name will be associated with that name node in the hierarchy. As the robot learns new objects and names, it must update the hierarchy. NEOL deploys a method using crowdsourcing and a background name hierarchy to efficiently do so. Details will be

Algorithm 3 Pseudo-Code for NEOL

Input:

Y^S: mapping between source-domain images and names

Output:

A hierarchy $H^{(t)}$ after seeing t object-name pairs

Initialize:

$H^{(0)} := \emptyset$, $t := 0$

while the robot receives $(\iota^{(t+1)}, n^{(t+1)})$ **do**

  $t := t + 1$

  $(H^{(t)}, \tilde{\mathtt{n}})$:= `updateHierarchy`$(H^{(t-1)}, n^{(t)})$

  $H^{(t)}$:= `updateAnnotation`$(H^{(t)}, \iota^{(t)}, \tilde{\mathtt{n}})$

  for $(\iota, n^{(t)}) \in Y^S$ **do**

    $H^{(t)}$:= `updateAnnotation`$(H^{(t)}, \iota, \tilde{\mathtt{n}})$

  for $\tilde{\mathtt{n}}' \in H^{(t)}$ **do**

    $\tilde{\mathtt{n}}'.f$:= `trainClassifier`$(\tilde{\mathtt{n}}'.\mathcal{I}^+, \tilde{\mathtt{n}}'.\mathcal{I}^-)$

Return: $H^{(t)}$

discussed in Section 6.3.1.

How to annotate objects. Usually, one object can be referred to by several names (multi-label). However, a human teacher rarely enumerates all possible names related to the object. Therefore, the robot knows only a few names for a target object. Naively, it could associate the object with just those names mentioned by the teacher, but not the other names. NEOL deploys a method to fill in all the annotations of a target object efficiently using the name hierarchy and crowdsourcing. The details will be given in Section 6.3.2.

How to train object classifiers. In the never-ending learning setting, positive examples from the teacher could be very sparse. Building image classifiers with sparse training examples could cause overfitting, and not generalize well to recognize new instances of the same category. We propose to improve the generalization ability of image classifiers in NEOL using annotated images from external knowledge bases whenever possible. This will be discussed in Section 6.3.3.

We describe the meta algorithm of NEOL before going into details, using the following notations..

$\mathcal{N} = \{n_1, \ldots, n_L\}$ denotes a set of names, and $\mathcal{I} = \{\iota_1, \ldots, \iota_M\}$ a set of object images. Each ι_m is associated with at least one name in $\mathcal{N}$. All names in $\mathcal{N}$ are organized in a tree-structured hierarchy H. Each node $\tilde{\mathtt{n}}_l$ in the hierarchy is a quadruple $(n, \mathcal{I}^+, \mathcal{I}^-, f)$, where $\tilde{\mathtt{n}}_l.n = n_l$, $\mathcal{I}^+$ includes all object images that can be referred to by the name n, and vice versa for $\mathcal{I}^-$. f denotes the classifier that takes the image feature $\boldsymbol{x}_m$ of an object image ι_m and predicts $f(\boldsymbol{x}_m)$, indicating whether ι_m can be annotated with the name n.

Since we use external image datasets in NEOL, it is necessary to distinguish them from the images collected in the target domain. Following standard convention, we call the external databases the source domain and use superscript S to indicate any concepts (names or objects) from it. Concepts shown by the teacher are called the target domain. We use the superscript T to denote target domain concepts. If no superscript is used for a concept, it is in the target domain by default.

Using these notations, we formalize NEOL in Algorithm 3. NEOL is initialized with source domain images $\mathcal{I}^S$, name labels $\mathcal{N}^S$, and the mapping between images and names denoted as Y^S. It then takes a sequence of object images and names in the target domain as inputs. Each object and name pair (ι, n) is added into NEOL's knowledge base, which consists of a set of images $\mathcal{I}^T$, names $\mathcal{N}^T$, and the hierarchy H. NEOL organizes objects and names in a hierarchy H and runs the process described in Figure 6.1 to update H after seeing a new object: It first updates the structure of the hierarchy and then updates the images associated with each node in the hierarchy. Finally, it updates the classifier in each node. We will describe each component in the following sections.

6.3.1 Name Hierarchy

As in Algorithm 3, NEOL maintains a hierarchy of names to organize images and names. For the reasons stated above, existing name hierarchies (such as the WordNet ontology) do not contain all possible names people might use to refer to objects. Furthermore, these hierarchies are built by domain experts and do not necessarily conform to people's ideas of how object names relate to each other. Therefore, we decided to incorporate an online hierarchy learning approach into NEOL. Specifically, we adopt and extend our previous method [185] to build name hierarchies. We summarize the gist of the method and discuss our new extension on the following pages. More

details can be found in [185].

Building a Name Hierarchy

We want to build a hierarchy H over L nodes (names) by asking crowdsourcing workers questions about the hierarchy structure. Our goal is to find the hierarchy that best matches workers' answers in a space $\mathcal{H} = \{H_1, \ldots, H_M\}$ over all possible hierarchies with L nodes. However, workers' answers are inherently noisy. Even if every worker gives the best possible answers, name relationships might be ambiguous and there may be no single hierarchy that consistently explains all the workers' answers. To capture the uncertainty over semantic hierarchies and worker noise, we use a Bayesian framework to estimate probability distributions over hierarchies rather than using a single best guess.

The key challenge here is the huge number of candidate hierarchies, which is $(L+1)^{L-1}$ for $L+1$ nodes (we add one root node), resulting in 1,296 trees for 6 names, but already $2.3579e+09$ trees for only 11 names. It is thus intractable to directly model the distribution as a multinomial. We introduce a compact model to represent the family of distributions over hierarchies

$$P(H|W) = \frac{\prod_{e_{i,j} \in H} W_{i,j}}{Z(W)}, \tag{6.1}$$

where $W_{i,j}$ is a non-negative weight for the edge $e_{i,j}$ representing whether node n_j is an immediate descendent of node n_i, and $Z(W) = \sum_{H' \in \mathcal{H}} \prod_{e_{i,j} \in H'} W_{i,j}$ is the partition function.

Our method updates the model parameter W by asking workers a sequence of questions $q^{(1)}, \ldots, q^{(t)}, \ldots$ and getting the corresponding answers $a^{(1)}, \ldots, a^{(t)}, \ldots$. Each question $q^{(t)}$ is posed to determine whether a direct path $p_{i,j}$ exists in the hierarchy from n_i to n_j. An example would be "*Is an apple a type of fruit?*". The answer $a^{(t)}$ to the question is denoted as $a_{i,j} \in \{0, 1\}$, where $a_{i,j} = 1$ means a worker believes that there is a path from node n_i to n_j, and vice versa. The method then updates the posterior $P(H|W^{(t)})$ at the (t)-th iteration using Bayes' rule

$$P(H|W^{(t)}) \propto P(H|W^{(t-1)}) f(a^{(t)}|H), \tag{6.2}$$

where $f(a^{(t)}|H)$ is the likelihood of getting answer $a^{(t)}$ given the hierarchy H. It can be defined as

$$f(a_{i,j}|H) = \begin{cases} (1-\gamma)^{a_{i,j}}\gamma^{1-a_{i,j}}, \text{ if } p_{i,j} \in H \\ \gamma^{a_{i,j}}(1-\gamma)^{1-a_{i,j}}, \text{ otherwise} \end{cases}.$$

Here $p_{i,j} \in H$ checks the existence of the path $p_{i,j}$ in H and γ is the noise rate of workers.

Unfortunately, the likelihood is not conjugate to the posterior. Hence, there is no analytical form for updating the weights. Sun et al. propose to update the weight matrix by approximate inference. The optimal solution W^* is found by minimizing the KL-divergence [117] between $P(H|W)$ and $P(H|W^{(t)})f(a^{(t+1)}|H)$

$$W^* = \arg\min_W KL(P(H|W^{(t)})f(a^{(t+1)}|H)\|P(H|W)), \tag{6.3}$$

which can be solved efficiently using a sampling method and gradient descent.

To summarize, the algorithm takes a set of names as input and gives a hierarchy with the highest probability as output. The algorithm first initializes the parameters W of the distribution (se the following section). It then poses a sequence of queries to humans. Each answer allows the algorithm to update the distribution parameter W.

Extending A Name Hierarchy

In many cases, `NEOL` needs only to update the hierarchy rather than build it from scratch. Assume that the current hierarchy H has L nodes, and we want to insert a new node into the hierarchy such that the new hierarchy H' has $L+1$ nodes. One naïve way to build the new hierarchy is to run the hierarchy building algorithm from scratch. However, this way ignores all information carried by the previous posterior distribution $P(H|W)$. We adopt a more informative way to initialize the algorithm. Using basic rules of probability, it is true that

$$P(H'|W) = \sum_H P(H'|H)P(H|W), \tag{6.4}$$

where $P(H'|H)$ is a uniform distribution over all the possible hierarchies by adding one node into H, and $P(H|W)$ is the distribution over H so far, which is the output of the algorithm described in the previous section.

Since there exist efficient algorithms to sample i.i.d. (independent identically distributed) samples from $P(H|W)$ [185], it is then also efficient to get i.i.d. samples from $P(H'|W)$ using ancestral sampling [112]. Then, we initialize W' by optimizing the following problem

$$W'^{*} = \arg\min_{W'} KL(P(H'|W)\|P(H'|W')), \tag{6.5}$$

which can be solved using the same technique for optimizing equation (6.3).

After initializing W', the algorithm iteratively updates W' by following the same algorithm of building a hierarchy.

Implementation Details

Though using crowdsourcing to answer a single question is inexpensive, answering all relevant questions via crowdsourcing is still very costly, especially for building large hierarchies. To save questions, the approach from [185] selects the sequence of path questions based on information gain, thereby reducing the number of questions needing to be asked.

Additionally, due to the noise in workers' answers, the same question has to be answered by multiple workers to improve confidence. Fortunately, we can avoid asking a lot of questions using source domain semantic hierarchies, *e.g.*, the WordNet. For example, if both nodes n_i and n_j are in the WordNet hierarchy, we can use WordNet to answer the path question $p_{i,j}$ by checking whether n_i is a hypernym of n_j in WordNet. Since WordNet is built by experts, the error rate γ for that answer would be very small. On the RGB-D dataset [122], we found that using WordNet to answer questions saves about 40% of the queries, also resulting in a learned hierarchy that is more consistent with WordNet if there are overlapping nodes between them.

After collecting sufficient information from workers and WordNet, our approach produces a very concentrated probability distribution over a few hierarchies. `NEOL` uses the maximum a

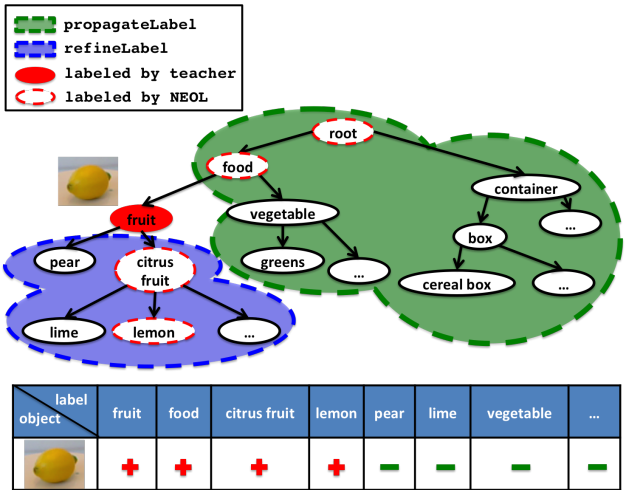


Figure 6.2: Label propagation using a hierarchy. The object given to the robot is a lemon. The name given by a person is *citrus fruit*, which is marked as a solid red ellipsis in the hierarchy. The lemon will be used as a positive example of *citrus fruit*. It will also be used as a negative or positive example for other nodes in the hierarchy. To decide whether it is a positive or negative example of a particular node, we define two sub-routines, `propagateLabel`, indicated by the green region, and `refineLabel`, indicated by the blue region. After applying both routines to the object, all its annotations will be filled in as shown in the table bellow the hierarchy.

posteriori (MAP) hierarchy for label propagation. Finding the MAP tree of the distribution defined in equation (6.1) can be accomplished efficiently using the minimum-spanning tree algorithm [38].

6.3.2 Consistent Image Annotation

As we will show in our experiments, organizing object names into a hierarchy significantly improves the training of visual object classifiers. This is mostly because a hierarchy can help to label objects more accurately and efficiently.

Specifically, a human teacher might teach a robot a few names for the object. However, the teaching is often not complete. The incomplete annotation of names causes sparsity of training data and even confusion for image annotations. For instance, imagine a lemon is presented to the robot along with a name *fruit*. This lemon should obviously be treated as a positive training example

of *fruit*. Unfortunately, it is not obvious how to use it for the remaining names in the hierarchy. One possible method is not using it at all for training classifiers of the other nodes except for *fruit*. However, this method could cause a great loss of negative examples. In fact, there will be no negative examples unless a human explicitly says that an object cannot be referred to by certain names. Another way of using the object is to use it as a negative example for all nodes except those mentioned by the human teacher as positive labels. However, this naïve way would introduce many false negative labels and might result in inaccurate classifiers.

To resolve this problem, we propose a method described in Algorithm 4 to ensure that every object is correctly labeled for learning every name in the hierarchy. The goal of Algorithm 4 is to put every object image $\iota \in \mathcal{I}^T$ into either $\mathcal{I}^+$ or $\mathcal{I}^-$ of every node $\tilde{\mathtt{n}}_l$ in the hierarchy H.

The algorithm has two tasks. First, if it sees a new name inserted into the hierarchy, it updates labels of previously seen objects, so that it knows whether some previous objects are also positive examples of the new name. Second, the algorithm needs to determine whether the newly provided object is a positive example of previously seen names. Both tasks can be fulfilled via two sub-routines, `propagateLabel` and `refineLabel`.

Propagate Label

By knowing the hierarchical organization of names, a robot could determine some labels of an object without acquiring extra knowledge. Take *fruit* in Figure 6.2 as an illustrative example. If an object is labelled as *fruit*, we can determine whether it is a positive or negative example for any non-descendant nodes of *fruit* in the hierarchy. The green region in Figure 6.2 illustrates nodes covered by this type of propagation.

For a new node $\tilde{\mathtt{n}}$ inserted into the hierarchy, we can initialize $\tilde{\mathtt{n}}.\mathcal{I}^+$ and $\tilde{\mathtt{n}}.\mathcal{I}^-$ using the hierarchy H (line 4 – 5). Using the example in Figure 6.2, any particular types of fruit, such as pear and citrus fruit, are also fruits, so that $\tilde{\mathtt{n}}$ takes positive images belonging to its child nodes also as positive. On the other hand, if an object is not a food, it could not be fruit. Therefore, $\tilde{\mathtt{n}}$ inherits negative examples of its ancestor nodes as negative. Furthermore, fruit and vegetable are exclusive according to the hierarchy, suggesting that images referred to by any siblings of $\tilde{\mathtt{n}}$ should also be put into $\tilde{\mathtt{n}}.\mathcal{I}^-$.

