

Interactive and Interpretable Machine Learning Models for Human Machine Collaboration

by

Been Kim

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Aeronautics and Astronautics

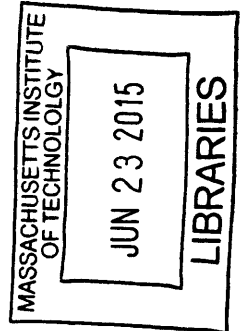
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.

ARCHIVES



Author **Signature redacted**
Department of Aeronautics and Astronautics
Signature redacted May 18, 2015

Certified by
Prof. Julie Shah
Thesis Supervisor

Certified by.. **Signature redacted**
Prof. Randall Davis
Thesis Committee Member

Certified by... **Signature redacted**
Prof. Cynthia Rudin
Thesis Committee Member

Accepted by... **Signature redacted**
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Interactive and Interpretable Machine Learning Models for Human Machine Collaboration

by

Been Kim

Submitted to the Department of Aeronautics and Astronautics
on May 18, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

I envision a system that enables successful collaborations between humans and machine learning models by harnessing the relative strength to accomplish what neither can do alone. Machine learning techniques and humans have skills that complement each other — machine learning techniques are good at computation on data at the lowest level of granularity, whereas people are better at abstracting knowledge from their experience, and transferring the knowledge across domains. The goal of this thesis is to develop a framework for human-in-the-loop machine learning that enables people to interact effectively with machine learning models to make better decisions, without requiring in-depth knowledge about machine learning techniques. Many of us interact with machine learning systems everyday. Systems that mine data for product recommendations, for example, are ubiquitous. However these systems compute their output without end-user involvement, and there are typically no life or death consequences in the case the machine learning result is not acceptable to the user. In contrast, domains where decisions can have serious consequences (e.g., emergency response planning, medical decision-making), require the incorporation of human experts' domain knowledge. These systems also must be transparent to earn experts' trust and be adopted in their workflow. The challenge addressed in this thesis is that traditional machine learning systems are not designed to extract domain experts' knowledge from natural workflow, or to provide pathways for the human domain expert to directly interact with the algorithm to interject their knowledge or to better understand the system output. For machine learning systems to make a real-world impact in these important domains, these systems must be able to communicate with highly skilled human experts to leverage their judgment and expertise, and share useful information or patterns from the data.

In this thesis, I bridge this gap by building human-in-the-loop machine learning models and systems that compute and communicate machine learning results in ways that are compatible with the human decision-making process, and that can readily incorporate human experts' domain knowledge. I start by building a machine learning model that infers human teams' planning decisions from the structured form of natural language of team meetings. I show that the model can infer a human teams' final plan with 86% accuracy

on average. I then design an interpretable machine learning model then “makes sense to humans” by exploring and communicating patterns and structure in data to support human decision-making. Through human subject experiments, I show that this interpretable machine learning model offers statistically significant quantitative improvements in interpretability while preserving clustering performance. Finally, I design a machine learning model that supports transparent interaction with humans without requiring that a user has expert knowledge of machine learning technique. I build a human-in-the-loop machine learning system that incorporates human feedback and communicates its internal states to humans, using an intuitive medium for interaction with the machine learning model. I demonstrate the application of this model for an educational domain in which teachers cluster programming assignments to streamline the grading process.

Prof. Julie Shah
Thesis Supervisor

Prof. Randall Davis
Thesis Committee Member

Prof. Cynthia Rudin
Thesis Committee Member

Prof. Finale Doshi-Velez
Thesis Reader

Prof. Andrea Thomaz
Thesis Reader

Acknowledgement

I would like to thank my advisor Prof. Julie Shah, who has been an amazing mentor during my time at MIT. Her encouragement and support kept me energetic while working to complete this thesis. She has been not only a brilliant person that I feel lucky to have an opportunity to work with, but also a person that I can rely on for advice and guidance for just about anything. I am also grateful for the opportunities that she gave me to travel, give lectures, and be a mentor of many undergraduate students. I was lucky to have her influence on my perspectives, my work and my values.

I would also like to thank my committee members, Prof. Cynthia Rudin and Prof. Randall Davis. The first course I took when I joined the PhD program was the machine learning course that Cynthia taught. Her passion for solving real-world problems and for building models that are useful for practitioners inspired my own work on interpretable models. I would like to thank her for all her support, kindness and encouragement throughout my program. Randy has also been a very supportive and helpful committee member. He went above and beyond in terms of the depth and detail of his feedback, which my work has greatly benefited from. I also would like to thank Prof. Andrea Thomaz and Prof. Finale Doshi-Velez for being readers of my thesis and providing helpful feedback for improvement.

I would like to thank Seth Teller, who has been a great mentor for my entire time here at MIT. He always believed in me and the things that I could achieve, and would always make time for talking with me. He really cared. I miss him very much.

I was lucky to have a fantastic team of undergraduate researchers who worked with me — Brittney Johnson, Caleb Chacha, Janelle Mansfield and Alex Lednev. These brilliant people have contributed in various parts of my thesis.

Elena Glassman made working on the last piece of my thesis a joy. I am grateful for having her as a collaborator and a friend. I would like to thank the interactive robotics group members for being amazing supporters and collaborators, who also happen to be wonderful people that never fail to brighten up my day.

I would like to thank Peter Krafft for being so kind and supportive. Without him, many moments would have been statistically significantly less enjoyable and fun.

Finally, I am very grateful for my family, without whom I would never have had the wonderful opportunities I had at MIT. My mom, dad and my sister have always believed in me and been supportive of what I do, no matter what I do. Thank you for your love and support.

Contents

1	Introduction	17
1.1	Infers Decisions of Humans — Inferring Human Task Plan	23
1.2	Make Sense To Humans — Provide Explanations to Humans (Bayesian Case Model)	26
1.3	Interact with Humans — interactive Bayesian Case Model	29
2	Inferring Team Task Plans from Human Meetings: A Generative Modeling approach with Logic-Based Prior	35
2.1	Introduction	35
2.2	Problem Formulation	37
2.2.1	Input	38
2.2.2	Output	41
2.3	Approach in a Nutshell and Related Work	41
2.3.1	Plan Recognition	43
2.3.2	Combining Logic and Probability	46
2.4	My Approach	47
2.4.1	Generative Model	47
2.4.2	Plan validation tool	51
2.4.3	Gibbs Sampling	56
2.5	Experimentation	59
2.5.1	Web-based Tool Design	59
2.5.2	Scenarios	60
2.6	Evaluation	63

2.6.1	Human Team Planning Data	64
2.6.2	Method Implementation	64
2.6.3	Concept-of-Operations Robot Demonstration	66
2.7	Discussion	68
2.7.1	Results	68
2.7.2	Sampling Hyper-parameters	69
2.7.3	The Benefit of PDDL	73
2.7.4	The i.i.d Assumption on the Utterance in the Generative Model	75
2.7.5	Engineering the Proposal Distribution in the Metropolis-Hastings Sampling Algorithm	76
2.8	Conclusion and Future Work	78
3	The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification	81
3.1	Introduction	82
3.2	Background and Related Work	83
3.3	The Bayesian Case Model	84
3.3.1	Motivating example	88
3.3.2	Inference: collapsed Gibbs sampling	90
3.4	Results	90
3.4.1	BCM maintains prediction accuracy.	91
3.4.2	Verifying the interpretability of BCM	93
3.4.3	Learning subspaces	96
3.5	Conclusion	96
4	iBCM: Interactive Bayesian Case Model — Empowering Humans via Intuitive Interaction	99
4.1	Introduction	100
4.2	Related work	102
4.3	Interactive Bayesian Case Models (iBCM)	104
4.3.1	Bayesian Case Model	104

4.3.2	Interactive Bayesian Case Model (iBCM)	106
4.4	Evaluation	114
4.4.1	Human subject experiment for a domain with ground truth	114
4.4.2	Implementing and validating iBCM system for online education	118
4.5	Conclusion	125
5	Conclusion and Future Work	127
A	The visualization of Gibbs Sampling Convergence: Trace Plot	133

List of Figures

1-1	Outline of the thesis	18
1-2	Learned prototypes and subspaces for the <i>Handwritten Digit</i> and <i>Recipe</i> datasets. Ingredients in red rectangles are subspaces of each prototype in <i>Recipe</i> datasets.	28
1-3	Interface combining iBCM to help teachers grade and provide useful feedback for students	32
2-1	Web-based tool developed and used for data collection	42
2-2	Graphical model representation of the generative model. The <i>plan</i> latent variable represents the final agreed-upon plan. The p_t^n variable represents the n^{th} predicate of t^{th} utterance, while s_t^n represents the absolute ordering of that predicate in the <i>plan</i> . The s_t' represents the relative ordering of s_t^n within the utterance t . The latent variable ω_p represents the noisiness of predicates, and β represents the noisiness of the ordering.	48
2-3	Radioactive material leakage scenario	61
2-4	Police incident response scenario	62
2-5	Percent improvements in median noise rejection and median sequence accuracy when sampling hyper-parameters versus setting $\omega_p = 0.8$ and $\beta = 5$	70
2-6	Inferred values of hyper-parameters (only showing subset of data set) . .	71
2-7	Ratio of runs that show the benefit of using PDDL	74
2-8	The distribution of the utterances included in the final plan (normalized)	76

2-9	The impact of different proposal distributions (The highest accuracy with perfect PDDL files)	78
3-1	Graphical model for the Bayesian Case Model	85
3-2	Prediction test accuracy reported for the <i>Handwritten Digit</i> [73] and <i>20 Newsgroups</i> datasets [88]. (a) applies SVM for both LDA and BCM, (b) presents the unsupervised accuracy of BCM for <i>Handwritten Digit</i> (top) and <i>20 Newsgroups</i> (bottom) and (c) depicts the sensitivity analysis conducted for hyperparameters for <i>Handwritten Digit</i> dataset. Datasets were produced by randomly sampling 10 to 70 observations of each digit for the <i>Handwritten Digit</i> dataset, and 100-450 documents per document class for the <i>20 Newsgroups</i> dataset. The <i>Handwritten Digit</i> pixel values (range from 0 to 255) were rescaled into seven bins (range from 0 to 6). Each 16-by-16 pixel picture was represented as a 1D vector of pixel values, with a length of 256. Both BCM and LDA were randomly initialized with the same seed (one half of the labels were incorrect and randomly mixed), The number of iterations was set at 1,000. $S = 4$ for <i>20 Newsgroups</i> and $S = 10$ for <i>Handwritten Digit</i> . $\alpha = 0.01, \lambda = 1, c = 50, q = 0.8$	92
3-3	Web-interface for the human subject experiment	94
3-4	Learned prototypes and subspaces for the <i>Handwritten Digit</i> and <i>Recipe</i> datasets.	95
4-1	Graphical model depicting the BCM and iBCM. Double circle nodes represent <i>interacted latent variables</i>	105
4-2	A subset of data points. Each data point has two features (top) or three features (bottom): shape, color and pattern.	113
4-3	Graphical user interface for interaction. Each row represents a cluster. Prototypes of each cluster are shown on the left. The numbers below them are data ID. Subspaces are marked as stars. Each data point has three features: shape, color and pattern.	116
4-4	The raw data	119

4-5	Experiment interface for iBCM	120
4-6	Experiment interface for BCM with random restarts	121
5-1	A portion of the analysis report using the data representation studied in this work. Each data point is compressed using topic-based feature compression. Zoomed in (top left), where a pie chart inside of a red square indicates incorrectly classified data points. The document ID number and true labels are also shown as texts. The top 10 words within each color-coded topic are shown at the top right. The bottom shows reports in a bird's-eye view.	131
A-1	Trace Plots (only showing a subset of the data set)	134

List of Tables

2.1	Utterance tagging: Dialogue and structured form examples. (The structured form uses the following shorthand - ST: send to, rr: red robot, br: blue robot, rm: red medical, bm: blue medical, e.g. ST(br,A) : "send the blue robot to room A.")	39
2.2	Resources available in police incident response scenario	63
2.3	Radioactive material leakage scenario plan accuracy results, before announcement (13 teams / 26 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.	67
2.4	Radioactive material leakage scenario plan accuracy results, after announcement (21 teams / 42 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.	67
2.5	Police incidents response scenario plan accuracy results (14 teams / 28 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.	67

3.1	The mixture of smiley faces for LDA and BCM	89
4.1	Experiment design. Participants were randomly assigned to one of four groups, with four participants per group. "Is balanced" means that there were an equal number of data points with each feature. (e.g., equal number of red squares and yellow squares).	114
4.2	Human subject experiment results	117
4.3	Survey questions (tool A: iBCM, tool B: BCM with re-groupings).	124

Chapter 1

Introduction

Machine learning (ML) techniques and humans have skills that complement each other — ML techniques are good at computation on data at the lowest level of granularity, whereas people are better at abstracting knowledge from their experience, and transferring the knowledge across domains. I envision a system that enables successful collaborations between humans and machine learning models by harnessing their relative strength to accomplish what neither can do alone. The goal of this thesis is to develop a framework for human-in-the-loop machine learning that enables people to interact effectively with ML models to make better decisions, without requiring in-depth knowledge about machine learning techniques.

Many of us interact with ML systems everyday. Systems that mine data for product recommendations, for example, are ubiquitous. However these systems compute their output without end-user involvement, and there are typically no life or death consequences in the case the ML result is not acceptable to the user. In contrast, domains where decisions can have serious consequences (e.g., emergency response planning, and medical decision-making), require the incorporation of human experts' domain knowledge. These systems also must be transparent to earn experts' trust and be adopted in their workflow. The challenge addressed in this thesis is that traditional machine learning systems are not designed to extract domain experts' knowledge from natural workflow, or to provide pathways for the human domain expert to directly interact with the algorithm to inject their knowledge or to better understand the system output. For ML systems to make a real-

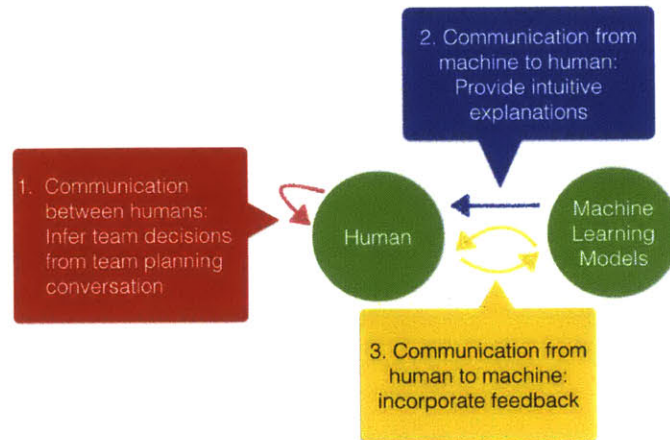


Figure 1-1: Outline of the thesis

world impact in these important domains, these systems must be able to communicate with highly skilled human experts to leverage their judgment and expertise, and share useful information or patterns from the data.

In this thesis, I bridge this gap by building human-in-the-loop machine learning models and systems that compute and communicate machine learning results in ways that are compatible with the human decision-making process, and that can readily incorporate human experts' domain knowledge. This process is depicted in Figure 1-1. I start by building a machine learning model that infers human teams' planning decisions from the structured form of natural language of team meetings. I then design an interpretable ML model that "makes sense to humans" by exploring and communicating patterns and structure in data to support human decision-making. Finally, I design a machine learning model that supports transparent interaction with humans without requiring that a user has expert knowledge of the machine learning technique. I build a human-in-the-loop machine learning system that incorporates human feedback and communicates its internal states to humans, using an intuitive medium for interaction with the ML model.

There has been limited prior work on the explicit design of ML systems that interactively incorporate human expert knowledge and explain the results in an intuitive manner. Related work can be categorized in two ways — the direction of communication

(e.g., humans to machines, machines to humans, between machines and humans) and the level of system modification upon the interaction (i.e., how much the internal states of models or systems change). Active learning [37, 128, 102], semi-supervised learning [36, 15], interactive machine learning [18, 89, 7] and a set of application-specific systems [119, 125, 42, 131, 5, 14, 7] are some examples of work that enable communication from humans to machines while improving the way to represent machines' internal states via interaction [89, 65, 8] or building models whose internal states are interpretable [126, 43, 45, 39, 20] to enable communication from machines to humans. Work that enables the communication between human experts and machines include expert systems [30, 26, 28, 29, 47, 98, 42] and a subset of interactive machine learning systems [92].

Another way to categorize related work is the way in which the machines' internal states are updated and modified upon the interaction from/to humans. Some work makes no changes to internal states [126, 43, 45, 39, 20], while some work changes only the way information is presented upon the interaction [89, 65, 8]. Others update and modify internal states of models and systems based on the interaction (e.g., as is done in active learning [37, 128, 102], semi-supervised learning [36, 15], interactive machine learning [18, 89, 7], and expert systems [30, 26, 28, 29, 47, 98, 42]).

Systems using active learning and semi-supervised learning allow interaction from humans to machines in order to improve the goal task by updating the internal states of models and systems. Interacting with humans to gather data labels that are informative for the machine learning model's task is one of the most common forms of interaction, as is done in active learning [37, 128, 102]. In an active learning framework, the new data point to be queried is often selected by machine learning models, not by humans. On the other hand, semi-supervised methods [36, 15] enable humans to select the data points based on their browsing and exploration of the data. Then, human feedback is often incorporated into machine learning models in the form of constraints or adjustment on the underlying similarity metric used by the clustering algorithm [21, 16]. While these frameworks are designed to interact with humans, they are not designed to "make sense to humans" by building a unified framework to explain the machine learning results through

interaction with humans.

Interactive machine learning systems enable communication from humans to machines and the internal states of machines and systems are often updated upon the interaction. Interactive machine learning systems include systems that are using active learning and semi-supervised learning frameworks as well as systems that allow humans to provide information beyond data labels. In more complex domains, such as clustering documents, interactive clustering systems are available [18, 89]; however, the medium of interaction assumes that a user has an expert level of knowledge, for example about n-grams for machine learning systems [18] or keyword weights for topic clusters [89]. Some works had users communicate their feedback through data points [7, 6], such as by providing positive and negative examples to train a classifier [6]. However, the difference between the machine representation (e.g., the averaged features of data points) and representation of information obtained from human feedback (i.e., the examples) may lead to inconsistencies in how user feedback is incorporated into the machine.

Interactive interfaces and interactive visualization enable communication from machines to humans with small to no changes to internal states of models or systems. Interfaces that improve user workflow when modifying model settings are intended to reduce the burden of repeated processes to achieve the desired clustering results [105]. Instead of modifying model settings, some systems aim to improve the user's internalization of the current clustering through visualization [89, 65, 8]. Visualization techniques are often developed for specific domains (e.g., topic model visualization [31]). Some also introduce new metrics to help users internalize and conceptualize clustering results [63].

Another way to enable communication from machines to humans is to build models whose internal states are interpretable to humans. Interpretable models enable communication from machines to humans, without changing internal states of models. In my view, there are at least three widely known types of interpretable models: sparse linear classifiers ([126, 32, 129]); discretization methods, such as decision trees and decision lists (e.g., [43, 134, 45, 90, 60]); and prototype- or case-based classifiers (e.g., nearest neighbors [39] or a supervised optimization-based method [20]). The work presented in this thesis is intended as the third model type, but uses unsupervised generative mechanisms

to explain clusters instead of supervised approaches [61] or myopic focus on neighboring points [13].

Expert systems allow communication between humans and machines and modify their internal states based on the information gathered from communication. An expert system is defined as a computer system that emulates the decision-making abilities of human experts [75] and is of particular to the fields of medicine [120], fault diagnosis [3] and science [135]. The most relevant works to this thesis are those that design systems to interactively communicate the ML results [30, 26, 28, 29, 47, 98]. For example, Davis et al. [42] enable experts to interact with their system (TEIRESIAS) using styled interaction to build a knowledge base. The work support two-way interaction — the system explains the results and motivations for its actions to the human experts, and the human experts adds to or modifies the knowledge base. The distinguishing feature of this prior work as it relates to the thesis is that this approach enables interaction with users who are not programmers or machine learning experts. The system guides the user to add to or modify rules by asking questions, such as yes/no questions. The work demonstrates that it is advantageous to ask questions, as it helps to narrow down the missing or wrong knowledge quickly. The system also supports “second guessing” in which the model checks whether the new knowledge from the user is consistent with its existing model. However, one key disadvantage of these expert systems is that they do not leverage machine learning techniques to also incorporate information or structure from large data sets that can potentially improve results.