Algorithm 4 `updateAnnotation`

1: **Input:**
   H: hierarchy
   ι: new object image
   $\tilde{\mathtt{n}}$: node in H
2: **Output:**
   H: updated hierarchy
3: **Initialization:**
4: **if** $\tilde{\mathtt{n}}$ is a new node **then**
5: `initialize`($\tilde{\mathtt{n}}.\mathcal{I}^+$, $\tilde{\mathtt{n}}.\mathcal{I}^-$, H)
6: $\tilde{\mathtt{n}}.\mathcal{I}^+ := \tilde{\mathtt{n}}.\mathcal{I}^+ \cup \{\iota\}$
7: **for** $\tilde{\mathtt{n}}' \in$ `ancestors`($\tilde{\mathtt{n}}$) **do**
8: $\tilde{\mathtt{n}}'.\mathcal{I}^+ := \tilde{\mathtt{n}}'.\mathcal{I}^+ \cup \{\iota\}$
9: **for** $\tilde{\mathtt{n}}' \notin$ (`ancestors`($\tilde{\mathtt{n}}$) $\cup$ `descendants`($\tilde{\mathtt{n}}$)) **do**
10: $\tilde{\mathtt{n}}'.\mathcal{I}^- := \tilde{\mathtt{n}}'.\mathcal{I}^- \cup \{\iota\}$
11: $\tilde{\mathtt{n}}' := \tilde{\mathtt{n}}$
12: **while** $\tilde{\mathtt{n}}'$ is not a leaf node **do**
13: $\prec$:= s.t. $a \prec b$ iff
   $\tilde{\mathtt{n}}_a.f(\iota) > \tilde{\mathtt{n}}_b.f(\iota), \forall\, \tilde{\mathtt{n}}_a, \tilde{\mathtt{n}}_b \in$ `children`($\tilde{\mathtt{n}}'$)
14: $\tilde{\mathtt{n}}' :=$ `refineLabel`(H, ι, $\tilde{\mathtt{n}}'$, $\prec$)
15: **if** $\tilde{\mathtt{n}}'$ = `nullptr` **then**
16: break
17: $\tilde{\mathtt{n}}'.\mathcal{I}^+ := \tilde{\mathtt{n}}'.\mathcal{I}^+ \cup \{\iota\}$
18: **if** $\tilde{\mathtt{n}}$ is a new leaf node **then**
19: **for** $\iota' \in$ (`parent`($\tilde{\mathtt{n}}$).$\mathcal{I}^+ - \cup_{\tilde{\mathtt{n}}_l \in \mathtt{sibling}(\tilde{\mathtt{n}})} \tilde{\mathtt{n}}_l.\mathcal{I}^+$) **do**
20: **if** `isA`(ι', $\tilde{\mathtt{n}}.n$) **then**
21: $\tilde{\mathtt{n}}.\mathcal{I}^+ := \tilde{\mathtt{n}}.\mathcal{I}^+ \cup \{\iota'\}$
22: **else**
23: $\tilde{\mathtt{n}}.\mathcal{I}^- := \tilde{\mathtt{n}}.\mathcal{I}^- \cup \{\iota'\}$
24: **Return:** H

Similarly, the new object ι is also a positive example of any ancestors of $\tilde{\mathtt{n}}$ (line 7 – 8 in Algorithm 4), and a negative example for nodes that are not ancestors or descendants of $\tilde{\mathtt{n}}$ (line 9 – 10).

Refine Label

Unfortunately, it is still unclear whether the *fruit* is a *lemon*, *pear* or some other fruit. We propose to ask crowdsourcing workers questions to annotate the object with any descendant nodes of *fruit*. The blue region in Figure 6.2 illustrates nodes handled by this type of annotation, referred to as `refineLabel`.

The goal of `refineLabel` is to find more specific names of an object. Without hierarchy, this can be implemented by asking workers to verify whether an object has a name for every possible name. It should be noted that, without hierarchy, all annotations labelled via `propagateLabel` need to be answered by asking workers, as well. Therefore, the total cost of annotating objects could be very expensive.

A hierarchy captures the semantic relationship between names; that relationship can help to prevent redundant questions. If the robot knows that the fruit is a citrus fruit, there is no need to ask whether the fruit is a *pear* or an *apple*. Once there is a "yes" answer to one of the nodes, all nodes at the same level and all their descendants will be marked as negative except for the "yes" node. Then we can recursively refine the new "yes" node until it is a leaf node or there is no more "yes" node. Line 12 – 17 of Algorithm 4 describe the idea of recursively refining object labels.

The robot can increase the efficiency of finding positive nodes during `refineLabel` by using its already trained classifiers. Again, take *fruit* as an example. If the image is more like a *pear* than an *apple*, the robot should ask whether the object is a *pear* first. To implement this idea, all $\tilde{\mathtt{n}}_l.f$ are applied to the current image ι. The scores are used to rank the possible names of the object (line 13). `refineLabel` searches positive labels according to this order and returns a positive node if finds one. Otherwise, the search stops. ι will become a negative instance for any nodes that have not been visited. As we will show, this strategy saves a significant number of questions.

We also need to refine labels of previously seen objects if a new node is inserted into the hierarchy as a leaf node. Fortunately, we need to refine only labels of objects belonging to the parent node, but not any siblings of the new node. Take *citrus fruit* as an example. Assume an orange is labelled as *citrus fruit* in the previous iteration. It does not belong to any other citrus fruit according

to our algorithm. After the new node *orange* is inserted into the hierarchy, it is necessary to refine the label of the orange, to associate it with the *orange* node as a positive example. Specifically, our algorithm checks all images, the most specific label of which is the parent node of the new node, by asking crowdsourcing workers whether it can be referred to by the new name. Line 18 – 23 of Algorithm 4 describe this refinement to the previously seen objects.

In summary, `propagateLabel` does not require human annotations and, can therefore be performed for free. Yet, it cannot annotate all object labels of an object. `refineLabel` complements `propagateLabel` by asking crowdsourcing workers a few questions to efficiently annotate the remaining missing labels of an object. It is easy to verify that after applying both strategies, all possible labels of all accumulated images are filled in.

6.3.3 Leveraging Background Image Datasets

A challenge in the never-ending learning setting is that a robot might be unable to acquire a large number of training examples for each label. For instance, it might initially see only a single soda can, which makes it extremely hard to generalize to other types of soda cans. To increase generalization capabilities, `NEOL` uses additional training images from an existing background or source domain dataset, such as ImageNet [50]. The source domain dataset has a large amount of RGB images for a set of categories $\mathcal{N}^S = \{n_1^S, \ldots, n_N^S\}$. `NEOL` uses the same feature extraction method applied to target domain RGB images to extract features $\boldsymbol{x}_{1,RGB}^S, \ldots, \boldsymbol{x}_{M,RGB}^S$ from source domain images. We will show how to incorporate images and labels from a background dataset to build classifiers that generalize well in the robotics target domain.

The first issue that needs to be handled is that the external dataset might have different annotations from the target domain annotations. For this reason, `NEOL` uses only the source domain names that also exist in the target domain. Using Algorithm 4, it is straightforward to put objects from the source domain into nodes in the hierarchy. Since *refineLabel* involves human annotators, it could be costly to label all source domain images. Meanwhile, labelling source domain images is not as useful as labelling target domain images. So, we skip *refineLabel* for source domain images, and use only the existing labels and *propagateLabel*, corresponding to lines 7 – 8 and lines 9 – 10 in

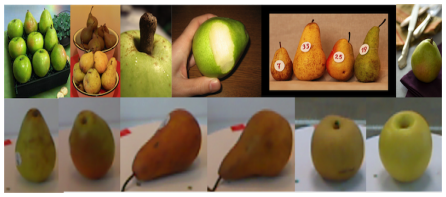
Algorithm 4.

Even if the source domain and the target domain images are from the same category, the images can be very different, causing a significant performance drop when the classifiers built on the source domain are applied directly to the target domain. To understand how different they are, we trained object recognition classifiers on the ImageNet dataset [50] and tested them on the RGB-D object dataset [122]. We found the average precision across all objects on the RGB-D object dataset to be 33% lower than the average precision across the same type of objects on the ImageNet test set. This drop in performance is not due to the difficulty of the RGB-D dataset, but rather due to the fact that ImageNet images have very different and diverse appearances. For example, cereal boxes can be classified on the ImageNet test set with more than 90% precision. However, the same classifier has less than 50% precision on the "easier" cereal box images in the RGB-D dataset.

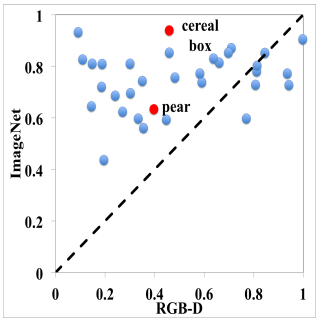
To deal with such a domain-mismatch, we propose a method based on the observation that the source domain dataset is often a combination of several distinct sub-groups, each of which has its own characteristics [76, 92]. Some groups are more similar to the target domain, while others are not. For example, the top row of Figure 6.3 (a) shows images of cereal boxes from ImageNet. Some are more similar to the cereal boxes in the RGB-D dataset (bottom row), which are mostly single cereal box images. Some are not, *e.g.*, images with cereal boxes on a shelf. If we model the source domain dataset using a single model, the discriminative information from the different groups will be mixed together, diminishing the benefit of adding the source domain. To avoid this, we group source domain images into different sub-groups and learn a specific classifier for each sub-group. For grouping, we tried both sophisticated methods [76] and simple clustering methods, such as K-means. The performance of the different methods was very similar in our context, so we decided to use K-means to partition the full source-domain dataset into K groups.

Next, we build one classifier for each sub-group by combining all the target-domain RGB and depth images with the source domain images from that sub-group. A classifier for the k-th sub-group

(a) *cereal box* samples from ImageNet (top) and RGB-D (bottom).



(b) *pear* samples from ImageNet (top) and RGB-D (bottom).



(c) Average precisions of ImageNet classifiers tested on ImageNet vs RGB-D.

Figure 6.3: ImageNet images have a very different appearance them RGB-D images. Recognition models built using the ImageNet dataset perform poorly on the RGB-D dataset.

in the l-th category is learned by optimizing the objective

$$\begin{aligned}\boldsymbol{w}_{l(k)} = \arg\min_{\boldsymbol{w}} \|\boldsymbol{w}\|^2 + C_1 \sum_{m=1}^{M_k^T} h\left(\boldsymbol{w}'_{RGB}\boldsymbol{x}^T_{m,rgb}, y_m^{(l)}\right) \\ + C_1 \sum_{m=1}^{M^S} h\left(\boldsymbol{w}'_{RGB}\boldsymbol{x}^T_{m,rgb}, y_m^{(l)}\right) \\ + C_2 \sum_{m=1}^{M^S} h\left(\boldsymbol{w}'_{depth}\boldsymbol{x}^T_{m,depth}, y_m^{(l)}\right), \forall l \in \{1, \ldots, L\}\end{aligned}$$

where $\boldsymbol{w} = [\boldsymbol{w}'_{RGB}, \boldsymbol{w}'_{depth}]'$, $h(\cdot,\cdot)$ is the hinge loss, and M_k^T are the target domain images in the k-th cluster. C_1 and C_2 balance the importance of RGB and depth channels. This objective function incorporates RGB images from both the source and the target domains, as well as depth information that exists only for the target domain. When applying the models to predict new images from class l, we average the prediction scores given by all K sub-group classifiers.

Table 6.1: Comparison of category level classification accuracies for the RGB-D object dataset. LOIO: Leave One Instance Out. 1I/C: 1 Instance per Category. 1V/O: 1 View per Object. Differences in accuracy relative to baseline methods are marked by colors (red: performance improvement, blue: performance decrease). All numbers are averages over 10 random data splits.

method	Category Accuracy(%)		
	LOIO	1I/C	1V/O
	RGB/D	RGB/D	RGB/D
Baseline [169]	83.6/89.2	58.8/67.3	65.1/70.2
Baseline-S	−58.7/ − −−	−33.9/ − −−	−40.2/ − −−
Baseline-(T+S)	−6.4/−5.3	−5.0/−3.9	−20.1/−17.9
GFK[76]	−53.2/ − −−	−28.4/ − −−	−34.7/ − −−
Fru[47]	−1.1/−1.8	+1.4/−0.5	−6.4/−4.9
NEOL	+1.3/+0.9	+5.5/+5.9	+4.6/+2.8
NEOL	**84.9/90.1**	**64.3/73.2**	**69.7/73.0**

6.4 Experiments

We evaluate the properties of NEOL via two sets of experiments. First, we evaluate the method used by NEOL to leverage background image databases in Section 6.4.1. Next, we evaluate the full NEOL system in Section 6.4.2. The hierarchy building method described in 6.3.1 has been demonstrated to be effective in [185], and we refer readers to that paper for more detail.

6.4.1 Leveraging Background Image Databases

In this section, we test our approach to deploying background image datasets for building better target domain classifiers. We use a popular RGB-D dataset [122] as the target domain dataset; it consists of 300 objects falling into 51 object categories. We use ImageNet [50] as the source domain by taking the images from the same 51 categories in ImageNet. As in [169], we evaluate the recognition accuracy in multi-class classification.

To evaluate the performance of our approach, we compare it to the following baseline approaches:

- `Baseline`: Uses only target domain images following [169].
- `Baseline-S`: Uses only source domain images to train classifiers.
- `GFK`: Uses a domain adaptation method using only source domain images [75]. It is a state-of-the-art unsupervised domain adaptation method.
- `Baseline-(T+S)`: Combines the source-domain and target-domain images into a larger-size training set.
- `Fru`: Uses frustratingly-easy domain-adaptation, is a supervised domain adaptation method [47] that achieves state-of-the-art performance in many domain-adaptation tasks.

All the methods take features extracted via a pre-trained CNN implemented in Caffe [101]. A linear SVM is used as the classifier for all methods unless a method specifies a classifier. For `NEOL`, the parameter K is set to 10 clusters in all experiments; (we tried larger Ks but did not observe a significant difference).

Results are summarized in Table 6.1, where the first and last rows give absolute accuracies. Other rows provide changes relative to `Baseline`.

Leave One Instance Out (LOIO)

For the first experiment, we conduct the Leave-One-Instance-Out test following the protocol in [122]. Each object in both training and test sets has 90 views uniformly and randomly sampled from the full dataset.

The target-domain only baseline method using RGB-D information achieves 89.2% accuracy. Surprisingly, adding source domain images decreases the performance except when using `NEOL`'s adaptation technique. Methods using only source domain images do not consider the target domain; They therefore perform the worst. Simply combining source-domain and target-domain data

without adaptation also harms the performance (`Baseline-(T+S)`). Frustratingly-easy domain adaptation (`Fru`) builds a single domain adaptation model using a feature replication method to bridge the gap between domains. Although it improves the performance over the naïve combination, it still performs worse than the baseline method, which uses only target domain data. Our approach outperforms the baseline method, with a 1.3% and a 0.9% improvement on RGB and RGB-D data, respectively. This improvement over the baseline is statistically significant.

It is worth noting that the LOIO setting in this experiment has a large amount of target-domain training instances and poses. However, in never-ending learning, the target domain training instances may be sparse. We therefore test the competing methods under the small-training-set setting in the following experiments.

One Instance per Category (1I/C)

To compare to LOIO, we adopt the same testing protocol and leave one instance out of each category as our test set. The second column of Table 6.1 shows the results. The performance of the baseline drops to 67.3% (RGB-D) as it overfits to the small training set. `Baseline-(T+S)` and `Fru` both use extra source domain images, yet they do not manage to improve the performance over the baseline due to the failure of handling domain mismatch. On the other hand, `NEOL` achieves a 5.9% improvement over the baseline, suggesting that `NEOL` can transfer object knowledge from the source domain to the target domain.

One View per Object (1V/O)

For the third experiment, we evaluate how different methods handle variance of object poses. We follow the leave-one-instance-out setting in the first experiment, but we give only 1 view for each training instance (1V/O in Table 6.1). `NEOL` obtains a 2.8% improvement over the baseline methods, which shows its advantage in handling the variance between object poses.

To summarize, the domain adaptation performed by `NEOL` is the only technique that can take advantage of additional image data available in ImageNet. Furthermore, the improvements are not

drastic with sufficient target domain objects and views since the CNN features already incorporate a lot of stability with respect to feature invariances.

6.4.2 Never-Ending Object Learning

We now evaluate `NEOL` on the task of never-ending object learning. We assume a person is continuously teaching a robot different objects. For each object, the person shows the robot a few views of the object and one name that can be used to refer to it. We simulate this setting using the RGB-D object dataset [122]. We do a $50:50$ split, so 150 objects will be used for sequential training, and the remaining 150 objects will be used for testing. At each iteration, one of the objects is randomly picked from the training set and shown to the robot. The robot will be able to perceive 3 views of the object and get one name of the object from a worker at AMT. Here, workers act as teachers for the robot. The iterative process repeats for $2,000$ iterations in each trial. We repeated 10 random trials and report the average performance.

At each iteration, the robot uses the `NEOL` (Algorithm 3) to improve the system: first, if a name is new, it inserts the name into the hierarchy using `BuildingHierarchy` described in 6.3.1; second, it updates the training example annotations using Algorithm 4 described in 6.3.2, which includes `propagateLabel` and `PropagateDown`; finally, it updates the classifiers of existing names using the domain adaptation method (`Adaptation`) per section 6.3.3.

For both `BuildingHierarchy` and `refineLabel`, we use AMT to get answers to the questions asked by `NEOL`. To repeat the experiments with different configurations, we collected answers to all possible questions beforehand using AMT. Later, we simulate the online responses using the collected answers. Specifically, for each question related to building hierarchies, we follow the original paper [185] and simulate 8 workers to answer it and use the maximum likelihood estimate of the noise rate from the real AMT answers to generate worker noise. For the questions asked by `refineLabel`, we simulate 3 independent workers to answer the same question. Each worker has a noise rate (< 0.5) such that he/she flips the ground truth answers with the probability of the noise rate. The majority vote of the 3 workers is used as the final answer of a question. We tried different noise rates in our experiments.

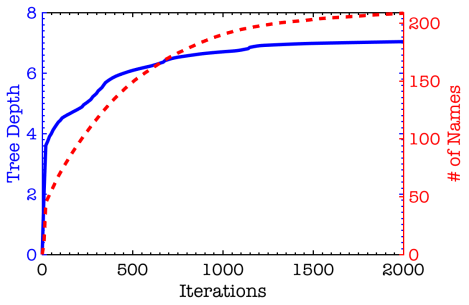


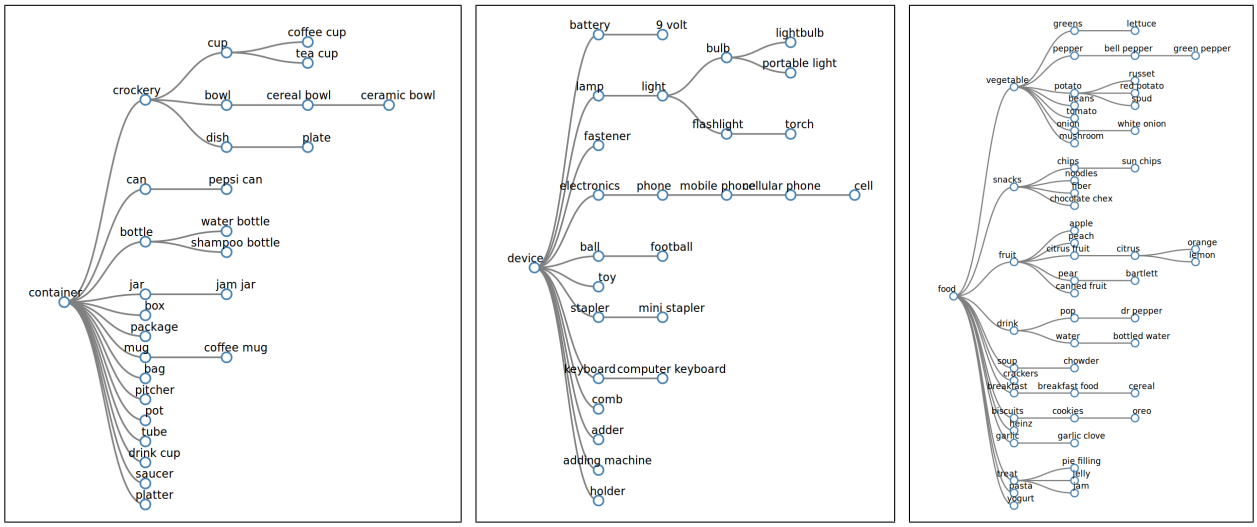
Figure 6.4: The growth of the tree depth and number of unique names as objects are presented.

We use mean average precision (mAP) as our evaluation metric. Given a testing image, an approach being evaluated outputs a prediction score for each name known to the system. Then we compute the average precision (AP) for each name by ranking the test images by the classification scores. The average AP for all names gives the mAP. The performance of the robot is evaluated at each iteration on both previously seen and unseen objects.

Building Hierarchies

NEOL maintains a hierarchy of names during the sequential learning. Figure 6.4 plots some important statistics of the hierarchies built by NEOL. The red curve shows the number of unique names (number of nodes in the hierarchy) used by workers to refer to objects. It increases almost linearly for the first 200 iterations because almost all the objects are new to the robot, and new objects yield new names. The number of names still grows between 200 and $1,500$ iterations because the fact that different people use different names to refer to the same objects. After $1,500$ iterations, the number remains flat as new names are rare. The curve showing the depth of the generated MAP hierarchy has a similar trend. The hierarchy has about 4 levels after about 50 iterations, and eventually it grows to 7 levels. Across different trails, the hierarchies built by the system are consistent, showing that the hierarchy building algorithm is robust. Figure 6.5 shows the 3 largest sub-trees in the hierarchy at $2,000$ iterations.

We also compare the NEOL hierarchy to the corresponding hierarchy in WordNet, which is the



(a) `Container` sub-tree. (b) `Device` sub-tree. (c) `Food` sub-tree.

Figure 6.5: Three largest sub-trees in the hierarchy built by `NEOL` at $2,000$ iterations.

minimum sub-tree in WordNet covering as many names used by people as possible. As discussed before, not all names used by people can be found in WordNet. In this experiment, we find that among 207 names, only 142 are also in WordNet. To find the minimum sub-tree covering the 142 names, we first find the lowest common ancestors of all pairs of names; we then recover the tree according to WordNet over the union of the 142 names and their ancestors. The final sub-tree includes 193 nodes and has 11 levels in depth. The numbers shows that the WordNet hierarchy is usually more complex than the hierarchy crowdsourced by humans.

Filling Annotations via Crowdsourcing

`NEOL` asks workers questions to build hierarchies and annotate objects. We evaluate the efficiency of `NEOL` using the number of accumulated questions over $2,000$ iterations as the evaluation metric. Figure 6.6 shows the accumulated questions asked by different methods.

We compare the following methods:

- `refineLabel`: As described in Algorithm 4. It uses the hierarchy and the learned image

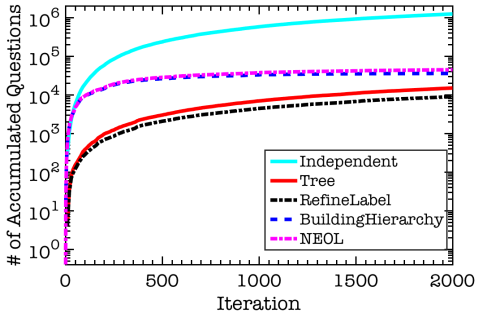


Figure 6.6: Number of accumulated questions asked by different approaches. Y-axis is in log-scale.

classifiers to determine the order in which questions about names in the tree are asked.

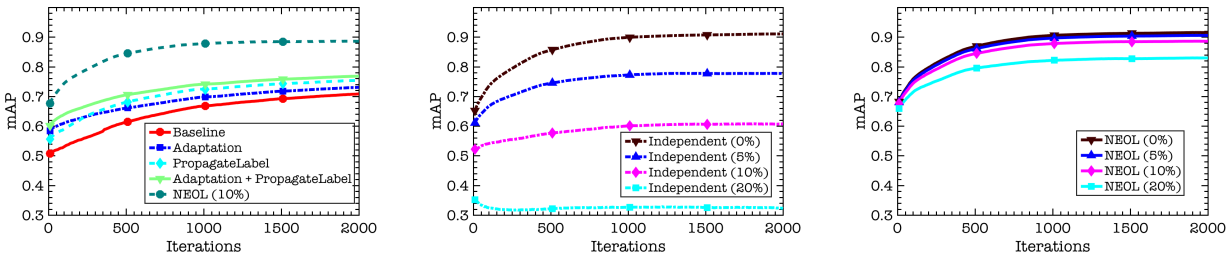
- `Tree`: A variation of `refineLabel` that uses only the hierarchy and ignores the image classifiers as guidance.

- `Independent`: Treats all names independently and asks workers about all possible object-name pairs.

- `NEOL`: The total number of questions asked by `NEOL`, which is the total number of questions asked to build the hierarchy and to propagate labels through the hierarchy (`refineLabel` + `refineLabel`).

According to the plots in Figure 6.6, `NEOL` performs much better than `Independent`. The number of questions asked by `Independent` grows linearly in the product of number of objects and number of names, and it quickly explodes as more names and objects are added into the system. It has asked $574,000$ questions in total for the first $1,000$ iterations. At the end of $2,000$ iterations, the number of accumulated questions asked by `Independent` reaches $1,232,000$. `Independent` ignores both relationships between names and the object knowledge learned by the system. As a result, it must ask a huge amount of redundant questions.

On the other hand, `NEOL` asks far fewer questions. Comparing `refineLabel` to `Tree` shows that using the learned image classifiers as guidance helps `NEOL` to more quickly locate



(a) mAP scores of different methods.

(b) mAP scores of baseline method under different levels of noise.

(c) mAP scores of NEOL under different levels of noise.

Figure 6.7: mAP scores of different methods under different experimental settings.

the correct labels of objects. refineLabel therefore manages to save about 50% of the total questions when compared to Tree, an approach that does not use image classifiers to propagate labels. Building the hierarchy costs some questions, especially in early iterations. The number of questions needed to build and update the hierarchy quickly converges as far fewer new names are shown to the system as time passes. The total number of questions asked by NEOL, which is the summation of BuildingHierarchy and refineLabel, is $8,484$ and $13,300$ after $1,000$ and $2,000$ iterations, respectively. When compared to the $574,000$ and $1,232,000$ questions required by Independent, building and using the hierarchy maintained by NEOL saves more than 98% of the questions for consistent image labeling.

Recognition Performance

We evaluate the recognition performance of NEOL and other baseline approaches at each iteration. The test set consists of all 300 objects in the RGB-D dataset. We remove all views of training objects seen in the training stage from the test set such that the test set contains either new objects or new views of previously seen objects. We report the average mAP scores for the first $2,000$ iterations in Figure 6.7.

Figure 6.7 (a) shows the performance of different methods. We compare several methods besides NEOL and Baseline, where Baseline uses only the label names provided with each new image

to generate positive training examples for each name classifier. All non-positive examples are treated as negative examples. NEOL, on the other hand, uses the hierarchy to generate consistent labels for each image and each name.

- Adaptation: Adds the adaptation technique deployed by NEOL to the baseline method.
- propagateLabel: Deploys the hierarchy to propagate labels up and trains the baseline with the updated object annotations.
- propagateLabel + Adaptation: Adds both Adaptation and propagateLabel to Baseline.

We first compare the performance of different methods that do not query humans for object annotations (but possibly do for building the hierarchy). After the first iteration, propagateLabel + Adaptation attains an mAP of 0.6, while the baseline gets only 0.5. Since there is only one name, using the hierarchy does not yet yield useful information. Adaptation and propagateLabel + Adaptation thus perform almost identically, suggesting that, at the early stage, the gain of NEOL is mainly due to domain adaptation. Using hierarchy becomes effective after about 50 iterations. As the hierarchy gets deeper, it can capture more meaningful semantic relations between names. As can be seen in Figure 6.7 (a), propagateLabel surpasses Adaptation at around 200 iterations.

After $1,000$ iterations, the marginal improvement of propagateLabel + Adaptation over the baseline is still over 10%. Eventually, the performance of both methods converges as more and more training examples are shown to the robot. Nevertheless, propagateLabel + Adaptation retains more than 5% advantage until $2,000$ iterations, showing that it can learn new objects and names more efficiently than the baseline method.

The full NEOL adds refineLabel on top of propagateLabel + Adaptation to further improve consistency of object annotations. By adding these annotations into the training, NEOL manages about a 15% marginal improvement over refineLabel + Adaptation, a very significant amount.

Table 6.2: Overall annotation error rates under different levels of per-worker noise.

method	Per-Worker Noise Rate (%)				
	1	5	10	20	30
Independent	0.028	0.717	2.76	10.3	21.3
NEOL	0.002	0.046	0.175	0.64	1.36

We also test the recognition performance of Independent. This approach is not only extremely inefficient (Figure 6.6), but it is also not very robust to annotation noise, which harms the performance of the whole system. To verify, we vary the per-worker noise (to$0\%, 1\%, 5\%, 10\%, 20\%$) and apply Independent and NEOL to train and test on exactly the same setting. With no noise (0%), Independent achieves comparable performance to NEOL. However, as per-worker noise increases, the performance of Independent drops significantly. When the per-worker noise rate is around 10%, Independent performs even worse than the baseline method that asks no extra queries at all (Baseline in Figure 6.7 (a)). On the other hand, NEOL, using Algorithm 4 to annotate objects, is very robust to noise. Its performance does not drop too much even with 10% per-worker noise, and it drops less than 10% when the per-worker noise rate reaches 20%.

A detailed examination indicates that NEOL is not sensitive to noise, since it asks far fewer questions by making more judgements based on hierarchy information. But the baseline method depends totally on the workers' answers; therefore, it is more sensitive to the noise of individual workers. Table 6.2 shows the overall error rate of training object annotations achieved using NEOL and Independent under different levels of per-worker noise. The overall error rate of the training target data set grows quickly as the per-worker noise rate increases when the baseline method is applied. NEOL is robust to the noise and has less than a 1% overall error rate of annotations even when per-worker noise rate reaches 20%. All numbers are achieved by asking 3 workers each question.

To summarize, different components of NEOL contribute variously at different learning stages. When just a few examples are shown to the robot, Adaptation helps more significantly. As the

hierarchy of names becomes more informative, most gains are from the hierarchy. Labelling missing names of objects further improves the performance of NEOL with a large margin. Since NEOL uses both hierarchy and existing classifiers to guide object annotations, the annotation process is very efficient and robust to noisy answers given by crowdsourcing workers.

6.5 Discussion

6.5.1 NEOL

This chapter presents the never-ending learning framework NEOL that helps robots to learn new objects and names over their lifetimes. NEOL uses crowdsourcing to organize the stream of object images and names into a semantic hierarchy. It uses the hierarchy and accumulated image information to ensure that object annotations are consistent and complete. It also includes a domain adaptation method that combines images from existing image databases with images the robot perceives in order to better generalize to unseen objects.

Our experiments on the RGB-D dataset using ImageNet and WordNet as additional background information demonstrate that NEOL can learn objects efficiently and effectively, under the setting of never-ending object learning. The experiments also show that building and using a name hierarchy for consistent image annotation significantly improves NEOL's capabilities. Using ImageNet as additional background knowledge further improves performance, while improvements due to domain adaptation are most significant when only small sets of objects are perceived in the environment.