Lin et al. [92] leverages both machine learning techniques and domain expert knowledge by combining past data of lost-person behaviors with search and rescue workers' domain knowledge to generate search plans that target the likely walking paths of lost-persons. However, the interaction between the human experts and the model (machine) is highly constrained. The experts' knowledge is taken into account as a prior probability distribution over likely travel paths and there is no feedback loop through which human experts can interactively contribute their domain knowledge and observe the impact of their feedback. In contrast, the proposed work aims to support more flexible interactions between humans and machine learning models that allow interactive incorporation

of domain experts' knowledge.

This thesis aims to provide a framework and proof-of-concept implementations that establish collaboration between humans and ML systems, where the system leverages information in data, communicates it to the user in an interpretable manner, and the user communicates their knowledge with the system to improve the final result. This thesis aims to demonstrate the feasibility of this approach with users who are not experts in ML models.

This chapter provides an executive summary of the key innovations presented in this thesis; its section structure follows the chapter structure of this thesis. In Section 1.1, I present a model that infers decisions of humans — it infers the final plan from a processed form of the human team's planning conversation. I show that the model can infer a human teams' final plan with 86% accuracy on average. Next, I propose a model that can make sense to humans in Section 1.2. I propose and validate a new unsupervised clustering model that provides interpretable explanations to humans about machine learning results, who use examples as a fundamental part of their decision-making strategies. This model, Bayesian Case Model (BCM), is a general framework for Bayesian case-based reasoning that brings the intuitive power of case-based reasoning to a Bayesian generative framework. Through human subject experiments, I show that BCM offers statistically significant quantitative improvements in interpretability while preserving clustering performance. Building on to BCM, Section 1.3 presents the interactive Bayesian Case Model (iBCM), which opens a communication channel between the clustering model and the user. Users provide direct input to iBCM in order to achieve effective clustering results, and iBCM optimizes the clustering by achieving a balance between what the data indicate and what makes the most sense to the user. iBCM provides feedback for users and does not assume any prior knowledge of machine learning on their part. I demonstrate the application of this model for an educational domain in which teachers cluster programming assignments to streamline the grading process.

1.1 Infers Decisions of Humans — Inferring Human Task Plan

The goal in this chapter is to demonstrate the feasibility of an ML system to extract relevant information from the natural workflow of domain experts. As proof-of-concept, I study the emergency response domain, in which teams of people plan the deployment of resources, and I present a machine learning model that infers the final plan from a processed form of the human team's planning conversation.

This hybrid approach combines probabilistic generative modeling with logical plan validation used to compute a highly structured prior over possible plans, enabling us to overcome the challenge of performing inference over a large solution space with only a small amount of noisy data from the team planning session. I validate the model through human subject experimentations and show that it is able to infer a human team's final plan with 86% accuracy on average. I also describe a robot demonstration in which two people plan and execute a first-response collaborative task with a PR2 robot. To the best of my knowledge, this is the first work to integrate a logical planning technique within a generative model to perform plan inference.

Goal: Infer human teams' final plan from a processed form of the human team's planning conversation

I describe the formal definition, the input and output of the problem. Formally, this problem can be viewed as one of plan recognition, where the plan follows the formal representation of the Planning Domain Description Language (PDDL). The PDDL has been widely used in the planning research community and planning competitions (i.e., the International Planning Competition). A plan is *valid* if it achieves a user-specified goal state without violating user-specified plan constraints. Actions may be constrained to execute in sequence or in parallel with other actions. Other plan constraints can include discrete resource constraints (e.g. the presence of two medical teams) and temporal deadlines (e.g. a robot can only be deployed for up to 1 hour at a time due to battery life

constraints).

The input to the model is a machine-understandable form of human conversation data. Text data from the human team conversation is collected in the form of utterances, where each utterance is one person's turn in the discussion. Each utterance is formally represented as ordered tuple of sets of grounded predicates. This structured form captures the actions discussed and the proposed ordering relations among actions for each utterance. In this work, I focus on inferring a final plan using text data that can be logged from chat or transcribed speech. Processing human dialogue into more machine-understandable forms is an important research area [86, 124, 84, 104, 110], but I view this as a separate problem and do not focus on it in this paper.

The form of input I use preserves many of the challenging aspects of natural human planning conversations, and can be thought of as noisy observation of the final plan. Because in my work, the team is discussing the plan under time pressure, planning sessions often consist of a small number of succinct communications. This hybrid approach combines probabilistic generative modeling with logical plan validation can infer the final agreed-upon plan using a single planning session, despite a small amount of noisy data.

The output of my program is an inferred final plan, sampled from the probability distribution over the final plans. The final plan has the same representation as the structured utterance tags (*ordered tuple of sets of grounded predicates*). The predicates in each set represent actions that should happen in parallel, and the ordering of sets indicates the sequence. Unlike the utterance tags, however, the sequence ordering relations in the final plan represent the *absolute* order in which the actions are to be carried out. An example of a plan is $(\{A_1, A_2\}, \{A_3\}, \{A_4, A_5, A_6\})$, where A_i represents a predicate. In this plan, A_1 and A_2 will happen at step 1 of the plan, A_3 happens at step 2 of the plan, and so on.

Approach: Combine a Bayesian Generative Model with Logic-Based Prior

Planning conversations performed under time pressure exhibit unique characteristics and challenges for inferring the final plan. First, these planning conversations are succinct — participants tend to write quickly and briefly, and to be in a hurry to make a final decision. Second, there may be a number of different valid plans for the team’s deployment — even with a simple scenario, people tend to generate a broad range of final plans. This variety of final plans represents the typical challenges faced during real rescue missions, where each incident is unique and participants cannot have a library of plans to choose from at each time. Third, these conversations are noisy — often, many suggestions are made and rejected more quickly than they would be in a more casual setting. In addition, there are not likely to be many repeated confirmations of agreement, which might typically ease detection of the agreed-upon plan.

It might seem natural to take a probabilistic approach to the plan inference problem, as we are working with noisy data. However, the combination of a small amount of noisy data and a large number of possible plans means that inference using typical, uninformative priors over plans may fail to converge to the team’s final plan in a timely manner.

This problem could also be approached as a logical constraint problem of partial order planning, if there were no noise in the utterances: If the team were to discuss only partial plans relating to the final plan, without any errors or revisions, then a plan generator or scheduler [38] could produce the final plan using global sequencing. Unfortunately, data collected from human conversation is sufficiently noisy to preclude this approach.

These circumstances provided motivation for a combined approach, wherein I built a probabilistic generative model and used a logic-based plan validator [72] to compute a highly structured prior distribution over possible plans. Intuitively speaking, this prior encodes my assumption that the final plan generated by human planners is likely, but not required, to be a valid plan. This approach naturally deals with both the noise in the data and the challenge of performing inference over plans with only a limited amount of data. I performed sampling inference in the model using Gibbs and Metropolis-Hastings sampling

to approximate the posterior distribution over final plans, and empirical validation with human subject experiments indicated that the model achieves 86% accuracy on average.

Results and Contributions

I formulate the novel problem of performing inference to extract a finally agreed upon plan from a human team planning conversation. I propose and validate a hybrid approach to perform this inference that applies the logic-based prior probability over the space of possible agreed-upon plans. This approach performs efficient inference for the probabilistic generative model. I demonstrate the benefit of this approach using human team meeting data collected from human subject experiments (total 96 subjects) and am able to infer a human team's final plan with 86% accuracy on average.

1.2 Make Sense To Humans — Provide Explanations to Humans (Bayesian Case Model)

I present the Bayesian Case Model (BCM), a general framework for Bayesian case-based reasoning (CBR) and prototype classification and clustering. BCM brings the intuitive power of CBR to a Bayesian generative framework. The BCM learns *prototypes*, the “quintessential” observations that best represent clusters in a dataset, by performing joint inference on cluster labels, prototypes and important features. Simultaneously, BCM pursues sparsity by learning *subspaces*, the sets of features that play important roles in the characterization of the prototypes. The prototype and subspace representation provides quantitative benefits in interpretability while preserving classification accuracy. Human subject experiments verify statistically significant improvements to participants' understanding when using explanations produced by BCM, compared to those given by prior art.

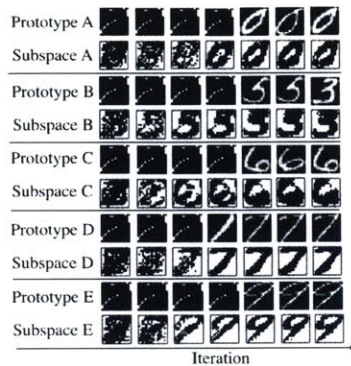
Goal: Provide quantitative benefits in interpretability while preserving classification accuracy

Numerous studies have demonstrated that exemplar-based reasoning, involving various forms of matching and prototyping, is fundamental to humans' most effective strategies for tactical decision-making ([101, 35, 83]). For example, naturalistic studies have shown that skilled decision makers in the fire service use *recognition-primed decision making*, in which new situations are matched to typical cases where certain actions are appropriate and usually successful [83]. To assist humans in leveraging large data sources to make better decisions, we desire that machine learning algorithms provide output in forms that are easily incorporated into the human decision-making process.

The same basic idea lies behind studies of human decision-making and cognition that provided the key inspiration for artificial intelligence Case-Based Reasoning (CBR) approaches [2, 122]. CBR relies on the idea that a new situation can be well-represented by the summarized experience of previously solved problems [122]. CBR has been used in important real-world applications [91, 19], but is fundamentally limited, in that it does not learn the underlying complex structure of data in an unsupervised fashion and may not scale to datasets with high-dimensional feature spaces (as discussed in [123]).