6.5.2 Future Work

In this work, NEOL organizes names in a semantic hierarchy, which covers most common relations between names. However, there are still other types of relations between names that are not covered by the hierarchy, for example, co-occurrence [55] relationships. We plan to extend the hierarchy building algorithm in NEOL to discover other types of relations between names that could potentially help robots learn objects more efficiently and effectively.

In this work, NEOL uses the MAP hierarchy over names to propagate annotations. However,

there may not always be a unique, correct hierarchy. For example, some people think "tomato" is a type of "fruit", while others consider it to be a "vegetable". Our hierarchy-building algorithm [185] can capture such uncertainty over hierarchies, but it would be interesting to consider uncertainty in building a recognition system that is robust to "miscommunication" between humans and robots.

Videos illustrating NEOL are available at
http://rse-lab.cs.washington.edu/projects/neol.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Contributions

This thesis has contributed to robotic object learning research and implementation in three key ways. First, it studied object recognition based on natural language description. In particular, we studied how a robot can recognize objects based on attributes (Chapter 2) and identify objects based on new names (Chapter 3). Second, it defined a method for building hierarchies of concepts via crowdsourcing. Specifically, we studied three problems in the construction of semantic hierarchies of concepts: learning a distribution over hierarchies (Chapter 4), evaluating the quality of hierarchies via crowdsourcing experiments (Chapter 5), and learning user-dependent hierarchies (Appendix B). Third, it offered a framework of never ending object learning for robots (Chapter 6).

7.1.1 Language-Based Object Identification

Language-based object identification enables a robot to recognize target objects based on a human user's language description. In this thesis, we propose an object identification system that uses appearance attributes and names described by natural language to identify objects (Chapter 2 and Chapter 3). To handle the case when a robot may encounter new names, we design a metric learning algorithm (Chapter 3) to associate new names to existing names, so the system can identify objects based on new names without even seeing any training instances of the new names. The overall system enables a robot and a human to interact using natural language, and it lets the robot recognize both seen and unseen objects based on language description.

7.1.2 Building a Hierarchy via Crowdsourcing

We propose a principle algorithm (Chapter 4) to build hierarchies of concepts via crowdsourcing. The proposed method can be applied as a general tool to create task-dependent hierarchies for various applications. Experiments demonstrate that our algorithm is efficient and able to build expert-level hierarchies. We further extend the algorithm to infer the hierarchies used by each person (Appendix B) and therefore enable a robot to provide personalized services based on user-specific hierarchies. Finally, we study the evaluation criteria used to assess the quality of hierarchies. Specifically, we design a crowdsourcing experiment to assess the quality of hierarchies with respect to how well they facilitate human users accomplishment of an image navigation task.

7.1.3 Never-Ending Object Learning for Robots

Finally, we combine the techniques proposed in the first few chapters of this thesis to build a framework of never-ending object learning for robots (Chapter 6). The proposed framework- `NEOL`- enables a robot to learn new objects continuously by interacting with human users via natural language. The robot organizes learned concepts in semantic hierarchies and uses the hierarchy to improve efficiency and accuracy of object annotations. The robot can also combine external image resources to learn to recognize objects with just a few training instances. The never-ending object learning framework could run 24/7 on a real robot and help the robot learn an increasing number of objects more efficiently and more accurately over its lifetime.

7.2 Future Work

Enabling a personal robot to learn various kinds of knowledge over its lifetime will be crucial to making it continuously useful. In this thesis, we focused on enabling a robot to learn various kinds of object knowledge. Based on our findings, we find the following directions appear particularly promising for future work.

7.2.1 Crowdsourcing for Robots

Although we have witnessed vast improvements in robotics research, performing complicated tasks remains quite challenging for a household robot. As a very practical example, we give a robot a recipe for making an Omelet, ask it to read the recipe, and then to make the Omelet for us. Unfortunately, current robots cannot perform this simple, practical task. There are multiple challenges in doing so: 1) reading the recipe to extract micro-tasks (units of action that should be conducted together, *e.g.*, cracking an egg, beating an egg), 2) generating a precise execution graph of these micro-tasks based on the recipe, 3) performing micro-tasks if the robot has never performed them previously. Our experience from the findings of this thesis suggest that we can involve humans in the loop of robotic learning, specifically, by hiring crowdsourcing workers to teach robots various abilities.

A promising direction is to include crowdsourcing teachers to help a robot understand the semi-structured natural language in certain application domains, *e.g.*, cooking recipes. The research questions would include how to initialize the understanding of recipes using NLP and CV techniques, how to design crowdsourcing tasks to improve the initial understanding, and how to transfer the learned knowledge to the understanding of new recipes. Another direction is to involve crowdsourcing teachers in teaching robots some simple operations, *e.g.*, cracking and beating eggs.

Based on the findings of this thesis, we believe that crowdsourcing could be an accurate and cost-effective resource to help autonomous robots learn various kinds of knowledge and skills.

7.2.2 Systems

A never-ending learning robot will gather an overwhelming amount of language, visual and semantic data. In this case, how to store, manipulate and analyze the data pose challenges on both the algorithmic and system domains. On the algorithm side, a challenge is how to scale up learning algorithms, and crowdsourcing algorithms and how to make the algorithms distributed. From the system side, one challenge is how to store and share the knowledge with other robots around the world. Another system-side challenge is related to crowdsourcing. Researchers design most

crowdsourcing systems in a one-by-one manner, meaning they design every system from scratch. This design method is inefficient if we have to design multiple related crowdsourcing systems. One interesting direction is to develop new graphical programming models that help robotics researchers design crowdsourcing tasks more efficiently. The third challenge relates to the crowdsourcing workers. Currently, the user-interface for crowdsourcing workers to teach robots is very limited in its expressiveness and quite inefficient in use. These limitations call for a new graphical programming model that helps crowdsourcing user accomplish needed tasks.

BIBLIOGRAPHY

[1] N. Abdo, C. Stachniss, L. Spinello, and W. Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15)*, pages 1557–1564. IEEE, 2015.

[2] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing over complete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11): 4311–4322, 2006.

[3] O. Alonso and R. Baeza-Yates. Design and implementation of relevance assessments using crowdsourcing. In *Advances in Information Retrieval*, pages 153–164. Springer, 2011.

[4] Amazon. Amazon's site directory. http://www.amazon.com/gp/site-directory/ref=nav_sad, 2015.

[5] A. Barbu, A. Bridge, Z. Burchill, D. Coroian, S. Dickinson, S. Fidler, A. Michaux, S. Mussman, S. Narayanaswamy, D. Salvi, et al. Video in sentences out. *arXiv preprint arXiv:1204.2742*, 2012.

[6] E. Bart, I. Porteous, P. Perona, and M. Welling. Unsupervised learning of visual taxonomies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, 2008.

[7] H. Bay, A. Ess, T. Tuytelaars, and G.L. Van. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[8] S. Bender-deMoll and D.A. McFarland. The art and science of dynamic network visualization. *Journal of Social Structure*, 7(2):1–38, 2006.

[9] BestBuy. Bestbuy's site directory. http://www.bestbuy.com/, 2016.

[10] I. Biederman. Recognition-by-components: a theory of human image understanding. *Psychological Review*, 94:115–147, 1987.

[11] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] 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 17 (NIPS'04)*, 16:17, 2004.

[13] S. Bloehdorn, A. Hotho, and S. Staab. An ontology-based framework for text mining. In *GLDV*, 2005.

[14] M. Blum, J.T. Springenberg, J. Wülfing, and M. Riedmiller. A learned feature descriptor for object recognition in RGB-D data. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'12)*, 2012.

[15] L. Bo, X. Ren, and D. Fox. Depth Kernel descriptors for object recognition. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11)*, 2011.

[16] L. Bo, X. Ren, and D. Fox. Hierarchical matching pursuit for image classification: architecture and fast algorithms. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, 2011.

[17] L. Bo, X. Ren, and D. Fox. Unsupervised feature learning for RGB-D based object recognition. In *Proceedings of the 13th International Symposium on Experimental Robotics (ISER'12)*, 2012.

[18] L. Bo, X. Ren, and D. Fox. Multipath sparse coding using hierarchical matching pursuit. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'13), 2013.

[19] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 1247–1250. ACM, 2008.

[20] K.M. Borgwardt, A. Gretton, M.J. Rasch, H. Kriegel, B. Schölkopf, and A.J. Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14): e49–e57, 2006.

[21] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *Proceedings of the 9th IEEE International Conference on Computer Vision (ICCV'07)*, 2007.

[22] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[23] J. Bragg, Mausam, and D. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *Proceedings of the 1st AAAI Conference on Human Computation and Crowdsourcing (HCOMP'13)*, 2013.

[24] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. In *Proceedings of the 8th European Conference on Computer Vision (ECCV'10)*, pages 438–451. Springer, 2010.

[25] C. Brewster, H. Alani, S. Dasmahapatra, and Y. Wilks. Data driven ontology evaluation. In *LREC 2004*, 2004.

[26] M. Bruls, K. Huizing, and J.J. Van Wijk. *Squarified tree maps*. Springer, 2000.

[27] S.S. Bucak, R. Jin, and A.K. Jain. Multi-label learning with incomplete class assignments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'2011)*, pages 2801–2808, 2011.

[28] C. Buchheim, M. Jünger, and S. Leipert. Improving walkers algorithm to run in linear time. In *Graph Drawing*, pages 344–353. Springer, 2002.

[29] A. Budanitsky. Lexical semantic relatedness and its application in natural language processing. Technical report, University of Toronto, 1999.

[30] P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology learning from text: an overview*, pages 3–12. IOS Press, 2005.

[31] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka, and T.M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

[32] P. Chandar and B. Carterette. Using preference judgments for novel document retrieval. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 861–870. ACM, 2012.

[33] C. Chang and C. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[34] X. Chen and L. Zitnick. Learning a recurrent visual representation for image caption generation. *arXiv preprint arXiv:1411.5654*, 2014.

[35] X. Chen, W. Pan, J. Kwok, and J. Carbonell. Accelerated gradient method for multi-task sparse learning problem. In *Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09)*, 2009.

[36] X. Chen, A. Shrivastava, and A. Gupta. Neil: extracting visual knowledge from web data.

In *Proceedings of the 15th IEEE International Conference on Computer Vision (ICCV'13)*, December 2013.

[37] L. Chilton, G. Little, D. Edge, D. Weld, and J. Landay. Cascade: crowdsourcing taxonomy creation. In *Proceedings of the 31st Annual Conference on Human Factors in Information Systems (CHI'13)*, 2013.

[38] Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14, 1965.

[39] M. Chung, M. Forbes, M. Cakmak, and R.P. Rao. Accelerating imitation learning through crowdsourcing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14)*, 2014.

[40] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, et al. Predicting protein structures with a multiplayer online game. *Nature*, 466 (7307):756–760, 2010.

[41] T. Cour, B. Sapp, and B. Taskar. Learning from partial labels. *The Journal of Machine Learning Research*, 12:1501–1536, 2011.

[42] CrowdFlower. http://www.crowdflower.com/, 2016. [Online; accessed 07-April-2016].

[43] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[44] A. Dale and E. Reiter. Computational interpretations of Gricean maxims in the generation of referring expressions. *Cognitive Science*, 18:233–263, 1995.

[45] C. Darwin and W.F. Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt, 2009.

[46] Datawrapper. http://datawrapper.de, 2016. [Online; accessed 07-April-2016].

[47] H. Daume. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL'07)*, 2007.

[48] M.H. DeGroot. *Optimal statistical decisions*, volume 82. John Wiley & Sons, 2005.

[49] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[50] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-fei. ImageNet: a large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009.

[51] J. Deng, A.C. Berg, and L. Fei-Fei. Hierarchical semantic indexing for large scale image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, pages 785–792. IEEE, 2011.

[52] J. Deng, J. Krause, and L. Fei-Fei. Fine-grained crowdsourcing for fine-grained recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'13)*, 2013.

[53] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *Proceedings of the 12th European Conference on Computer Vision (ECCV'14)*, pages 48–64. Springer, 2014.

[54] J. Deng, O. Russakovsky, J. Krause, M. Bernstein, A. Berg, and L. Fei-Fei. Scalable multi-label annotation. In *Proceedings of the 32nd SIGCHI Conference on Human Factors in Computing Systems (CHI'14)*, pages 3099–3102, 2014.

[55] J. Deng, O. Russakovsky, J. Krause, M.S. Bernstein, A. Berg, and L. Fei-Fei. Scalable multi-label annotation. In *Proceedings of the 32nd ACM Conference on Human Factors in Computing Systems (CHI'14)*, 2014.

[56] J. Deng, J. Krause, M. Stark, and L. Fei-Fei. Leveraging the wisdom of the crowd for fine-grained recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38, 2015.

[57] M. Dewey. *Classification and subject index for cataloguing and arranging the books and pamphlets of a library*. eBook, 1876.

[58] J. Donahue, H.L. Anne, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR:2015)*, pages 2625–2634, 2015.

[59] K. Duan, D. Parikh, D. Crandall, and K. Grauman. Discovering localized attributes for fine-grained recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12)*, 2012.

[60] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240, 1967.

[61] D. Elliott and F. Keller. Image description using visual dependency representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'13)*, volume 13, pages 1292–1302, 2013.

[62] K. Erk. Vector space models of word meaning and phrase meaning: a survey. *Language and Linguistics Compass*, 2012.

[63] J. Euzenat, P. Shvaiko, et al. *Ontology matching*, volume 333. Springer, 2007.

[64] J. Evermann and J. Fang. Evaluating ontologies: towards a cognitive measure of quality. *Information Systems*, 35(4):391–403, 2010.

[65] H. Fang, S. Gupta, F. Iandola, R.K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J.C. Platt, et al. From captions to visual concepts and back. In *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR'15), pages 1473–1482, 2015.

[66] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009.

[67] A. Farhadi, M. Hejrati, M.A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: Generating sentences from images. In *Proceedings of the 8th European Conference on Computer Vision (ECCV'10)*, pages 15–29. Springer, 2010.

[68] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 28:594–611, April 2006.

[69] C. Fellbaum. *WordNet: an electronic lexical database*. Bradford Books, 1998.

[70] S. Feng, S. Ravi, R. Kumar, P. Kuznetsova, W. Liu, A. Berg, T. Berg, and Y. Choi. Refer-to-as relations as semantic knowledge. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 2015.

[71] V. Ferrari and A. Zisserman. Learning visual attributes. In *Advances in Neural Information Processing Systems 20 (NIPS'07)*, 2007.

[72] B.B. Fu, N.F. Noy, and M. Storey. Indented tree or graph? a usability study of ontology visualization techniques in the context of class mapping evaluation. In *The Semantic Web (ISWC 2013')*, pages 117–134. Springer, 2013.

[73] R. Fu, B. Qin, and T. Liu. Exploiting multiple sources for open-domain hypernym discovery. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'13)*, pages 1224–1234, 2013.

[74] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by back propagation. *arXiv preprint arXiv:1409.7495*, 2014.

[75] B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12)*, 2012.

[76] B. Gong, K. Grauman, and F. Sha. Reshaping visual datasets for domain adaptation. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*, 2013.

[77] C. Grady and M. Lease. Crowdsourcing document relevance assessment with mechanical turk. In *Proceedings of the NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 172–179. Association for Computational Linguistics, 2010.

[78] M. Graham and J. Kennedy. A survey of multiple tree visualization. *Information Visualization*, 9(4):235–252, 2010.

[79] K. Grauman and B. Leibe. *Visual object recognition*. Morgan & Claypool, 2011.

[80] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, 2008.

[81] T. Griffiths, J. Tenenbaum, and M. Steyvers. Topics in semantic representation. *Psychological Review*, 2007.