My approach, the Bayesian Case Model (BCM), simultaneously performs unsupervised clustering and learns both the most representative cases (i.e., prototypes) and important features (i.e., subspaces). BCM preserves the power of CBR in generating interpretable output (phrased in terms of specific examples), where interpretability comes not only from sparsity but from the prototype exemplars.

The inputs to this model can be any datasets with discrete values. The output of this model is clustering results of those data points and explanations of the clustering results using prototypes and subspaces.



(a) *Handwritten Digit* dataset

Prototype (Recipe names)	Ingredients (Subspaces)
<i>Herbs and Tomato in Pasta</i>	basil, garlic, Italian seasoning, oil, pasta, pepper, salt, tomato
<i>Generic chili recipe</i>	beer, chili powder, cumin, garlic, meat, oil, onion, pepper, salt, tomato
<i>Microwave brownies</i>	baking powder, sugar, butter, chocolate, chopped pecans, eggs, flour, salt, vanilla
<i>Spiced-punch</i>	cinnamon stick, lemon juice, orange juice, pineapple juice, sugar, water, whole cloves

(b) *Recipe* dataset

Figure 1-2: Learned prototypes and subspaces for the *Handwritten Digit* and *Recipe* datasets. Ingredients in red rectangles are subspaces of each prototype in *Recipe* datasets.

Approach: Combine Case-Based Reasoning and Bayesian Generative Model

In BCM, the prototype is the exemplar that is most representative of the cluster. The subspace representation is a powerful output of the model because it improves transparency — we neither need nor want the best exemplar to be similar to the current situation in all possible ways: for instance, a moviegoer who likes the same horror films as we do might be useful for identifying good horror films, regardless of their cartoon preferences. I model the underlying data using a mixture model, and infer sets of features that are important within each cluster (i.e., subspace). This type of model can help to bridge the gap between machine learning methods and humans, who use examples as a fundamental part of their decision-making strategies.

Results and Contributions

I show that BCM produces prediction accuracy comparable to or better than prior art for standard datasets (76% versus 77% on average). I also show that the exemplar-

based output of BCM resulted in statistically significant improvements to participants' performance of a task requiring an understanding of clusters within a dataset, as compared to outputs produced by prior art. BCM learns prototypes and subspaces that are presented as meaningful feedback for the characterization of important aspects of the dataset (shown in Figure 3-4).

1.3 Interact with Humans — interactive Bayesian Case Model

Clustering methods optimize the partitioning of data points with respect to an internal metric, such as likelihood, in order to approximate the goodness of clustering for users. However, this internal metric does not necessarily translate into effective clustering from the user's perspective. This work presents the interactive Bayesian Case Model (iBCM), a model that opens a communication channel between the clustering model and the user. Users can provide direct input to iBCM in order to achieve more useful clustering results, and iBCM optimizes the clustering by creating a balance between what the data indicate and what makes the most sense to the user. This model provides feedback for users and does not assume any prior knowledge of machine learning on their part. I provide quantitative evidence that users are able to obtain more satisfactory clustering results through iBCM than without an interactive model. I also demonstrate the use of this method in a real-world setting where computer language class teachers utilize iBCM to cluster students' coding assignments for grading.

Goal: Enable intuitive communication between a clustering model and humans to incorporate humans' expertise and leverage useful patterns in data

The ultimate goal of any clustering method is to partition data points in the most effective and useful way for the user. A number of existing methods [49] are able to group a large number of data points according to certain internal metrics. Unlike classification methods

where ground truth is available and prediction accuracy often serves as an evaluation metric, clustering metrics that directly optimize for a particular application are difficult to articulate and can be domain-specific. In clustering, there might be multiple ways to cluster data points that are equally good according to the internal metric of the given model, but some of these methods may better align with a user's knowledge or preference. For example, if a user in a retail setting wants to cluster customer data in order to build a recommendation engine and pair sales staff with appropriate customers using purchase records, clustering with respect to item categories or region of residence could yield equally good results. However, clustering with respect to region may offer a more cost-effective implementation of the engine by matching the recommendation plans with the existing team structure of the company. Hard-coding such information does not translate to other domains, and requires in-depth knowledge of machine learning.

iBCM aims to enable efficient and intuitive use of clustering models, without requiring in-depth knowledge about machine learning. iBCM combines CBR approaches [2, 122] with a Bayesian generative model to both generate effective clustering and provide an intuitive medium for interaction. Specifically, iBCM uses examples (data points provided by the user) for communication, enabling interaction between the model and any user with some degree of familiarity with the data. The use of examples as a medium for interaction is well-grounded in several cognitive science studies that demonstrated that exemplar-based reasoning, involving various forms of matching and prototyping, is fundamental to humans' most effective strategies for tactical decision-making ([101, 35, 83]). In order to assist user decision-making by leveraging data sources, we desire machine learning algorithms that communicate in forms that are easily incorporated into the human decision-making process.

The inputs to this model are datasets with discrete feature values and humans' feedback. The output of this model is clustering results of input data points and explanations of the clustering results.

Approach: Iteratively Incorporate Humans' feedback in the Inference Loop using Interactive Bayesian Case Model

iBCM empowers users via direct, intuitive interaction with a clustering model, in order to incorporate their domain knowledge and preference while simultaneously optimizing the internal goodness of clustering metric. Moreover, the communication channel is bi-directional: iBCM communicates back to its users by providing explanations in natural language, offering insight into potential conflicts between user feedback and data point patterns and allowing for more fluent collaboration between the user and machine learning models.

The communication channel does not assume that a user has prior knowledge of machine learning. Instead, users provide examples (i.e., data points) and features of these examples in order to articulate their feedback. For example, when building a recommendation engine for films, users may think that the length of a specific movie is not an important feature for clustering, even if its use would yield good results. Instead, the user may think that genre or the number of awards each film has received are more important features. iBCM incorporates the user's feedback and propagates it to the inference engine in order to achieve better clustering. If incorporating user-suggested features does not yield good clustering, iBCM then provides feedback using the examples the user provided.

Results and Contributions

I performed a human experiment, along with a proof-of-concept study, to validate our approach. The experiment was designed to measure quantifiable improvements in how well final clustering performed with iBCM matches the intent of the subject compared with clustering with BCM. The proof-of-concept study was a demonstration of iBCM in a real-world setting (computer science education).

```

def dotProduct(listA, listB):
    """
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    """
    res = 0
    for j in range(len(listA)):
        res = res + listA[j] * listB[j]
    return res

def dotProduct(listA, listB):
    """
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    """
    combo_list = []
    for number in range(len(listA)):
        combo_list.append(listA[number] * listB[number])
    return sum(combo_list)

def dotProduct(listA, listB):
    """
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    """
    dot_product = sum(listA[i] * listB[i] for i in range(len(listA)))
    return dot_product

```

(a) The raw data

<p>Cluster Prototypes and Subspaces</p> <p>Demote from Prototype</p> <pre> def dotProduct(listA, listB): total=0 for (a,b) in zip(listA, listB): product=a*b total+=product return total </pre>	<p>id: 1</p>	<p>Promote to Prototype</p> <pre> def dotProduct(listA, listB): iB=0 length=len(listA) total=0 while iB<length: total+=listA[iB]*listB[iB] iB+=1 return total </pre>	<p>id: 45</p>
<p>Demote from Prototype</p> <pre> def dotProduct(listA, listB): assert len(listA)==len(listB) return sum(a*b for(a,b) in zip(listA, listB)) </pre>	<p>id: 15</p>	<p>Promote to Prototype</p> <pre> def dotProduct(listA, listB): listC=[] iB=0 while iB<len(listA) and iB<len(listB): listC.append(listA[iB]*listB[iB]) iB+=1 return sum(listC) </pre>	<p>id: 52</p>
<p>Demote from Prototype</p> <pre> def dotProduct(listA, listB): length=len(listA) iB=0 total=0 while iB<length: total+=int(listA[iB])*int(listB[iB]) </pre>	<p>id: 62</p>	<p>Promote to Prototype</p> <pre> def dotProduct(listA, listB): total=0 </pre>	<p>id: 54</p>

(b) iBCM with features extracted from using OverCode

Figure 1-3: Interface combining iBCM to help teachers grade and provide useful feedback for students

Users are able to obtain statistically significantly more satisfactory clustering through iBCM.

In order to obtain objective measures of iBCM's performance, I designed an experiment such that the quality of clustering obtained through interaction can be evaluated while

controlling for data characteristics that can affect clustering results. I constructed datasets that could be clustered in multiple ways, all of which were equal in terms of likelihoods, with the optimal ways to cluster according to internal metrics also known a priori.

iBCM achieved significantly better agreement with the users' intentions, while creating a balance between what the data indicates and what makes sense to user in all given conditions. The complexity of the domain, the interaction with iBCM or of the dataset did not influence the effectiveness of the model. This experiment validates that the subjects achieved significantly more satisfying results using iBCM compared with BCM, even though the results from BCM provided equally good solutions according to the model's internal metric.

I demonstrate the use of iBCM for computer programming language education. Teachers are able to obtain statistically significantly more satisfactory exploring students submission using iBCM.

I also present an implementation of iBCM for a real-world application: introductory programming education. My institute, for example, offers several courses that teach basic programming to hundreds of students simultaneously. It can take hours for teachers to grade exams for courses of this size, and many teachers generate grading rubrics for each problem based on a small random sample of student solutions. Rather than depend on such a sample, I implemented a system for introductory Python programming teachers that uses iBCM as the main computational engine for grouping. The system utilizes iBCM to help teachers group their students' assignments for grading and provide helpful comments for the students. I invited twelve teachers of introductory Python computer language classes (i.e., domain experts) to interact with the system to cluster students' assignment submissions, to show that using iBCM allows the users to easily explore variations across hundreds of student solutions before designing a grading rubric or composing feedback for students. Teachers explored the range of student submission using iBCM and a version of BCM in which they requested multiple sets of clusters. On a Likert-style questionnaire, the teachers agreed more strongly that they were satisfied with the interaction and that they better explored student responses using iBCM compared to the BCM baseline. Responses

to the open-ended questionnaire also highlighted the strength of iBCM. Comments include “I was able to pull out examples that were really representative of a lot of the solutions and then skim through each solution set to find the minor differences.” and that iBCM allowed to explore “in depth as to how students could do”. A few also noted that “[iBCM] is useful with large datasets where brute-force would not be practical”.