[82] T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[83] J. Guerra-Gomez, M.L. Pack, C. Plaisant, and B. Shneiderman. Visualizing change over time using dynamic hierarchies: Treeversity2 and the stemview. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2566–2575, 2013.

[84] A. Gupta and P. Mannem. From image annotation to image description. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*, pages 196–204. Springer, 2012.

[85] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[86] M.A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[87] J. Heer and M. Bostock. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proceedings of the 28st Annual Conference on Human Factors in Information Systems (CHI'10)*, pages 203–212. ACM, 2010.

[88] I. Herman, G. Melançon, and M.S. Marshall. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics*, 6 (1):24–43, 2000.

[89] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[90] A. Hippisley, D. Cheng, and K. Ahmad. The head-modifier principle and multilingual term extraction. *Natural Language Engineering*, 11(2):129–157, June 2005.

[91] M. Hodosh, P. Young, and J. Hockenmaier. Framing image description as a ranking task: data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, pages 853–899, 2013.

[92] J. Hoffman, B. Kulis, T. Darrell, and K. Saenko. Discovering latent domains for multi-source domain adaptation. In *Proceedings of the 10th European Conference on Computer Vision (ECCV'12)*, 2012.

[93] J. Hoffman, E. Tzeng, J. Donahue, Y. Jia, K. Saenko, and T. Darrell. One-shot adaptation of supervised deep convolutional models. *CoRR*, abs/1312.6204, 2013. URL http://arxiv.org/abs/1312.6204.

[94] S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV'11)*, 2011.

[95] W.C. Howell and S.A. Burnett. Uncertainty measurement: a cognitive taxonomy. *Organizational Behavior and Human Performance*, 22(1):45–68, 1978.

[96] X.-S. Hua and J. Li. Prajna: towards recognizing whatever you want from images without image labeling. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, January 2015.

[97] J. Huang, A. Gretton, K.M. Borgwardt, B. Schölkopf, and A.J. Smola. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 601–608, 2006.

[98] A. Jain, D. Das, J.K. Gupta, and A. Saxena. Planit: a crowdsourcing approach for learning to plan paths from large scale preference feedback. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15)*, pages 877–884. IEEE, 2015.

[99] S. Ji, L. Tang, S. Yu, and J. Ye. Extracting shared subspace for multi-label classification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 381–389. ACM, 2008.

[100] Y. Jia, M. Salzmann, and T. Darrell. Learning cross-modality similarity for multinomial data. In *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV'11)*, pages 2407–2414. IEEE, 2011.

[101] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. URL http://arxiv.org/abs/1408.5093.

[102] T. Joachims, T. Finley, and C. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[103] P. Jolicoeur, M.A. Gluck, and S.M. Kosslyn. Pictures and names: making the connection. *Cognitive Psychology*, 16:243–275, 1984.

[104] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 5th International Conference on World Wide Web (WWW'06)*, 2006.

[105] M. Kääriäinen. Active learning in the non-realizable case. In *Proceedings of the 16th Conference on Algorithmic Learning Theory (ALT'06)*, pages 63–77. Springer, 2006.

[106] S. Kazemzadeh, V. Ordonez, M. Matten, and T.L. Berg. Referitgame: referring to objects in photographs of natural scenes. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing and Computational Natural Language Learning (EMNLP'14)*, 2014.

[107] R. Kiros, R. Salakhutdinov, and R. Zemel. Multimodal neural language models. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*, pages 595–603, 2014.

[108] R. Kiros, R. Salakhutdinov, and R.S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*, 2014.

[109] A. Kittur, E.H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*, pages 453–456. ACM, 2008.

[110] D. Klein and C. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Association of Computational Linguistic (ACL'03)*, 2003.

[111] K. Knight. Building a large ontology for machine translation. In *the Workshop on Human Language Technology*, pages 185–190. Association for Computational Linguistics, 1993.

[112] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

[113] T. Koo, A. Globerson, P.X. Carreras, and M. Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*, pages 141–150, 2007.

[114] L. Kotlerman, I. Dagan, I. Szpektor, and M. Zhitomirsky-Geffet. Directional distributional similarity for lexical inference. *Natural Language Engineering*, 16(04):359–389, 2010.

[115] A. Krizhevsky, S. Ilya, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*. Curran Associates, Inc., 2012.

[116] G. Kulkarni, V. Premraj, V. Ordonez, S. Dhar, S. Li, Y. Choi, A.C. Berg, and T. Berg. Babytalk: understanding and generating simple image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2891–2903, 2013.

[117] S. Kullback and R.A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[118] A. Kumar, A. Saha, and H. Daume. Co-regularization based semi-supervised domain adaptation. In *Advances in Neural Information Processing Systems 23 (NIPS'10)*, pages 478–486, 2010.

[119] N. Kumar, A. Berg, P. Belhumeur, and S. Nayar. Attribute and simile classifiers for face verification. In *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV'09)*, 2009.

[120] L.I. Kuncheva and C.J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, May 2003.

[121] K. Lai, L. Bo, X. Ren, and D. Fox. A scalable tree-based approach for joint object and pose recognition. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*, 2011.

[122] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'11)*, 2011.

[123] C. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009.

[124] C. Leacock and M. Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An Electronic Lexical Database*, 49(2):265–283, 1998.

[125] B. Lee, C.S. Parr, D. Campbell, and B.B. Bederson. How users interact with biodiversity information using taxontree. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 320–327. ACM, 2004.

[126] B. Lee, G.G. Robertson, M. Czerwinski, and C.S. Parr. Candidtree: visualizing structural uncertainty in similar hierarchies. *Information Visualization*, 6(3):233–246, 2007.

[127] M. Lee, J. Forlizzi, S. Kiesler, P. Rybski, J. Antanitis, and S. Savetsila. Personalization in hri: a longitudinal field experiment. In *Proceedings of the 7th ACM/IEEE International Conference on Human Robot Interaction (HRI'12)*, pages 319–326. IEEE, 2012.

[128] A. Lenci and G. Benotto. Identifying hypernyms in distributional semantic spaces. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 75–79. Association for Computational Linguistics, 2012.

[129] D. Leyzberg, S. Spaulding, and B. Scassellati. Personalizing robot tutors to individuals' learning differences. In *Proceedings of the 9th ACM/IEEE International Conference on Human Robot Interaction (HRI'14)*, pages 423–430. ACM, 2014.

[130] X. Ling and D.S. Weild. Fine-grained entity resolution. In *Proceedings of the 26 AAAI Conference on Artificial Intelligence (AAAI'12)*, 2012.

[131] C.J. Lintott, K. Schawinski, A. Slosar, K. Land, S. Bamford, D. Thomas, M.J. Raddick, R.C. Nichol, A. Szalay, D. Andreescu, et al. Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 389(3):1179–1189, 2008.

[132] J. Liu, S. Ji, and J. Ye. *SLEP: sparse learning with efficient projections*. Arizona State University, 2009. URL http://www.public.asu.edu/~jye02/Software/SLEP.

[133] D. Lowe. Distinctive image features from scale-invariant key points. *International Journal of Computer Vision*, 60:91–110, 2004.

[134] W. Lowe. Towards a theory of semantic space. In *the Annual Conference of the Cognitive Science Society (CogSci'01)*, 2001.

[135] Y. Lu, W. Zhang, K. Zhang, and X. Xue. Semantic context learning with large-scale weakly-labeled image set. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, pages 1859–1863. ACM, 2012.

[136] T. Luong, R. Socher, and C.D. Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113. Citeseer, 2013.

[137] Mingsheng M. Long and J. Wang. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

[138] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 251–263. Springer, 2002.

[139] J. Mao, W. Xu, Y. Yang, J. Wang, and A.L. Yuille. Explain images with multimodal recurrent neural networks. *arXiv preprint arXiv:1410.1090*, 2014.

[140] M. Marszałek and C. Schmid. Semantic hierarchies for visual object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)*, pages 1–7. IEEE, 2007.

[141] M.H. Maryam and A. Popescu-Belis. Using crowdsourcing to compare document recommendation strategies for conversations. In *RecSys, Recommendation Utility Evaluation (RUE 2012)*, 2012.

[142] W. Mason and S. Suri. Conducting behavioral research on amazon's mechanical turk. *Behavior Research Methods*, 44(1):1–23, 2012.

[143] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox. A joint model of language and perception for grounded attribute learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, 2012.

[144] B. McFee and G. Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, 2010.

[145] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[146] L. Micallef, P. Dragicevic, and J.D. Fekete. Assessing the effect of visualizations on bayesian reasoning through crowdsourcing. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2536–2545, 2012.

[147] T. Minka and J. Lafferty. Expectation-propagation for the generative aspect model. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 352–359. Morgan Kaufmann Publishers Inc., 2002.

[148] 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 *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 2015.

[149] J.M. Mortensen, M.A. Musen, and N.F. Noy. Developing crowdsourced ontology engineering tasks: an iterative process. In *Proceedings of the 1st International Conference on Crowdsourcing the Semantic Web-Volume 1030 (CEUR-WS'13)*, pages 79–88. CEUR-WS. org, 2013.

[150] G. Obozinski, B. Taskar, and M. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, 2010.

[151] V. Ordonez, G. Kulkarni, and T.L. Berg. Im2text: describing images using 1 million captioned photographs. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, pages 1143–1151, 2011.

[152] V. Ordonez, J. Deng, Y. Choi, A. Berg, and T. Berg. From large scale image categorization to entry-level categories. In *Proceedings of the 15th IEEE International Conference on Computer Vision (ICCV'13)*, pages 2768–2775, 2013.

[153] D. Parikh and K. Grauman. Relative attributes. In *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV'11)*, 2011.

[154] D. Parikh and C.L. Zitnick. Finding the weakest link in person detectors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, pages 1425–1432. IEEE, 2011.

[155] M. Pasca, D. Lin, J. Bigham A. Lifchits, and A. Jain. Names and similarities on the web: fact extraction in the fast lane. In *Proceedings of the 44th Association of Computational Linguistic (ACL'06)*, 2006.

[156] P. Pasupat and P. Liang. Zero-shot entity extraction from web pages. In *Proceedings of the 52nd Association of Computational Linguistic (ACL'14)*, pages 391–401, 2014.

[157] J. Pennington, R. Socher, and C.D. Manning. Glove: global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[158] R. Porzel and R. Malaka. A task-based approach for ontology evaluation. In *ECAI Workshop on Ontology Learning and Population*, 2004.

[159] U. Priss. Facet-like structures in computer science. *Axiomathes*, 18(2):243–255, 2008.

[160] Tableau Public. http://public.tableau.com/s/, 2016. [Online; accessed 07-April-2016].

[161] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

[162] K. Ramirez-Amaro, M. Beetz, and G. Cheng. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, pages –, 2015.

[163] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the 49th Association of Computational Linguistic (ACL'11)*, 2011.

[164] P. Resnik. Semantic similarity in a taxonomy: an information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence*, 11: 95–130, 1999.

[165] M. Rohrbach, M. Stark, and B. Schiele. Evaluating knowledge transfer and zero-shot learning in a large-scale setting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, pages 1641–1648, 2011.

[166] E. Rosch. Principles of categorization. *Etnolingwistyka*, 17:11–35, 2005.

[167] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal on Computer Vision*, 77(1-3): 157–173, 2008.

[168] A. Saxena, A. Jain, O. Sener, A. Jami, D.K. Misra, and H.S. Koppula. Robobrain: large-scale knowledge engine for robots. *CoRR*, abs/1412.0691, 2014. URL http://arxiv.org/abs/1412.0691.

[169] M. Schwarz, H. Schulz, and S. Behnke. The object recognition and pose estimation based on pre-trained convolutional neural network features. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15)*, 2015.

[170] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[171] M. Sifer. Filter co-ordinations for exploring multi-dimensional data. *Journal of Visual Languages & Computing*, 17(2):107–125, 2006.

[172] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[173] J. Sivic, B. Russell, A. Zisserman, I. Ecole, and N. Suprieure. Unsupervised discovery of visual object class hierarchies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, 2008.

[174] R. Snow, D. Jurafsky, and A.Y. Ng. Learning syntactic patterns for automatic hypernym and discovery. *Advances in Neural Information Processing Systems 17 (NIPS'04)*, 2004.

[175] R. Socher, B. Huval, C. Manning, and A. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'12)*, 2012.

[176] R. Socher, A. Karpathy, Q.V. Le, C.D. Manning, and N.Y. Andrew. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics (TACL'14)*, 2:207–218, 2014.

[177] A. Sorokin, D. Berenson, S. Srinivasa, and M. Hebert. People helping robots helping people: crowdsourcing for grasping novel objects. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10)*, October 2010.

[178] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies*, 53(5):663–694, 2000.

[179] M. Steel and S. Bocker. Simple but fundamental limitations on super tree and consensus tree methods. *Systematic Biology*, 49(2):363–368, 2000.

[180] F.M. Suchanek, G. Kasneci, and G. Weikum. Yago: a large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[181] C. Sun, N.Rampalli, F. Yang, and A. Doan. Chimera: large-scale classification using machine learning, rules, and crowdsourcing. *The VLDB Endowment*, 7(13):1529–1540, August 2014.

[182] Y. Sun and D. Fox. Neol: toward never-ending object learning for robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'16)*, 2016.

[183] Y. Sun, L. Bo, and D. Fox. Attribute based object identification. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'13)*, 2013.

[184] Y. Sun, L. Bo, and D. Fox. Learning to identify new objects. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14)*, 2014.

[185] Y. Sun, A. Singla, D. Fox, and A. Krause. Building hierarchies of concepts via crowdsourcing. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 2015.

[186] Y.-Y. Sun, Y. Zhang, and Z.-H. Zhou. Multi-label learning with weak label. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

[187] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: a large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, 2005.

[188] M. Tenorth and M. Beetz. Representations for robot knowledge in the knowrob framework. *Artificial Intelligence*, 2015.

[189] S. Thrun and T.M. Mitchell. Lifelong robot learning. Technical report, Robotics and Autonomous Systems, 1993.

[190] T.B. Todd and P. Neville. A comparison of 2-d visualizations of hierarchies. In *Infovis*, page 131. IEEE, 2001.

[191] S. Tong and D. Koller. Active learning for structure in bayesian networks. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 863–869, 2001.

[192] A. Torralba and A.A. Efros. Unbiased look at dataset bias. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, June 2011.

[193] K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, 2000.

[194] Amazon Mechanical Turk. http://www.mturk.com/mturk/, 2016. [Online; accessed 07-April-2016].

[195] P. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning (ECML'01)*, pages 491–502, 2001.

[196] P. Turney and P. Pantel. From frequency to meaning: vector space models of semantics. *Journal of Artificial Intelligence Research*, 2010.

[197] W. Tutte. *Graph Theory*. Addison-Wesley, 1984.

[198] A. Vailaya, M.A. Figueiredo, A.K. Jain, and H.J. Zhang. Image classification for content-based indexing. *IEEE Transactions on Image Processing*, 10(1):117–130, 2001.

[199] I. Vendrov, R. Kiros, S. Fidler, and R. Urtasun. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*, 2015.

[200] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: a neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, pages 3156–3164, 2015.

[201] C. Vondrick, D. Patterson, and D. Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1):184–204, 2013.

[202] E. Voorhees. Using wordnet to disambiguate word senses for text retrieval. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993.

[203] C. Wah, S. Branson, P. Perona, and S. Belongie. Multiclass recognition and part localization with humans in the loop. In *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV'11)*, pages 2524–2531. IEEE, 2011.

[204] C. Wang, S. Yan, L. Zhang, and H. Zhang. Multi-label sparse coding for automatic image annotation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, pages 1643–1650, 2009.

[205] J. Wang, K. Markert, and M. Everingham. Learning models for object recognition from natural language descriptions. In *the 20th British Machine Vision Conference (BMVC'09)*, 2009.

[206] WikiHow. Wikihow. http://www.wikihow.com, 2016.

[207] D.B. Wilson. Generating random spanning trees more quickly than the cover time. In *the 28th Annual ACM Symposium on Theory of Computing*, pages 296–303, 1996.

[208] W. Wong, W. Liu, and M. Bennamoun. Ontology learning from text: a look back and into the future. *ACM Computing Surveys*, 44(4):20:1–20:36, 2012.

[209] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *Proceedings of Robotics: Science and Systems (RSS'14)*, 2014.

[210] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics (ACL'94)*, pages 133–138. Association for Computational Linguistics, 1994.

[211] F. Xu and J. Tenenbaum. Word learning as bayesian inference. In *Proceedings of Annual Conference of the Cognitive Science Society (CogSci'00)*, 2000.

[212] X. Xue, W. Zhang, J. Zhang, B. Wu, J. Fan, and Y. Lu. Correlative multi-label multi-instance image annotation. In *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV'11)*, pages 651–658. IEEE, 2011.

[213] Y. Yang, C. Teo, H. Daumé III, and Y. Aloimonos. Corpus-guided sentence generation of natural images. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'11)*, pages 444–454. Association for Computational Linguistics, 2011.

[214] A. Yates and O. Etzioni. Unsupervised methods for determining object and relation synonyms on the web. *Journal of Artificial Intelligence Research*, 34(1):255, 2009.

[215] K. Yu, Y. Lin, and J. Lafferty. Learning image representations from the pixel level via hierarchical sparse coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, 2011.

[216] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the 12th European Conference on Computer Vision (ECCV'14)*, pages 818–833. Springer, 2014.

[217] M.-L. Zhang and Z.-H. Zhou. Ml-knn: a lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[218] W. Zhang, Y. Lu, X. Xue, and J. Fan. Automatic image annotation with weakly labeled dataset. In *Proceedings of the 19th ACM International Conference on Multimedia (MM'11)*, pages 1185–1188. ACM, 2011.

[219] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, April 2012.

[220] C.L. Zitnick and D. Parikh. The role of image understanding in contour detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12)*, pages 622–629. IEEE, 2012.

[221] A. Zweig and D. Weinshall. Exploiting object hierarchy: combining models from different category levels. In *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV'07)*, pages 1–8. IEEE, 2007.

Appendix A

PROOFS

A.1 Derivation of the Objective Function

$$L^{\beta}_{\tilde{\pi}}(\Lambda') - L^{\beta}_{\tilde{\pi}}(\Lambda) \tag{A.1}$$

$$= \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \log P(T|\Lambda) - \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \log P(T|\Lambda') + \sum_{i,j} \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.2}$$

$$= \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \log \frac{\exp(\sum_{e_{i,j}\in T} \lambda_{i,j})}{\exp(\sum_{e_{i,j}\in T} \lambda_{i,j} + \delta_{i,j})} + \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \log \frac{Z(\Lambda')}{Z(\Lambda)} + \sum_{i,j} \beta(|\lambda'_{i,j} - |\lambda_{i,j}|) \tag{A.3}$$

$$= \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \sum_{e_{i,j}\in T} -\delta_{i,j} + \log \frac{Z(\Lambda')}{Z(\Lambda')} + \sum_{i,j} \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.4}$$

$$= \sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}) + \log \frac{\sum_{T'\in\mathcal{T}} \exp(\sum_{e_{i,j}\in T'} \lambda_{i,j} + \delta_{i,j})}{\sum_{T\in\mathcal{T}} \exp(\sum_{e_{i,j}\in T} \lambda_{i,j})} + \sum_{i,j} \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.5}$$

$$= \sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}) + \log \sum_{T\in\mathcal{T}} P(T|\Lambda) \exp(\sum_{e_{i,j}\in T} \delta_{i,j}) + \sum_{i,j} \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.6}$$

$$= \sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}) + \log \sum_{T\in\mathcal{T}} P(T|\Lambda) \exp(\sum_{e_{i,j}\in T} \frac{1}{N} N\delta_{i,j}) + \sum_{i,j} \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.7}$$

$$\leq \sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}) + \frac{1}{N} P(e_{i,j})(e^{N\delta_{i,j}} - 1) + \beta(|\lambda'_{i,j}| - |\lambda_{i,j}|) \tag{A.8}$$

(A.2) $\rightarrow$ (A.3) uses the definition of $P(T|\Lambda)$.

(A.3) $\rightarrow$ (A.4) uses the fact that

$$\sum_{T\in\mathcal{T}} \tilde{\pi}(T) \sum_{e_{i,j}\in T} -\delta_{i,j} = \sum_{i,j} -\delta_{i,j} \sum_{T\in\mathcal{T}: e_{i,j}\in T} \tilde{\pi}(T) = \sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}),$$

where $\tilde{P}(e_{i,j})$ is the marginal likelihood of the edge $e_{i,j}$.

To get (A.7) $\rightarrow$ (A.8), we use an inequality that, if $x_j \in \mathbb{R}$ and $p_j \geq 0$ with $\sum_j p_j \leq 1$, then

$$\exp(\sum_j p_j x_j) - 1 \leq \sum_j p_j (e^{x_j} - 1).$$

Such that

$$\exp(\sum_{e_{i,j} \in T} \frac{1}{N} N\delta_{i,j}) \leq 1 + \sum_{e_{i,j} \in T} \frac{1}{N}(e^{N\delta_{i,j}} - 1).$$

The it follows

$$\log \sum_{T \in \mathcal{T}} P(T|\Lambda) \exp(\sum_{e_{i,j} \in T} \frac{1}{N} N\delta_{i,j}) \tag{A.9}$$

$$\leq \log(1 + \sum_{T \in \mathcal{T}} P(T|\Lambda) \sum_{e_{i,j} \in T} \frac{1}{N}(e^{N\delta_{i,j}} - 1)) \tag{A.10}$$

$$= \log(1 + \frac{1}{N} \sum_{i,j} P(e_{i,j})(e^{N\delta_{i,j}} - 1)) \tag{A.11}$$

$$\leq \frac{1}{N} \sum_{i,j} P(e_{i,j})(e^{N\delta_{i,j}} - 1) \tag{A.12}$$

(A.11) $\rightarrow$ (A.12) is true because $\log(1 + x) \leq x, \forall x \geq -1$, and

$$\frac{1}{N} \sum_{i,j} P(e_{i,j})(e^{N\delta_{i,j}} - 1) \geq \frac{1}{N} \sum_{i,j} -P(e_{i,j}) = -1.$$

A.2 Minimization of (A.8)

Case 1: $\delta_{i,j} = -\lambda_{i,j}$. Such that (A.8) becomes

$$\sum_{i,j} \lambda_{i,j} \tilde{P}(e_{i,j}) + \frac{1}{N} P(e_{i,j})(e^{-N\lambda_{i,j}} - 1) \tag{A.13}$$

Case 2: $\lambda_{i,j} + \delta_{i,j} \geq 0$. Such that (A.8) becomes

$$\sum_{i,j} -\delta_{i,j} \tilde{P}(e_{i,j}) + \frac{1}{N} P(e_{i,j})(e^{N\delta_{i,j}} - 1) + \beta\delta_{i,j} \tag{A.14}$$

Take derivative of (A.14), and set it to be 0:

$$-\tilde{P}(e_{i,j}) + P(e_{i,j})e^{N\delta_{i,j}} + \beta = 0,$$

the solution is

$$\frac{1}{N}\log\frac{\tilde{P}(e_{i,j}) - \beta}{P(e_{i,j})}.$$

Case 3: $\lambda_{i,j} + \delta_{i,j} < 0$. Such that (A.8) becomes

$$\sum_{i,j} -\delta_{i,j}\tilde{P}(e_{i,j}) + \frac{1}{N}P(e_{i,j})(e^{N\delta_{i,j}} - 1) - \beta\delta_{i,j} \tag{A.15}$$

Take derivative of (A.15), and set it to be 0:

$$-\tilde{P}(e_{i,j}) + P(e_{i,j})e^{N\delta_{i,j}} - \beta = 0,$$

the solution is

$$\frac{1}{N}\log\frac{\tilde{P}(e_{i,j}) + \beta}{P(e_{i,j})}.$$

A.3 Proof of Theorem 1

Theorem 3. *Assume β is strictly positive. Then Algorithm 1 produces a sequences $\Lambda^{(1)}, \Lambda^{(2)}, \ldots$ such that*

$$\lim_{\ell\to\infty} L^{\beta}_{\tilde{\pi}}(\Lambda^{(\ell)}) = \min_{\Lambda} L^{\beta}_{\tilde{\pi}}(\Lambda).$$

Proof. First let us define Λ^+ and Λ^- in terms of Λ as follows: for each (i,j), if $\lambda_{i,j} \geq 0$, then $\lambda^+_{i,j} = \lambda_{i,j}$ and $\lambda^-_{i,j} = 0$, and if $\lambda_{i,j} \leq 0$, then $\lambda^+_{i,j} = 0$ and $\lambda^-_{i,j} = -\lambda_{i,j}$. Λ'^+, Λ'^-, $\Lambda^{(\ell)+}$, $\Lambda^{(\ell)-}$, etc. are defined analogously.

Let $F_{i,j}$ denote the (i,j) component in (A.8), For any Λ and Δ, we have the following:

$$|\lambda+\delta|-|\lambda| = \min\{\delta^+ + \delta^- | \delta^+ \geq -\lambda^+, \delta^- \geq -\lambda^-, \delta^+ - \delta^- = \delta\} \tag{A.16}$$

Plugging into the definition of $F_{i,j}$ gives:

$$\begin{aligned} F_{i,j}(\Lambda,\Delta) =& -\delta_{i,j}\tilde{P}(e_{i,j}) + \frac{1}{N}P(e_{i,j})(e^{N\delta_{i,j}}-1) + \beta(|\lambda_{i,j}+\delta_{i,j}|-|\lambda_{i,j}|) && \text{(A.17)} \\ =& \min\{G_{i,j}(\Lambda,\Delta^+,\Delta^-) | \delta^+_{i,j} \geq -\lambda^+_{i,j}, \delta^-_{i,j} \geq \lambda^-_{i,j}, \delta^+_{i,j}-\delta^-_{i,j} = \delta_{i,j}\}, && \text{(A.18)} \end{aligned}$$

where

$$G_{i,j}(\Lambda,\Delta^+,\Delta^-) = (\delta^-_{i,j}-\delta^+_{i,j})\tilde{P}(e_{i,j}) + \frac{1}{N}P(e_{i,j})(e^{N(\delta^+_{i,j}-\delta^-_{i,j})}-1) + \beta(\delta^+_{i,j}+\delta^-_{i,j}) \tag{A.19}$$

So, by (A.8),

$$\begin{aligned} L^\beta_{\tilde{\pi}}(\Lambda^{(\ell+1)}) - L^\beta_{\tilde{\pi}}(\Lambda^{(\ell)}) \leq& \sum_{i,j} F_{i,j}(\Lambda^\ell,\Delta) && \text{(A.20)} \\ =& \sum_{i,j} \min_{\delta_{i,j}} F_{i,j}(\Lambda^\ell,\Delta) && \text{(A.21)} \\ =& \sum_{i,j} \min\{G_{i,j}(\Lambda^\ell,\Delta^+,\Delta^-) | \delta^+_{i,j} \geq -\lambda^+_{i,j}, \delta^-_{i,j} \geq \lambda^-_{i,j}, \delta^+_{i,j}-\delta^-_{i,j} = \delta_{i,j}\}\} && \text{(A.22)} \end{aligned}$$

Note that $G_{i,j}(\Lambda, \mathbf{0}, \mathbf{0}) = 0$, so none of the terms in this sum can be positive. So the Λ^ℓ's have a convergent subsequence converging to some $\hat{\Lambda}$ such that

$$\sum_{i,j} \min\{G_{i,j}(\Lambda^\ell,\Delta^+,\Delta^-) | \delta^+_{i,j} \geq -\lambda^+_{i,j}, \delta^-_{i,j} \geq \lambda^-_{i,j}, \delta^+_{i,j}-\delta^-_{i,j} = \delta_{i,j}\}\} = 0. \tag{A.23}$$

It is easy to verify that minimizing $L^\beta_{\tilde{\pi}}(\Lambda)$ is the dual problem of the following convex program:

$$\max_{p_1,\ldots,p_N \in \mathbb{R}^{+N}} \sum_{i=1}^{N} H(p_i) \tag{A.24}$$

$$s.t. \sum_{i=0}^{N} p(e_{i,j}) = 1, \forall j \tag{A.25}$$

$$\tilde{P}(e_{i,j}) - P(e_{i,j}) \leq \beta, \forall (i,j) \tag{A.26}$$

$$P(e_{i,j}) - \tilde{P}(e_{i,j}) \leq \beta, \forall (i,j). \tag{A.27}$$

We will show that $\hat{\Lambda}^+$ and $\hat{\Lambda}^-$ together with $P(T|\hat{\Lambda})$ satisfy the KKT condition of the previous convex program, and thus form a solution to the prime problem as well as to the dual, the minimization of $L^{\beta}_{\tilde{\pi}}$. For $P(T|\hat{\Lambda})$, these conditions work out to be the following for all (i,j):

$$\hat{\lambda}^+_{i,j} \geq 0, \tilde{P}(e_{i,j}) - P(e_{i,j}) \leq \beta, \hat{\lambda}^+_{i,j}(\tilde{P}(e_{i,j}) - P(e_{i,j}) - \beta) = 0 \tag{A.28}$$

$$\hat{\lambda}^-_{i,j} \geq 0, P(e_{i,j}) - \tilde{P}(e_{i,j}) \leq \beta, \hat{\lambda}^+_{i,j}(P(e_{i,j}) - \tilde{P}(e_{i,j}) - \beta) = 0 \tag{A.29}$$

Since $G_{i,j}(\hat{\Lambda}, \mathbf{0}, \mathbf{0}) = 0$, by (A.23), if $\hat{\lambda}_{i,j} > 0$ then $G_{i,j}(\Lambda, \Delta^+, \mathbf{0})$ is nonnegative in a neighborhood of $\delta^+_{i,j} = 0$, and so has a local minimum at this point. Such that

$$\frac{\partial G_{i,j}(\Lambda, \Delta^+, \mathbf{0})}{\partial \delta^+_{i,j}}|_{\delta^+_{i,j}=0} = -\tilde{P}(e_{i,j}) + P(e_{i,j}) + \beta = 0. \tag{A.30}$$

If $\hat{\lambda}^+_{i,j} = 0$, then (A.23) gives that $G_{i,j}(\hat{\Lambda}, \mathbf{0}, \mathbf{0}) = 0$ for $\delta^+_{i,j} \geq 0$. Thus $\partial G_{i,j}(\Lambda, \Delta^+, \mathbf{0})$ cannot be decreasing at $\delta^+_{i,j} = 0$. Therefore, the partial derivative above must be nonnegative. Altogether, these prove (A.28). (A.29) can be proved analogously.

As a whole, we proved that

$$\lim_{\ell \to \infty} L^{\beta}_{\tilde{\pi}}(\Lambda^{(\ell)}) = L^{\beta}_{\tilde{\pi}}(\hat{\Lambda}) = \min_{\Lambda} L^{\beta}_{\tilde{\pi}}(\Lambda).$$

□

A.4 Proof of Theorem 2

Lemma 1. *Suppose samples $\tilde{\pi}$ are obtained from any tree distribution π. Then*

$$|L_{\tilde{\pi}}(\Lambda) - L_{\pi}(\Lambda)| \leq \sum_{i=0}^{N}\sum_{j=1}^{N} |\lambda_{i,j}||\tilde{P}(e_{i,j}) - P(e_{i,j})|,$$

where $P(e_{i,j}) = \sum_{T\in\mathcal{T}:e_{i,j}\in T} \pi(T)$ and $\tilde{P}(e_{i,j}) = \sum_{T\in\mathcal{T}:e_{i,j}\in T} \tilde{\pi}(T)$.