Chapter 2

Inferring Team Task Plans from Human Meetings: A Generative Modeling approach with Logic-Based Prior

2.1 Introduction

The goal in this chapter is to demonstrate the feasibility of an ML system to extract relevant information from the natural workflow of domain experts. As proof-of-concept, I study the emergency response domain, in which teams of people plan the deployment of resources, and I present a machine learning model that infers the final plan from a processed form of the human team's planning conversation.

I present an approach that can interpret decisions of humans by inferring the final plan from a processed form of the human team's planning conversation. Inferring the plan from noisy and incomplete observation can be formulated as a plan recognition problem [117, 17, 95, 33, 25, 64, 51]. The noisy and incomplete characteristics of observation stem from the fact that not all observed data (e.g., the team's planning conversation) will be a part of the plan that I am trying to infer, and that the entire plan may not be observed.

The focus of existing plan recognition algorithms is often to search an existing knowledge base given noisy observation. However, deployment plans for emergency situations are seldom the same, making it infeasible to build a knowledge base. In addition, planning conversations are often conducted under time pressure and are, therefore, often short (i.e., contain a small amount of data). Shorter conversations result in a limited amount of available data for inference, often making the inference problem more challenging.

My approach combines probabilistic generative modeling with logical plan validation, which is used to compute a highly structured (i.e., non-uniform, more informative prior) prior over possible plans. This hybrid approach enables us to overcome the challenge of performing inference over a large solution space with only a small amount of noisy data collected from the team planning session.

In this thesis, I focus on inferring a final plan using text data that can be logged from chat or transcribed speech. Processing human dialogue into more machine-understandable forms is an important research area [86, 124, 84, 104, 110], but I view this as a separate problem and do not focus on it in this thesis.

The form of input I use preserves many of the challenging aspects of natural human planning conversations, and can be thought of as noisy observation of the final plan. Because the team is discussing the plan under time pressure, planning sessions often consist of a small number of succinct communications. My approach can infer the final agreed-upon plan using a single planning session, despite a small amount of noisy data.

I validate the approach through experiments with 96 human subjects and show that I am able to infer a human team's final plan with 86% accuracy on average. To the best of my knowledge, this is the first work to integrate a logical planning technique within a generative model to perform plan inference.

This chapter includes the following contributions:

- I formulate the novel problem of performing inference to extract a finally agreed-upon plan from a human team planning conversation.
- I propose and validate a hybrid approach to perform this inference that applies the logic-based prior probability over the space of possible agreed-upon plans. This

approach performs efficient inference for the probabilistic generative model.

- I demonstrate the benefit of this approach using human team meeting data collected from human subject experiments (total 96 subjects) and am able to infer a human team's final plan with 86% accuracy on average.

The formulation of the problem is presented in Section 2.2, followed by the technical approach and related work in Section 2.3. My approach is described in Section 2.4. The evaluation of the approach using various data sets is shown in Sections 2.5 and 2.6. Finally, I discuss the benefits and limitations of the current approach in Section 2.7, and conclude in Section 2.8.

2.2 Problem Formulation

Disaster response teams are increasingly utilizing Web-based planning tools to plan deployments [44]. Hundreds of responders access these tools to develop their plans using audio/video conferencing, text chat and annotatable maps. Rather than working with raw, natural language, my approach takes a structured form of the human dialogue from these Web-based planning tools as input. The goal of this chapter is to infer a human team's final plan from a processed form of the human team's planning conversation. In doing so, this work can be used to design an intelligent agent for these planning tools that can actively participate during a planning session to improve the team's decision.

This section describes the formal definition, input and output of the problem. Formally, this problem can be viewed as one of plan recognition, wherein the plan follows the formal representation of the Planning Domain Description Language (PDDL). The PDDL has been widely used in the planning research community and planning competitions (i.e., the International Planning Competition). A plan is *valid* if it achieves a user-specified goal state without violating user-specified plan constraints. Actions may be constrained to execute in sequence or in parallel with other actions. Other plan constraints can include discrete resource constraints (e.g. the presence of two medical teams) and temporal deadlines (e.g. a robot can only be deployed for up to 1 hour at a time due to battery life

constraints).

I assume that the team reaches agreement on a final plan. The techniques introduced by [82] can be used to detect the strength of this agreement, and to encourage the team to further discuss the plan to reach an agreement if necessary. Situations where a team agrees upon a flexible plan with multiple options to be explored will be included in future study. Also, while I assume that the team is more likely to agree on a valid plan, I do not rule out the possibility that the final plan is invalid.

2.2.1 Input

Text data from the human team conversation is collected in the form of utterances, where each utterance is one person's turn in the discussion, as shown in Table 2.1. The input to my approach is a machine-understandable form of human conversation data, as illustrated in the right-hand column of Table 2.1. This structured form captures the actions discussed and the proposed ordering relations among actions for each utterance.

Although I am not working with raw, natural language, this form of data still captures many of the characteristics that make plan inference based on human conversation challenging. Table 2.1 shows part of the data using the following shorthand:

ST = SendTo
rr = red robot, br = blue robot
rm = red medical, bm = blue medical
e.g. ST(br, A) = SendTo(blue robot, room A)

Utterance tagging

An utterance is tagged as an *ordered tuple of sets of grounded predicates*. Following a formal definition for first-order languages, a grounded predicate is an atomic formula whose argument terms are grounded (i.e., no free variables; all variables have an assigned value). In my case, a predicate represents an action applied to a set of objects (a crew member, robot, room, etc.), and an utterance can be represented as ordered sets of

	Natural dialogue	Structured form (ordered tuple of sets of grounded predicates)
U1	So I suggest using Red robot to cover “upper” rooms (A, B, C, D) and Blue robot to cover “lower” rooms (E, F, G, H).	({ST(rr,A),ST(br,E), ST(rr,B),ST(br,F), ST(rr,C),ST(br,G), ST(rr,D),ST(br,H)})
U2	Okay. so first send Red robot to B and Blue robot to G?	({ST(rr,B),ST(br,G)})
U3	Our order of inspection would be (B, C, D, A) for Red and then (G, F, E, H) for Blue.	({ST(rr,B),ST(br,G)}, {ST(rr,C),ST(br,F)}, {ST(rr,D),ST(br,E)}, {ST(rr,A), ST(br,H)})
U4	Oops I meant (B, D, C, A) for Red. ...	({ST(rr,B)},{ST(rr,D)}, {ST(rr,C)},{ST(rr,A)})
U5	So I can have medical crew go to B when robot is inspecting C ...	({ST(m,B), ST(r,C)})
U6	First, Red robot inspects B	({ST(r,B)})
U7	Yes, and then Red robot inspects D, Red medical crew to treat B	({ST(r,D),ST(rm,B)})

Table 2.1: Utterance tagging: Dialogue and structured form examples. (The structured form uses the following shorthand - ST: send to, rr: red robot, br: blue robot, rm: red medical, bm: blue medical, e.g. ST(br,A) : “send the blue robot to room A.”)

these actions. I only consider utterances related to plan formation; greetings and jokes, for example, are not tagged. Each set of grounded predicates represents a collection of actions that, according to the utterance, should happen simultaneously. The order of the sets of grounded predicates indicates the *relative* order in which these collections of actions should happen. For example, ($\{ST(rr, B), ST(br, G)\}, \{ST(rm, B)\}$) corresponds to sending the red robot to room B and the blue robot to room G simultaneously, followed by sending the red medical team to room B.

As indicated in Table 2.1, the structured dialogue still includes high levels of noise (i.e., dialogue contains a number of predicates that do not end up in the final plan). Each utterance (i.e. U1-U7) discusses a partial plan, and only predicates explicitly mentioned in the utterance are tagged (e.g. U6-U7: the “and then” in U7 implies a sequencing constraint with the predicate discussed in U6, but the structured form of U7 does not include $ST(r,B)$). Mistakes in utterances and misinformation are tagged without correction (e.g. B, C, D, A in U3), and any utterances indicating a need to revise information are not placed in context (e.g. U4). Utterances that clearly violate the ordering constraints (e.g. U1: all actions cannot happen at the same time) are also tagged without correction. In addition, information regarding whether an utterance was a suggestion, rejection of or agreement with a partial plan is not coded. The coding is done by four independent analysts, with each team planning session tagged and reviewed by two of these four analysts.

Note that the utterance tagging only contains information about *relative ordering* between the predicates appearing in that utterance, not the *absolute ordering* of their appearance in the final plan. For example, U2 specifies that the two grounded predicates happen at the same time. It does not state when the two predicates happen in the final plan, or whether other predicates will happen in parallel. This simulates how humans conversations often unfold — at each utterance, humans only observe the relative ordering, and infer the absolute order of predicates based on the whole conversation and an understanding of which orderings would make a valid plan. This utterance tagging scheme is also designed to support the future transition to automatic natural language processing. Automatic semantic role labeling [76], for example, can be used to detect the argu-

ments of predicates from sentences. One of the challenges with incorporating semantic role labeling into my system is that the dialogue from my experiments is often colloquial and key grammatical components of sentences are often omitted. Solving this problem and processing free-form human dialogue into more machine-understandable forms is an important research area, but I view that as a separate problem and do not focus on it in this thesis.

2.2.2 Output

The output of the approach is an inferred final plan, sampled from the probability distribution over the final plans. The final plan has the same representation the structured utterance tags (*ordered tuple of sets of grounded predicates*). The predicates in each set represent actions that should happen in parallel, and the ordering of sets indicates the sequence. Unlike the utterance tags, however, the sequence ordering relations in the final plan represent the *absolute* order in which the actions are to be carried out. An example of a plan is $(\{A_1, A_2\}, \{A_3\}, \{A_4, A_5, A_6\})$, where A_i represents a predicate. In this plan, A_1 and A_2 will happen at step 1 of the plan, A_3 happens at step 2 of the plan, and so on.