Proof.

$$L_{\tilde{\pi}}(\Lambda) = \sum_{T\in\mathcal{T}} \tilde{\pi}(T) \log \frac{\exp \sum_{e_{i,j}\in T} \lambda_{i,j}}{Z(\Lambda)} \tag{A.31}$$

Such that

$$|L_{\tilde{\pi}}(\Lambda) - L_{\pi}(\Lambda)| = |\sum_{i,j} \lambda_{i,j}(\tilde{P}(e_{i,j}) - P(e_{i,j}))| \leq \sum_{i=0}^{N}\sum_{j=1}^{N} |\lambda_{i,j}||\tilde{P}(e_{i,j}) - P(e_{i,j})|. \tag{A.32}$$

□

Lemma 2. *Suppose samples $\tilde{\pi}$ are obtained from any tree distribution π. Assume that $|P(e_{i,j}) - \tilde{P}(e_{i,j})| \leq \beta_{i,j}, \forall(i,j)$, Let $\hat{\Lambda}$ minimize the regularized log loss $L^{\beta}_{\tilde{\pi}}(\Lambda)$. The for every Λ it holds that*

$$L_{\pi}(\hat{\Lambda}) \leq L_{\pi}(\Lambda) + 2\sum_{i=0}^{N}\sum_{j=1}^{N} \beta|\lambda_{i,j}|.$$

Proof.

$$L_{\pi}(\hat{\Lambda}) \leq L_{\tilde{\pi}}(\hat{\Lambda}) + \sum_{i,j} \beta|\hat{\lambda}_{i,j}| = L^{\beta}_{\tilde{\pi}}(\hat{\Lambda}) \tag{A.33}$$

$$\leq L^{\beta}_{\tilde{\pi}}(\Lambda) = L_{\tilde{\pi}}(\Lambda) + \sum_{i,j} \beta|\lambda_{i,j}| \tag{A.34}$$

$$\leq L_{\pi}(\Lambda) + 2\sum_{i,j} \beta|\lambda_{i,j}|. \tag{A.35}$$

□

(A.33) to (A.34) is tree because of the optimality of $\hat{\Lambda}$. (A.34) to (A.35) follow from Lemma 1.

Theorem 4. *Suppose m samples $\tilde{\pi}$ are obtained from any tree distribution π. Let $\hat{\Lambda}$ minimize the regularized log loss $L_{\tilde{\pi}}^{\beta}(\Lambda)$ with $\beta = \sqrt{\log(N/\delta)/m}$. Then for every Λ it holds with probability at least $1-\delta$ that*

$$L_{\pi}(\hat{\Lambda}) \leq L_{\pi}(\Lambda) + 2\|\Lambda\|_1 \sqrt{\log(N/\delta)/m}$$

Proof. By Hoeffding's inequality, for a fixed pair of (i,j), the probability that $P(e_{i,j}) - \tilde{P}(e_{i,j})$ exceeds β is at most $e^{-2\beta^2 m} = \frac{\delta}{N^2}$. By the union bound, the probability of this happening for any pair of (i,j) is at most δ. Then the theorem follows from Lemma 2. □

A.5 Proof of proposition 1

To simplify the proof, we will parametrize the distribution using $\lambda_{i,j}$, such that

$$w_{i,j} = \begin{cases} \exp\{\lambda_{i,j}\}, \text{ if } i \neq j \\ 0, \text{ if } i = j \end{cases}$$

Lemma 3. *The partition function $Z(\Lambda) = |\hat{L}(\Lambda)|$, where $\hat{L}(\Lambda) = L(\Lambda) + diag(\log(\Lambda_{0,\cdot}))$*

Lemma 4. *The marginal likelihood $P(e_{i,j})$ of an edge $e_{i,j}$ can be computed as follows:*

$$P(e_{i,j}(\Lambda)) = \frac{\partial \log\left(|\hat{L}(\Lambda)|\right)}{\partial \lambda_{i,j}}$$

The proofs of these propositions can be found in [113].

Proposition 2. *Given that $e_{i,j}$ and $e_{i,k}$ are two edges, the marginal likelihood $P(e_{i,j}, e_{i,k})$ is given as follows*

$$P(e_{i,j}, e_{i,k}) = P(e_{i,j})P(e_{i,k}) -$$
$$W_{i,j}W_{i,k} \begin{cases} [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{j,l}, \text{ if } i = 0 \\ -\left([\hat{L}^{-1}]_{i,i}\right)^2 + [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{k,i} - [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{j,i} + [\hat{L}^{-1}]_{i,i}[\hat{L}^{-1}]_{k,j}, \text{ o.w.} \end{cases}$$

where $P(e_{i,j})$ is the marginal likelihood of edge $e_{i,j}$.

Proof. By the definition of the marginal likelihood, we have the following

$$P(e_{i,j}, e_{i,k}) = \frac{1}{Z} \sum_{T: e_{i,j}, e_{i,k} \in T} \prod_{e_{h,m} \in T} \exp\{\lambda_{h,m}\}$$

Using the results of Proposition 4, we can derive the following:

$$\frac{\partial \log Z(\Lambda)}{\partial \lambda_{i,j}} = \frac{1}{Z(\Lambda)} \sum_{T: e_{i,j} \in T} \prod_{e_{h,m} \in T} \exp(\lambda_{h,m}) \tag{A.36}$$

$$\frac{\partial \log Z(\Lambda)}{\partial \lambda_{i,j}} = \sum_{h,m} \frac{\partial \log |\hat{h}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)} \frac{\partial \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j}} \tag{A.37}$$

$$\log \frac{\partial \log Z(\Lambda)}{\partial \lambda_{i,j}} = \log \sum_{T: e_{i,j} \in T} \prod_{e_{h,m} \in T} \exp(\lambda_{h,m}) - \log Z(\Lambda) \tag{A.38}$$

$$\frac{\partial}{\partial \lambda_{i,k}} \left(\log \frac{\partial \log Z(\Lambda)}{\partial \lambda_{i,j}} \right) = \frac{\sum_{T: e_{i,j}, e_{i,k} \in T} \prod_{e_{h,m} \in T} \exp(\lambda_{h,m})}{\sum_{T: e_{i,j} \in T} \prod_{e_{h,m} \in T} \exp(\lambda_{h,m})} - P(e_{i,k}) \tag{A.39}$$

$$= \frac{P(e_{i,j}, e_{i,k})}{P(e_{i,j})} - P(e_{i,k}) \tag{A.40}$$

$$\frac{\partial}{\partial \lambda_{i,k}} \left(\log \frac{\partial \log Z(\Lambda)}{\partial \lambda_{i,j}} \right) = \frac{\sum_{h,m} \left[\frac{\partial \log |\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)} \frac{\partial^2 \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j} \partial \lambda_{i,k}} + \frac{\hat{L}(h,m)(\Lambda)}{\partial \lambda_{i,j}} \frac{\partial^2 \log |\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda) \partial \lambda_{i,k}} \right]}{P(e_{i,j})} \tag{A.41}$$

Compare right sides of both (A.40) and (A.41), we can see that

$$P(e_{i,j}, e_{i,k}) = \sum_{h,m} \frac{\partial \log |\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)} \frac{\partial^2 \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j} \partial \lambda_{i,k}} + \sum_{h,m} \frac{\partial \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j}} \frac{\partial^2 \log |\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda) \partial \lambda_{i,k}} + P(e_{i,j}) P(e_{i,k}) \tag{A.42}$$

$$\sum_{h,m} \frac{\partial \log |\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)} \frac{\partial^2 \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j} \partial \lambda_{i,k}} = 0 \tag{A.43}$$

$$\sum_{h,m} \frac{\partial \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j}} \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)\partial \lambda_{i,k}} = \sum_{h,m} \frac{\partial \hat{L}_{h,m}(\Lambda)}{\partial \lambda_{i,j}} \sum_{h',m'} \frac{\partial \hat{L}_{h',m'}(\Lambda)}{\partial \lambda_{i,k}} \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{h,m}(\Lambda)\partial \hat{L}_{h',m'}(\Lambda)} \tag{A.44}$$

If $i = 0$, it becomes

$$\exp\{\lambda_{0,j}\} \exp\{\lambda_{0,k}\} \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{0,j}(\Lambda)\partial \hat{L}_{0,k}(\Lambda)}, \tag{A.45}$$

otherwise,

$$\exp\{\lambda_{i,j}\} \exp\{\lambda_{i,k}\} \left(\frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{i,i}^2(\Lambda)} - \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{i,i}(\Lambda)\partial \hat{L}_{i,k}(\Lambda)} + \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{i,j}(\Lambda)\partial \hat{L}_{i,i}(\Lambda)} - \frac{\partial^2 \log|\hat{L}(\Lambda)|}{\partial \hat{L}_{i,j}\partial \hat{L}_{i,k}(\Lambda)} \right). \tag{A.46}$$

$$\frac{\partial \log|\hat{L}|}{\partial \hat{L}} = (\hat{L}^T)^{-1} \tag{A.47}$$

$$\frac{\partial^2 \log|\hat{L}|}{\partial \hat{L}_{k,l}\partial \hat{L}_{i,j}} = -((\hat{L}^T)^{-1})_{k,i}((\hat{L}^T)^{-1})_{j,l} \tag{A.48}$$

□

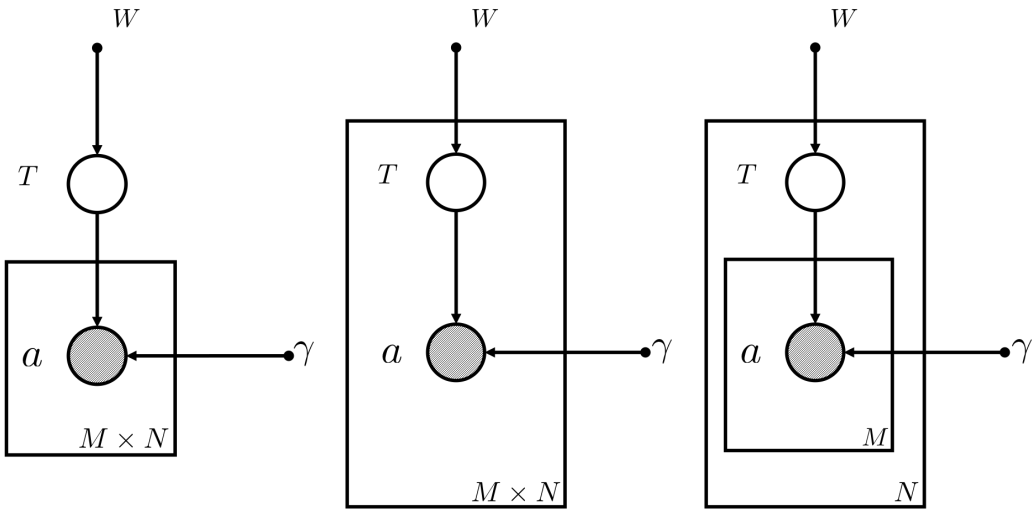
Appendix B

LEARNING USER-SPECIFIC HIERARCHIES

B.1 Background

In Chapter 4, we proposed a crowdsourcing method to estimate a distribution over hierarchies. The Bayesian model used by that method can be represented by the plate notation in Figure B.1 (a). The model assumes that there is one best hierarchy sampled according to a distribution $p(T|W)$ parameterized by weight matrix W. All the answers are generated based on the hierarchy and a single parameter γ denoting a single noise rate. γ is usually referred to as the channel noise, which includes all sought of noises, *e.g.*, uncertainty over hierarchies, worker noise and others. This model is referred to as Noisy-Realizable setting in Active Learning. Some work [105] has proved that it is possible to recover the best hierarchy with high probability if the overall noise rate β is bounded. The best hierarchy in this context is the hierarchy that matches the most answers given by workers.

The flaws of the Noisy-Realizable setting are apparent. It does not explicitly consider the generative process of the answers since it wraps up all the possible uncertainties into one single channel noise. We propose a new model considering the generative process of answers. The new model is represented by the plate notation in Figure B.1 (b). To generate an answer a to a question q (not shown in all the plate notations), the first step is to choose a hierarchy T according to the distribution $P(T|W)$. The next step is to generate an answer to q according to T and the noise rate γ, where γ represents how likely the answer given by T will be flipped by a worker. This model is more flexible than the original one since it allows different answers to be generated by different hierarchies. However, this model does not consider the fact that one worker will use the same hierarchy to answer all the questions he is given. We call this model worker-ignorant model. We refine the worker-ignorant model such that all answers given by the same worker are generated by the same hierarchy. Figure B.1 (c) shows the new model, referred to as worker-aware model.



(a) Realizable setting. (b) Worker-ignorant setting. (c) Worker-aware setting.

Figure B.1: Generative process of answers.

The worker-aware model also enables a robot to estimate the personalized hierarchy for each worker. For many robotics applications, personalized knowledge of individual users has been shown to be more useful. For example, a robot could enhance the satisfaction of users if it provides personalized service [127]. Personalization could also help a robot tutor to yield significant benefits in educational/assistive human-robot interactions [129]. We believe knowing the personalized hierarchy could potentially enhance the experience of a human being during his/her interaction with a robot.

In this chapter, we extend the model used in Chapter 4 to explicitly model the generative process of answers. The new model is a two-level hierarchical Bayesian model represented by the plate notation in Figure B.1 (c). Each answer given by a worker is modeled as a mixture over underlying hierarchies. Each hierarchy is modeled as a compact model over finite number of possible hierarchies. We present an efficient approximate inference based on sampling and an EM algorithm to estimate the model parameters. We also propose a new Active Learning method to select the most informative questions toward uncovering the underlying hierarchies.

B.2 Models, Parameter Estimation and Inference

We focus on learning a distribution over hierarchies via asking crowdsourcing workers simple questions regarding to the structure of the underlying hierarchies. The workers will give their answers to these questions according to their belief of the hierarchies.

Formally, we define the following terms. Suppose there are N workers, and M questions. We introduce an indicator matrix $\Pi = (\pi_{i,j})_{M\times N}$, where $\pi_{i,j} = 1$ indicates that the j-th worker gives an answer to the i-th question, and $\pi_{i,j} = 0$ indicates the other way. We also define an answer matrix A denoted by $A \in \{1, 0\}^{M\times N}$, in which $a_{i,j}$ is the answer to the i-th question, given by the j-th worker. Our goal is to learn a distribution $P(T|W)$ over hierarchies $T \in \mathcal{T}$, where T is a tree structure with L nodes. The model parameter is a weight matrix $W = (w_{i,j})_{(L+1)\times L}$, where $w_{i,j}$ is a non-negative weight for the edge $e_{i,j}$. The probability distribution is denoted as follows:

$$P(T|W) = \frac{\prod_{e_{i,j}\in T} w_{i,j}}{Z(W)}, \tag{B.1}$$

where $Z(W) = \sum_{T'\in\mathcal{T}} \prod_{e_{i,j}\in T'} w_{i,j}$ is the partition function.

We will discuss two new generative probabilistic models corresponding to the plate notations of Figure B.1 (b) and (c).

B.2.1 Models

Worker-Ignorant Model

The worker-ignorant model in Figure B.1 (b) assumes the following generative process for each answer a in A. It first chooses a hierarchy T according to $P(T|W)$. It then uses T to decide the answer $T(q)$ to the current question q. Finally, it generates the final answer a, such that $a = T(q)$ with a probability $1 - \gamma$.

Given the parameters W and γ, the joint distribution of the set of answers A, and the corresponding hierarchies $\boldsymbol{T} = \{T_1, \ldots, T_{M\times N}\}$ is given by:

$$P(A, \Pi, \boldsymbol{T}|W, \gamma) = \prod_{i,j} \left(P(a_{i,j}|T_{i,j}, \gamma)P(T_{i,j}|W)\right)^{\pi_{i,j}}, \tag{B.2}$$

where

$$P(a_{i,j}|T_{i,j}, \gamma) = \begin{cases} (1-\gamma)^{a_{i,j}}\gamma^{1-a_{i,j}}, \text{ if } T(p_{i,j}) = 1 \\ \gamma^{a_{i,j}}(1-\gamma)^{1-a_{i,j}}, \text{ o.w.} \end{cases} \tag{B.3}$$

Summing over $\boldsymbol{T}$, we obtain the marginal distribution of the answers:

$$P(A, \Pi|W, \gamma) = \prod_{i,j} \sum_{T \in \mathcal{T}} \left(P(a_{i,j}|T_{i,j} = T, \gamma)P(T_{i,j} = T|W)\right)^{\pi_{i,j}}. \tag{B.4}$$

Worker-Aware Model

The worker-aware model in Figure B.1 (c) assumes that all the answers given by the same worker are generated from the same hierarchy T. So this model first generates a hierarchy for each worker, and the worker uses the hierarchy to generate all his answers. So, the marginal distribution of the answers is given by:

$$P(A, \Pi|W, \gamma) = \prod_{i} \sum_{T \in \mathcal{T}} \prod_{j} \left(P(a_{i,j}|T_i = T, \gamma)P(T_i = T|W)\right)^{\pi_{i,j}}. \tag{B.5}$$

As can be seen, worker-ignorant model is a special case of worker-aware model, where each worker is only allowed to answer one question. So, in the rest of this section we will just derive parameter estimation and inference for the worker-aware model.

B.2.2 Model Parameter Estimation

In this section we present an EM algorithm for parameter estimation in the worker-aware model. In particular, given a set of answers $A = (a_{i,j})_{M \times N}$, we wish to find parameters W and γ that maximize the marginal log-likelihood of of the data:

$$\log P(A, \Pi|W, \gamma) = \sum_i \log \sum_{T \in \mathcal{T}} \prod_j \left(P(a_{i,j}|T_i = T, \gamma) P(T_i = T|W)\right)^{\pi_{i,j}} . \tag{B.6}$$

As can be seen that the ground truth hierarchy is hidden in the marginal log-likelihood. So we propose an Expectation-Maximizing (EM) algorithm to find the solutions [49, 145]. EM algorithm alternates between E-step and M-step. In the E-step, the algorithm computes the posterior probabilities of the hidden variables to derive a tightest lower bound of the real log-likelihood. In the M-step, the algorithm optimizes the derived lower bound to update the model parameters. The algorithm iterates between the two steps until converges.

E-step: In the E-step, we need to compute the posterior probability of each hierarchy given the current parameters W, γ and data A. It is given by

$$\gamma(T_i = T|W, \gamma, A) \propto P(T_i = T|W) \prod_j P(a_{i,j}|T_i = T, \gamma)^{\pi_{i,j}} \tag{B.7}$$

according to Bayes' rule. Given the posterior probability, we can derive the tightest lower-bound of the true log-likelihood as follows:

$$F = \sum_i \sum_{T \in \mathcal{T}} \gamma(T_i = T|W, \gamma, A) \sum_j \pi_{i,j} \log P(a_{i,j}|T_i = T, \gamma) P(T_i = T|W). \tag{B.8}$$

Usually, the M-step maximizes the lower bound in equation (B.8) directly. However, computing equation (B.8) requires enumerating all the possible hierarchies $T \in \mathcal{T}$, which is intractable. Instead of minimizing the exact lower, we adopt the importance sampling method to get an empirical estimate of (B.8). Such that in the following M-step, our algorithm optimizes the approximate lower-bound rather than the exact lower bound.

Our target distribution is $\gamma(T_i = T|W, \gamma, A)$. Since we know how to draw samples from $P(T|W)$, we will use it as the proposal distribution. Such that the importance weight for each sample T is computed as follows:

$$w(T_i = T) = \frac{\prod_j \left(P(a_{i,j}|T,\gamma)\right)^{\pi_{i,j}}}{\sum_{T' \in \tilde{\pi}} \prod_j \left(P(a_{i,.}|T',\gamma)\right)^{\pi_{i,j}}}. \tag{B.9}$$

Given a set of samples $\tilde{\pi}$ from the true posterior distribution $\gamma(T|W,\gamma,A)$, the empirical estimate of the lower bound defined in equation (B.8) is then computed as follows:

$$\hat{F} = \sum_i \sum_j \pi_{i,j} \sum_{T \in \tilde{\pi}} w(T_i = T) \log P(a_{i,j}|T_i = T,\gamma) P(T_i = T|W) = \hat{F}_W + \hat{F}_\gamma, \tag{B.10}$$

where

$$\hat{F}_W = \sum_i \sum_j \pi_{i,j} \sum_{T \in \tilde{\pi}} w(T_i = T) \log P(T_i = T|W) \tag{B.11}$$

and

$$\hat{F}_\gamma = \sum_i \sum_j \pi_{i,j} \sum_{T \in \tilde{\pi}} w(T_i = T) \log P(a_{i,j}|T_i = T,\gamma) \tag{B.12}$$

M-Step: In the M-Step, we maximize (B.10) to update the model parameters. It is worth noting that optimizing W and γ are decoupled in the new objective function, which means we can optimize W and γ independently.

To optimize W, we observe that the objective function of maximizing $\hat{F}_W$ has a similar form as the objective function of equation (4.9) in the non-realizable settings. The only difference is that the weights in (B.10) are the importance weights, while in the previous problem the weights are the empirical estimations of the posterior distribution. Therefore, the new objective can be optimized using the same iterative optimization techniques used in Chapter 4.

To update γ, we can take a gradient of $\hat{F}_\gamma$ denoted as follows

$$\frac{\partial \hat{F}_\gamma}{\partial \gamma} = \sum_{i,j} \pi_{i,j} \sum_{T \in \tilde{\pi}} w(T_i = T) \left(T(a_{i,j}) \frac{1}{\gamma} - (1 - T(a_{i,j})) \frac{1}{1-\gamma} \right), \tag{B.13}$$

and set it to be 0. The solution is as follows:

$$\gamma = \frac{\sum_{i,j} \pi_{i,j} \sum_{T \in \tilde{\pi}: T(a_{i,j})=1} w(T_i = T)}{\sum_{i,j} \pi_{i,j}}. \tag{B.14}$$

B.2.3 Inference

Another key inference problem in the worker-aware model is to estimate the most probable hierarchies for each individual worker:

$$T^* = \arg\max_{T \in \mathcal{T}} P(T|A, \Pi, W, \gamma). \tag{B.15}$$

Unfortunately, directly solving this problem is intractable. Instead, we propose to find a variational distribution of this posterior distribution. Specifically, we find the variational distribution of the form defined in equation (B.1) and optimize the parameter $W^\dagger$, such that:

$$W^\dagger = \arg\min_{W} \mathcal{KL}\left(P(T|W)|P(T|A, \Pi, W, \gamma)\right). \tag{B.16}$$

Again, this problem can be solved using the same techniques used in Chapter 4. Then we find the MAP hierarchy according to the variational distribution, which can be solved efficiently using the minimum spanning tree algorithm as described in Chapter 4.

B.3 Active Learning in Bayesian Setting

In this section, we will discuss how to use active learning to determine the questions that are the most informative towards discovering the underlying hierarchy distribution. In Chapter 4, the active learning criterion is to reduce the uncertainty of the best hierarchy. In this chapter, our learning goal is to uncover the distribution over hierarchies parametrized by the parameter W. Therefore, the goal of the active query would be minimizing the uncertainty over W.

To estimate the uncertainty over W, we will extend the previous model and use a Bayesian parameter estimation, such that we can keep a density over all possible parameter Ws. Figure B.2

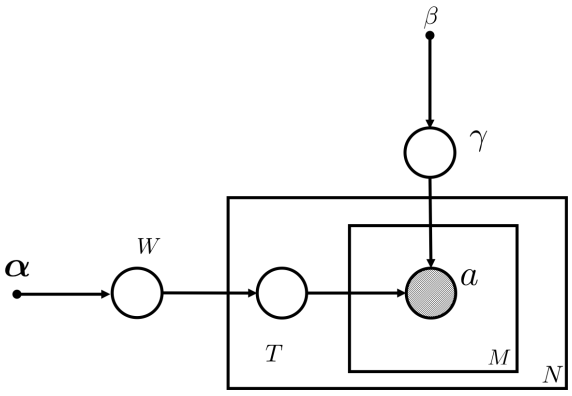


Figure B.2: Hierarchical Bayesian Model.

gives the plate notation of the new model. To make the problem tractable, we make an assumption that the columns of W are independent. This independence allows us to represent the joint distribution $P(W)$ as a set of independent distribution, one for each column $W_{\cdot,i}$.

Since each column of W is normalized to be 1, each $W_{\cdot,i}$ can be treated as a multinomial. For multinomials, the conjugate prior is a *Dirichlet* distribution [48], which is parametrized by hyperparameters $\alpha_i \in \mathcal{R}^+$. We will discuss how to update the parameters of the Dirichlet distribution with a new answer a to a path question q.

In the active learning setting, we have the ability to ask a certain question. We use Q to denote the set of candidate questions. In the case of learning hierarchies, Q includes all possible path questions $q_{i,j}, i \neq j$. To fully specify the algorithm, we need to address two issues: we need to describe how the parameters of the distribution is updated given the answer a to q, and we need to construct a mechanism for selecting the next query based on $P(W)$.

B.3.1 Updating Rule

In the Bayesian settings, we have one more hidden variable W. We present an empirical Bayes method for parameter estimation. In particular, given a set of answers $A = (a_{i,j})_{M \times N}$, we wish to find the parameter α and β that maximize the marginal log-likelihood of the data:

Algorithm 5 Gibbs Sampling Procedure

Input:

A: Set of answers given by crowdsourcing workers.

S: Number of samples

t': Burn in parameter

Output:

Samples for each T_i

Samples for parameter W

Initialize:

Initialize $W^{(0)}$, such that $P(T|W^{(0)})$ is a uniform distribution

Sample $T_i^{(0)} \sim P(T|W^{(0)})$

for $t = 1 \rightarrow S$ **do**

  for $i = 1 \rightarrow N$ **do**

    Sample $T_i^{(t)} \sim P(T_i|T_1^{(t)}, \ldots, T_{i-1}^{(t)}, T_{i+1}^{(t-1)}, \ldots, T_N^{(t-1)}, A, W^{(t-1)})$

  for $l = 1 \rightarrow L$ **do**

    Sample $W_{\cdot,l}^{(t)} \sim P(W_{\cdot,l}|\boldsymbol{T}^{(t)}, A, W_{\cdot,1}^{(t)}, \ldots, W_{\cdot,l-1}^{(t)}, W_{\cdot,l+1}^{(t-1)}, \ldots, W_{\cdot,L}^{(t-1)})$

Return: $T_i^{(t)}$ and $W^{(t)}$ for $t \geq t'$ and $i = 1, \ldots, N$.

$$P(A|\alpha,\beta) = \int\int P(W|\alpha) \prod_{i=1}^{M} \sum_{T \in \mathcal{T}} P(T_i = T|W) \prod_{j=1}^{N} P(a_{i,j}|T_j = T, \gamma) P(\gamma|\beta) dW d\gamma. \quad \text{(B.17)}$$

We drop Π to simplify the notation.

Again, the computation of this quality is not tractable. There are several strategies to learn this model, *e.g.*, using variational inference with EM as proposed by Blei et al., or using expectation propagation with EM [147]. In this work, we take the Gibbs Sampling methodology [82], which has similar performance as the EM-based method and is more robust to noise.

Algorithm 5 shows the procedure of Gibbs Sampling. The inference problem here is to estimate $P(T_1, \ldots, T_N, W_{\cdot,1}, \ldots, W_{\cdot,L}|A, \alpha, \beta)$. Suppose that we have chosen some initial hierarchies $T_1^{(0)}, \ldots, T_N^{(0)}$ and weight matrix $W^{(0)}$. Each step t of the Gibbs sampling procedure involves one of the two sampling process: 1) replacing T_i by a hierarchy drawn from the distribution of T_i con-

ditioned on remaining hierarchies $\boldsymbol{T}_{-i} = \{T_1^{(t)}, \ldots, T_{i-1}^{(t)}, T_{i+1}^{(t-1)}, \ldots, T_N^{(t-1)}\}$ and W; 2) replacing $W_{\cdot,l}^{(0)}$ by a multinomial distribution draw from the distribution of $W_{\cdot,l}^{(0)}$ conditioned on $\boldsymbol{T}$ and the remaining $W_{\cdot,-i} = \{W_{\cdot,1}^{(t)}, \ldots, W_{\cdot,l-1}^{(t)}, W_{\cdot,l+1}^{(t-1)}, \ldots, W_{\cdot,L}^{(t-1)}\}$. The procedure is repeated by cycling through all the T_is and $W_{\cdot,l}$s.

The major component of the algorithm is sampling from the conditional distribution $P(T_i|\boldsymbol{T}_{-i}, A, W)$ and $P(W_{\cdot,l}|\boldsymbol{T}, A, W_{\cdot,-l})$.

The conditional posterior distribution of T_i is given by:

$$P(T_i|\boldsymbol{T}_{-i}, A, W) = P(A_{\cdot,i}|T_i)P(T_i|W), \tag{B.18}$$

which is derived based on the conditional independence of the model. The first term of equation (B.18) is the likelihood, and the second term is the prior. Since we can sample from $P(T_i|W)$ according to the results in Chapter 4, we can also sample from the posterior distribution in equation (B.18) using *e.g.*, rejection sampling.

The posterior of $W_{\cdot,l}$ is given by:

$$P(W_{\cdot,l}|T, A, W_{\cdot,-l}) \propto P(A|T)P(T|W)P(W_{\cdot,l}|W_{\cdot,-l}). \tag{B.19}$$

Since we make an independence assumption for the parameters W that the columns of W are independent of each other, the last term of equation (B.19) only depends on the hyper parameter α_l. So we can use rejection sampling to sample from the target distribution in equation (B.19) with $P(W_{\cdot,l}|\alpha_l)$ as the proposal distribution.

B.3.2 Selection Rule

We have investigated several entropy-based strategies for selecting a target path question.

Maximum Entropy of Path Questions We can evaluate the entropies of questions using the set of tree samples sampled according to Algorithm 5. Therefore, the system will select the most confusing path questions.

Maximum Entropy of Weight Parameters We can estimate the entropies of the entries of weight matrix using the samples of weight matrix given by Algorithm 5. It is worth noting that weights reflect the likelihood of edges, but we are asking path questions. We use the uncertainty of W as a surrogate for the path uncertainty.

B.4 Summary

In this chapter, we discuss how to extend the hierarchy-building method to learn user-specific hierarchies. We propose a two-level hierarchical Bayesian model to model the generative process of all the questions, and provide an EM algorithm to learn model parameters. We also discuss several active learning methods over the model based on information gain criterion, and propose a Gibbs Sampling method to update the model parameters. In the future, we will perform evaluation of the proposed methods on both simulation and real-world data.