2.3 Approach in a Nutshell and Related Work

Planning conversations performed under time pressure exhibit unique characteristics and challenges for inferring the final plan. First, these planning conversations are succinct — participants tend to write quickly and briefly, and to be in a hurry to make a final decision. Second, there may be a number of different valid plans for the team's deployment — even with a simple scenario, people tend to generate a broad range of final plans. This represents the typical challenges faced during real rescue missions, where each incident is unique and participants cannot have a library of plans to choose from at each time. Third, these conversations are noisy — often, many suggestions are made and rejected more quickly than they would be in a more casual setting. In addition, there are not likely to be many repeated confirmations of agreement, which might typically ease detection of

Current time: 8pm.

Situation (Reference the map below):

Two fires started at 8pm: 1. a college dorm. 2. the theater building.
 Three street corners will be crime hot-spots (HS) from 8:30pm to 9pm.
 A street robbery is reported at 8pm. No injury occurred, but you need to talk to the victim.

What need to do:

Respond to as many incidents as possible given the resources below.

What you have:

Resource	Name	Function	Duration
Police Teams with robots	Alpha Bravo Charlie	Patrol Hotspot	Evacuation building in: 30 min with 1 robot 15 min with 2 robots 10 min with 3 robots (same for both dorm and Theater)
		or Deploy robot for evacuation or Respond to the street robbery	Talk to the victim in: 10 min with 1 police car
Fire Trucks	Delta Echo Foxtrot	Put out fire	Put out fire in: 30 min with 1 fire truck 15 min with 2 fire trucks 10 min with 3 fire trucks (same for both dorm and Theater)

Additional Information:

Fire incident

- Putting out a fire requires at least 1 firefighting team and 1 police car equipped with a special robot.
- Police cars also have robot operators in them, the operator has to stay with the robot until the evacuation is over.
- Only a robot can perform the evacuation. Firefighters and police men cannot.
- A robot can only be used once.
- Successfully responding to fire requires both evacuating people and putting out the fire. You need to do both, but they can happen simultaneously.

Crime Hotspots

- Responding to a hot spot patrol requires one police car to be located at the hotspot for a fully specified amount of time.

Street Robbery

- Only need 1 police car to respond to the street robbery

Others

- Priority of each task are intentionally not provided. Use your judgement.
- If there is no information about traffic, you can assume that travel time from places to places is negligible. If you are told there is traffic, you need to accommodate this in your plan.

Messages will be displayed here

Figure 2-1: Web-based tool developed and used for data collection

the agreed-upon plan.

It might seem natural to take a probabilistic approach to the plan inference problem, as we are working with noisy data. However, the combination of a small amount of noisy data and a large number of possible plans means that inference using typical, uninformative priors over plans may fail to converge to the team's final plan in a timely manner.

This problem could also be approached as a logical constraint problem of partial order planning, if there were no noise in the utterances: If the team were to discuss only partial plans relating to the final plan, without any errors or revisions, then a plan generator or scheduler [38] could produce the final plan using global sequencing. Unfortunately, data collected from human conversation is sufficiently noisy to preclude this approach.

These circumstances provided motivation for a combined approach, where I built a probabilistic generative model and used a logic-based plan validator [72] to compute a highly structured (i.e., nonuniform, more informative prior) prior distribution over possible plans. Intuitively speaking, this prior encodes my assumption that the final plan is likely, but not required, to be a valid plan. This approach naturally deals with both the noise in the data and the challenge of performing inference over plans with only a limited amount of data. I performed sampling inference in the model using Gibbs and Metropolis-Hastings sampling to approximate the posterior distribution over final plans, and empirical validation with human subject experiments indicated that the approach achieves 86% accuracy on average. More details of my model and inference methods are presented in Section 2.4.

The related work can be categorized into two categories: 1) application and 2) technique. In terms of application, my work relates to plan recognition (Section 2.3.1). In terms of technique, my approach relates to methods that combine logic and probability, though with different focused applications (Section 2.3.2).

2.3.1 Plan Recognition

Plan recognition has been an area of interest within many domains, including interactive software [117, 17, 95], story understanding [33] and natural language dialogue [25, 64].

The literature can be categorized in two ways. The first is in terms of requirements.

Some studies [94, 79] require a library of plans, while others [137, 115, 112, 118] replace this library with relevant structural information. If a library of plans is required, some studies [132, 78] assumed that this library can be collected, and that all future plans are guaranteed to be included within the collected library. By contrast, if a library of plans is not required, it can be replaced by related structure information, such as a domain theory or the possible set of actions performable by agents. The second categorization for the literature is in terms of technical approach. Some studies incorporated constraint-based approaches, while others took probabilistic or combination approaches.

First, I reviewed work that treated plan recognition as a knowledge base search problem. This method assumes that you either have or can build a knowledge base, and that your goal is to efficiently search this knowledge base [94, 79]. This approach often includes strong assumptions regarding the correctness and completeness of the plan library, in addition to restrictions on noisy data [132, 78], and is applicable in domains where the same plan reoccurs naturally. For example, Gal et al. [51] studied how to adaptively adjust educational content for a better learning experience, given students' misconceptions, using a computer-based tutoring tool. Similarly, Brown et al. [24] investigated users' underlying misconceptions using user data collected from multiple sessions spent trying to achieve the same goal. In terms of technical approach, the above approaches used logical methods to solve the ordering constraints problem of searching the plan library.

I also reviewed work that replaced a knowledge base with the structural information of the planning problem. Zhou et al. [137] replaced the knowledge base with action models of the domain, and formulated the problem as one of satisfiability to recognize multi-agent plans. A similar approach was taken in Ramirez et al. [115], wherein action models were used to replace the plan library, while Pynadath et al. [112] incorporated an extension of probabilistic context free grammars (PCFGs) to encode a set of predefined actions to improve efficiency. More recently, Markov logic was applied to model the geometry, motion model and rules for the recognition of multi-agent plans while playing a game of capture the flag [118]. Replacing the knowledge base with structural information reduces the amount of prior information required. However, there are two major issues with the application of prior work to recognize plans from team conversation: First, the above

work assumed some repetition of previous plans. For example, learning the weights in Markov logic (which represent the importance or strictness of the constraints) requires prior data from the same mission, with the same conditions and resources. Second, using first-order logic to express plan constraints quickly becomes computationally intractable as the complexity of a plan increases.

In contrast to logical approaches, probabilistic approaches allow for noisy observations. Probabilistic models are used to predict a user's next action, given noisy data [4, 71]. These works use actions that are *normally* performed by users as training data. However, the above approaches do not consider particular actions (e.g., actions that must be performed by users to achieve certain goals within a software system) to be more likely. In other words, while they can deal with noisy data, they do not incorporate structural information that could perhaps guide the plan recognition algorithm. An additional limitation of these methods is that they assume predefined domains. By defining a domain, the set of possible plans is limited, but the possible plans for time-critical missions is generally not a limited set. The situation and available resources for each incident are likely to be different. A method that can recognize a plan from noisy observations, and from an open set of possible plans, is required.

Some probabilistic approaches incorporate structure through the format of a plan library. Pynadath and Wellman [112] represented plan libraries as probabilistic context free grammars (PCFGs). Then this was used to build Bayes networks that modeled the underlying generative process of plan construction. However, their parsing-based approaches did not deal with partially-ordered plans or temporally interleaved plans. Geib et al. [53, 52] overcame this issue by working directly with the plan representation without generating an intermediate representation in the form of a belief network. At each time step, their technique observed the previous action of the agent and generated a pending action set. This approach, too, assumed an existing plan library and relied on the domains with some repetition of previous plans. More recently, Nguyen et al. [103] introduced techniques to address incomplete knowledge of the plan library, but for plan generation rather than plan recognition applications.

My approach combines a probabilistic approach with logic-based prior to infer team

plans without the need for historical data, using only situational information and data from a single planning session. The situational information includes the operators and resources from the domain and problem specifications, which may be updated or modified from one scenario to another. I do not require the development or addition of a plan library to infer the plan, and demonstrate my solution is robust to incomplete knowledge of the planning problem.

2.3.2 Combining Logic and Probability

The combination of a logical approach with probabilistic modeling has gained interest in recent years. Getoor et al. [55] introduced a language for the description of statistical models over typed relational domains, and demonstrated model learning using noisy and uncertain real-world data. Poon et al. [108] proposed statistical sampling to improve searching efficiency for satisfiability testing. In particular, the combination of first-order logic and probability, often referred to as Markov Logic Networks (MLN), was studied. MLN forms the joint distribution of a probabilistic graphical model by weighting formulas in a first-order logic [116, 121, 109, 113].

My approach shares with MLNs the philosophy of combining logical tools with probabilistic modeling. MLNs utilize first-order logic to express relationships among objects. General first-order logic allows for the use of expressive constraints across various applications. However, within the planning domain, enumerating all constraints in first-order logic quickly becomes intractable as the complexity of a plan increases. For example, first-order logic does not allow the explicit expression of action preconditions and postconditions, let alone constraints among actions. PDDL has been well-studied in the planning research community [96], where the main focus is to develop efficient ways to express and solve planning problems. My approach exploits this tool to build a highly structured planning domain within the probabilistic generative model framework.

2.4 My Approach

This section presents the details of my approach. I describe my probabilistic generative model and indicate how this model is combined with the logic-based prior to perform efficient inference. The generative model specifies a joint probability distribution over observed variables (e.g., human team planning conversation) and latent variables (e.g., the final plan); my model learns the distribution of the team's final plan, while incorporating a logic-based prior (plan validation tool). My key contribution is the design of this generative model with logic-based prior. I also derive the Gibbs sampling [12] representation and design the scheme for applying Metropolis-Hasting sampling [97] for performing inference on this model.

2.4.1 Generative Model

I model the human team planning process, represented by their dialogue, as a probabilistic Bayesian model. In particular, I utilize a probabilistic generative modeling approach that has been extensively used in topic modeling (e.g., [23]).

We start with a *plan* latent variable that must be inferred through observation of utterances made during the planning session. The model generates each utterance in the conversation by sampling a subset of the predicates in *plan* and computing the relative ordering in which they appear within the utterance. The mapping from the absolute ordering in *plan* to the relative ordering of predicates in an utterance is described in more detail below. Since the conversation is short and the level of noise is high, my model does not distinguish utterances based on the order in which they appear during the conversation. This assumption produces a simple yet powerful model, simplifies the inference steps. However, the model can also be generalized to take the ordering into account with a simple extension. I include further discussion on this assumption and the extension in Section 2.7.4.

The following is a step-by-step description of the generative model:

1. **Variable *plan*:** The *plan* variable in Figure 2-2 is defined as an ordered tuple of sets of grounded predicates, and represents the final plan agreed upon by the team.

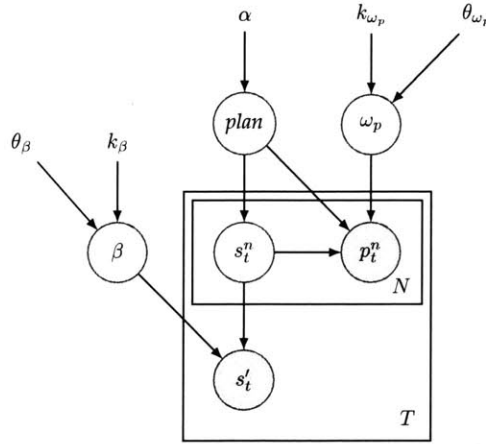


Figure 2-2: Graphical model representation of the generative model. The *plan* latent variable represents the final agreed-upon plan. The p_t^n variable represents the n^{th} predicate of t^{th} utterance, while s_t^n represents the absolute ordering of that predicate in the *plan*. The s'_t represents the relative ordering of s_t^n within the utterance t . The latent variable ω_p represents the noisiness of predicates, and β represents the noisiness of the ordering.

It is distributed over the space of ordered tuples of sets of grounded predicates. I assume that the total number of grounded predicates in one domain is fixed.

The prior distribution over the *plan* variable is given by:

$$p(\text{plan}) \propto \begin{cases} e^\alpha & \text{if the plan is valid} \\ 1 & \text{if the plan is invalid.} \end{cases} \quad (2.1)$$

where α is a positive number. This models my assumption that the final plan is more likely, but is not necessarily required, to be a valid plan.

The likelihood of the *plan* is defined as:

$$\begin{aligned} p(s, p | \text{plan}, \omega_p) &\sim \prod_t \prod_n p(s_t^n, p_t^n | \text{plan}, \omega_p) \\ &\sim \prod_t \prod_n p(p_t^n | \text{plan}, s_t^n, \omega, p) p(s_t^n | \text{plan}) \end{aligned}$$

Each set of predicates in $plan$ is assigned a consecutive absolute plan step index s , starting at $s = 1$ working from left to right in the ordered tuple. For example, given $plan = (\{A_1, A_2\}, \{A_3\}, \{A_4, A_5, A_6\})$, where each A_i is a predicate, A_1 and A_2 occur in parallel at plan step $s = 1$, and A_6 occurs at plan step $s = 3$.

2. **Variable s_t^n :** s_t^n represents a step index (i.e., absolute ordering) of the n th predicate in the t th utterance in the $plan$. A step index represents an *absolute* timestamp of the predicate in the plan. In other words, s_t^n indicates the absolute order of the predicate p_t^n as it appears in $plan$. I use $s_t = \{s_t^1, s_t^2 \dots\}$ to represent a vector of orderings for the t th utterance, where the vector s_t may not be a set of consecutive numbers.

s_t^n is sampled as follows: For each utterance, n predicates are sampled from $plan$. For example, consider $n = 2$, where the first sampled predicate appears in the second set (i.e., the second timestamp) of $plan$ and the second sampled predicate appears in the fourth set. Under these conditions, $s_t^1 = 2$ and $s_t^2 = 4$. The probability of a set being sampled is proportional to the number of predicates contained within that set. For example, given $plan = (\{A_1, A_2\}, \{A_3\}, \{A_4, A_5, A_6\})$, the probability of selecting the first set ($\{A_1, A_2\}$) is $\frac{2}{6}$. This models the notion that people are more likely to discuss plan steps that include many predicates, since plan steps with many actions may require more effort to elaborate. Formally:

$$p(s_t^n = i | plan) = \frac{\# \text{ predicates in set } i \text{ in plan}}{\text{total } \# \text{ of predicates in plan}}. \quad (2.2)$$

The likelihood of s_t is defined as:

$$\begin{aligned} p(s_t | p_t, s'_t) &\sim p(s_t | plan) p(p_t, s'_t | s_t, \beta, \omega_p, plan) \\ &\sim p(s'_t | s_t, \beta) \prod_n p(s_t^n | plan) p(p_t^n | plan, s_t^n, \omega_p) \end{aligned}$$

3. **Variable s'_t and β :** The variable s'_t is an array of size n , where $s'_t = \{s'_t{}^1, s'_t{}^2 \dots s'_t{}^n\}$. The $s'_t{}^n$ random variable represents the relative ordering of the n th predicate within the t th utterance in the *plan*, with respect to other grounded predicates appearing in the t th utterance.

s'_t is generated from s_t as follows:

$$p(s'_t|s_t) \propto \begin{cases} e^\beta & \text{if } s'_t = f(s_t) \\ 1 & \text{if } s'_t \neq f(s_t). \end{cases} \quad (2.3)$$

where $\beta > 0$. The β random variable is a hyper-parameter that represents the noisiness of the ordering of grounded predicates appearing throughout the entire conversation. It takes a scalar value, and is sampled from gamma distribution:

$$p(\beta|k_\beta, \theta_\beta) \sim \text{Gamma}(k_\beta, \theta_\beta), \quad (2.4)$$

where both k_β and θ_β are set to 10.

The function f is a deterministic mapping from the absolute ordering s_t to the relative ordering s'_t . f takes a vector of absolute plan step indices as input, and produces a vector of consecutive indices. For example, f maps $s_t = (2, 4)$ to $s'_t = (1, 2)$ and $s_t = (5, 7, 2)$ to $s'_t = (2, 3, 1)$.

This variable models the way predicates and their orders appear during human conversation: People frequently use relative terms, such as “before” and “after,” to describe partial sequences of a full plan, and do not often refer to absolute ordering. People also make mistakes or otherwise incorrectly specify an ordering. My model allows for inconsistent relative orderings with nonzero probability; these types of mistakes are modeled by the value of β .

4. **Variable p_t^n and ω_p :** The variable p_t^n represents the n th predicate appearing in the t th utterance. The absolute ordering of this grounded predicate is s_t^n . The p_t^n is sampled given s_t^n , the *plan* variable and a parameter, ω_p .

The ω_p random variable (hyper-parameter) represents the noisiness of grounded predicates appearing throughout the entire conversation. It takes a scalar value, and is sampled from beta distribution:

$$p(\omega_p | k_{\omega_p}, \theta_{\omega_p}) \sim \text{beta}(k_{\omega_p}, \theta_{\omega_p}), \quad (2.5)$$

where k_{ω_p} is set to 40, and θ_{ω_p} is set to 10.

With probability ω_p , I sample the predicate p_t^n uniformly with replacement from the “correct” set s_t^n in *plan* as follows:

$$p(p_t^n = i | \text{plan}, s_t^n = j) = \begin{cases} \frac{1}{\# \text{ pred. in set } j} & \text{if } i \text{ is in set } j \\ 0 & \text{o.w.} \end{cases}$$

With probability $1 - \omega_p$, I sample the predicate p_t^n uniformly with replacement from “any” set in *plan* (i.e., from all predicates mentioned in the dialogue). Therefore:

$$p(p_t^n = i | \text{plan}, s_t^n = j) = \frac{1}{\text{total } \# \text{ predicates}}. \quad (2.6)$$

In other words, with higher probability ω_p , I sample a value for p_t^n that is consistent with s_t^n but allows for nonzero probability that p_t^n is sampled from a random plan. This allows the model to incorporate noise during the planning conversation, including mistakes or plans that are later revised.

2.4.2 Plan validation tool

I use the Planning Domain Description Language (PDDL) 2.1 plan validation tool [72] to evaluate the prior distribution over possible plans. In this section, I briefly review the PDDL, and the plan validation tool that is used to form the prior in Equation 2.1.

Planning Domain Description Language

The Planning Domain Description Language (PDDL) [96] is a standard planning language, inspired by the Stanford Research Institute Problem Solver (STRIPS) [48] and Action Description Language (ADL) [106], and is now utilized in the International Planning Competition.

A PDDL model of a planning problem has two major components: a domain specification and a problem specification. The domain description consists of a domain-name definition, requirements on language expressivity, definition of object types, definition of constant objects, definition of predicates (i.e. templates for logical facts), and the definition of possible actions that are instantiated during execution. Actions have parameters that may be instantiated with objects, preconditions, and conditional or unconditional effects. An excerpt of the PDDL domain file used in this work, called the RESCUE domain, is shown below. For example, the predicate `isSaved` encodes the logical fact of whether or not a particular patient has been rescued, and the action `SEND-ROBOT` is instantiated during execution to send a particular robot to particular location.


```

(define (domain RESCUE)
  (:requirements :typing :durative-actions :negative-preconditions)
  (:types patient valve - thingsToFix location - location
    med-crew mechanic robot - resource)
  (:predicates
    (isAt ?p - thingsToFix ?l - location)
    (isSaved ?p - patient)
    (isFixed ?v - valve)
    (isInspected ?l - location)
    (isAvail ?r - resource)
  )
  (:durative-action SEND-ROBOT
    :parameters (?r - robot ?l - location)
    :duration (= ?duration 1)
    :condition (and
      (at start (isAvail ?r))
      (at start (not (isInspected ?l)))
    )
    :effect (and
      (at start (not (isAvail ?r) ) )
      (at end (isAvail ?r))
      (at end (isInspected ?l))
    )
  )
  ...)

```

The problem specification consists of a problem-name definition, the definition of the related domain-name, the definition of all the possible objects relevant to the planning problem, the initial state of the planning environment as a conjunction of true/false facts, and the definition of goal-states as a logical expression over true/false facts. An excerpt of the PDDL problem specification file used in this work is shown below. The 'init' section describes initial conditions — for example, patient pB is initially situated at location B, and patient pD is at D. The 'goal' section indicates the desired final state — in this case all rooms must be inspected, all patients must be rescued, and all valves fixed. The 'metric' section defines the metric that the planner optimizes when producing a plan.

```

(define (problem rescuepeople)
  (:domain RESCUE)
  (:objects
    pB pD pG -- patient
    vC vF -- valve
    A B C D E F G H -- location
    redMed blueMed -- med-crew
    redR blueR -- robot
    mech1 -- mechanic)
  (:init
    (isAt pB B)
    (isAt pD D)
    ...
    (isAvail redMed)
    (isAvail blueMed)
    (isAvail redR)
    (isAvail blueR)
    (isAvail mech1)
    (not (isSaved pB))
    ...
    (not (isInspected A))
    (not (isInspected B))
    ...
    (not (isFixed vC))
    (not (isFixed vF))
  )
  (:goal (and
    (isSaved pB)
    (isSaved pD)
    ...
    (isFixed vC)
    (isFixed vF)
    (isInspected A)
    (isInspected B)
    ...
  )
  )
  ;(:metric minimize (total-time))
)

```

For my work, I note that domain specification could be reused from one planning session to another if the capabilities of a team do not change. For example, once the set of possible set of actions is defined in domain specification, it may be sufficient to merely modify the number and names of locations, medical crews, or robots in the problem specification. The domain and the problem specification files represent the only 'prior knowledge' that my approach requires in order to infer the final agreed-upon plan. A valid plan is defined as a totally or partially ordered sequence of grounded predicates that achieves the goal state from the initial state, without violating any constraints. Otherwise, the plan is invalid. The next section describes a plan validation tool that assesses the validity of a given plan, given the domain and problem specification files.

Plan Validation Tool (VAL)

The plan validator is a standard tool that takes as input a planning problem described in PDDL and a proposed solution plan. The tool incorporates three input files: 1) a domain definition file, 2) a problem definition file and 3) a proposed solution plan file. The domain definition file contains types of parameters (e.g., resources, locations), predicate definitions and actions (which also have parameters, conditions and effects). The problem definition file contains information specific to the situation: For example, the number of locations and victims, initial goal conditions and a metric to optimize. A proposed solution plan file contains a single complete plan, described in PDDL. The output of a plan validation tool indicates whether the proposed solution plan is valid (true) or not (false).

Metrics represent ways to compute a plan quality value. For the purpose of this study, the metrics used included: 1) the minimization of total execution time for the radioactive material leakage scenario, and 2) the maximization of the number of incidents responded to in the police incident response scenario. Intuitively speaking, metrics are intended to reflect the preference of human experts. It is natural to assume that human experts would try to minimize the total time to completion of time-critical missions (such as in the radioactive material leakage scenario). If first responders cannot accomplish all the necessary tasks in a scenario due to limited availability of resources, they would most likely try to maximize the number of completed tasks (such as in the police incident response

scenario).

One could imagine that these input files could be reused in subsequent missions, as the capabilities (actions) of a team may not change dramatically. However, the number of available resources might vary, or there might be rules implicit in a specific situation that are not encoded in these files (e.g., to save endangered humans first before fixing a damaged bridge). In Section 2.6, I demonstrate the robustness of my approach using both complete and degraded PDDL plan specifications.

The computation of a plan's validity is generally cheaper than that of a valid plan generation. The plan validator gives us a way to compute $p(plan)$ (defined in Section 2.4.1) up to proportionality in a computationally efficient manner. Leveraging this efficiency, I use Metropolis-Hastings sampling, (details described in Section 2.4.3) without calculating the partition function.

2.4.3 Gibbs Sampling

I use Gibbs sampling to perform inference on the generative model. There are four latent variables to sample: $plan$, the collection of variables s_t^n , ω_p and β . I iterate between sampling each latent variable, given all other variables. The PDDL validator is used when the $plan$ variable is sampled.

Sampling plan using Metropolis-Hastings

Unlike s_t^n , where I can write down an analytic form to sample from the posterior, it is intractable to directly resample the $plan$ variable, as doing so would require calculating the number of all possible plans, both valid and invalid. Therefore, I use a Metropolis-Hasting (MH) algorithm to sample from the $plan$ posterior distribution within the Gibbs sampling steps.

The posterior of $plan$ can be represented as the product of the prior and likelihood,

as follows:

$$\begin{aligned}
p(plan|s, p) &\propto p(plan)p(s, p|plan) \\
&= p(plan) \prod_{t=1}^T \prod_{n=1}^N p(s_t^n, p_t^n|plan) \\
&= p(plan) \prod_{t=1}^T \prod_{n=1}^N p(s_t^n|plan)p(p_t^n|plan, s_t^n) \tag{2.7}
\end{aligned}$$

The MH sampling algorithm is widely used to sample from a distribution when direct sampling is difficult. This algorithm allows us to sample from posterior distribution according to the user-specified proposal distribution without having to calculate the partition function. The typical MH algorithm defines a proposal distribution, $Q(x'|x_t)$, which samples a new point (i.e., x' : a value of the *plan* variable in my case) given the current point x_t . The new point can be achieved by randomly selecting one of several possible moves, as defined below. The proposed point is then accepted or rejected, with a probability of $\min(1, \text{acceptance ratio})$.

Unlike simple cases, where a Gaussian distribution can be used as a proposal distribution, my distribution needs to be defined over the plan space. Recall that *plan* is represented as an ordered tuple of sets of predicates. In this work, the new point (i.e., a candidate plan) is generated by performing one of the following moves on the current plan:

- Move to next: Randomly select a predicate that is in the current plan, and move it to the next timestamp. If it is in the last timestamp in the plan, move it to the first timestamp.
- Move to previous: Randomly select a predicate that is in the current plan, and move it to the previous timestamp. If it is in the first timestamp in the plan, move it to the last timestamp.
- Add a predicate to plan: Randomly select a predicate that is not in the current plan, and randomly choose one timestamp in the current plan. Add the predicate to the chosen timestamp.

- Remove a predicate from plan: Randomly select and remove a predicate that is in the current plan.

These moves are sufficient to allow for movement from one arbitrary plan to another. The intuition behind designing this proposal distribution is described in Section 2.7.5.

Note that the proposal distribution, as it is, is not symmetrical — $Q(x'|x_t) \neq Q(x_t|x')$. We need to compensate for that according to the following,

$$p^*(x')Q(x'|x_t) = p^*(x)Q(x_t|x'), \quad (2.8)$$

where p^* is the target distribution. This can be done simply by counting the number of moves possible from x' to get to x , and from x to get to x' , and weighing the acceptance ratio such that Equation 2.8 is true. This is often referred to as Hastings correction, which is performed to ensure that the proposal distribution does not favor some states over others.

Next, the ratios of the proposal distribution at the current and proposed points are calculated. When $plan$ is valid, $p(plan)$ is proportional to e^α , and when $plan$ is invalid, it is proportional to 1, as described in Equation 2.1. Plan validity is calculated using the plan validation tool. The remaining term, $p(s_t^n | plan)p(p_t^n | plan, s_t^n)$, is calculated using Equations 2.2 and 2.6.

Then, the proposed $plan$ is accepted with the following probability: $\min\left(1, \frac{p^*(plan=x'|s,p)}{p^*(plan=x_t|s,p)}\right)$, where p^* is a function proportional to the posterior distribution.

Although I chose to incorporate MH, it is not the only usable sampling method. Any other method that does not require calculation of the normalization constant (e.g., rejection sampling or slice sampling) could also be used. However, for some methods, sampling from the ordered tuple of sets of grounded predicates can be slow and complicated, as pointed out by Neal et al. [100].

Sampling hyper-parameters β and ω_p with slice sampling

I use slice sampling to sample both β and ω_p . This method is simple to implement and works well with scalar variables. Distribution choices are made based on the valid value each can take. β can take any value, preferably with one mode, while ω_p can only take a

value between $[0, 1]$. MH sampling may also work; however, this method could be overly complicated for a simple scalar value. I chose the stepping out procedure, as described by Neal et al [100].

Sampling s_t^n

Fortunately, an analytic expression exists for the posterior of s_t^n :

$$\begin{aligned} p(s_t|plan, p_t, s'_t) &\propto p(s_t|plan)p(p_t, s'_t|plan, s_t) \\ &= p(s_t|plan)p(p_t|plan, s_t)p(s'_t|s_t) \\ &= p(s'_t|s_t) \prod_{n=1}^N p(s_t^n|plan)p(p_t^n|plan, s_t^n) \end{aligned}$$

Note that this analytic expression can be expensive to evaluate if the number of possible values of s_t^n is large. In that case, one can marginalize s_t^n , as the variable we truly care about is the *plan* variable.

2.5 Experimentation

2.5.1 Web-based Tool Design

Disaster response teams are increasingly using Web-based tools to coordinate missions and share situational awareness. One of the tools currently used by first responders is the Next Generation Incident Command System (NICS) [44]. This integrated sensing and command-and-control system enables the distribution of large-scale coordination across multiple jurisdictions and agencies. It provides video and audio conferencing capabilities, drawing tools and a chat window, and allows for the sharing of maps and resource information. Overall, the NICS enables the collection and exchange of information critical to mission planning.

I designed a Web-based collaboration tool modeled after this system, with a modification that requires the team to communicate solely via text chat. This tool was developed using Django [70], a free and open-source Web application framework written in Python.

Django is designed to ease working with heavy-duty data, and provides a Python API to enable rapid prototyping and testing. Incoming data can be easily maintained through a user-friendly administrative interface. Although it is a simplified version of the NICS, it provides the essence of the emerging technology for large-scale disaster coordination (Figure 2-1).

2.5.2 Scenarios

Human subjects were given one of two fictional rescue scenarios and asked to formulate a plan by collaborating with their partners. I collected human team planning data from the resulting conversations (i.e., the user input to the web-based tool), and used this data to validate my approach. The first scenario involves a radioactive material leakage accident in a building with multiple rooms, where all tasks (described below) were assumed to take one unit of time. I added complexity to the scenario by announcing a new piece of information halfway through the planning conversation, requiring the team to change their plan. The second scenario also included time-durative actions (e.g., action A can only take place if action B is taking place). These scenarios are inspired by those described in emergency response team training manuals [46], and are designed to be completed in the reasonable time for my experiments.

First scenario: radioactive material leakage

This disaster scenario involves the leakage of radioactive material on a floor consisting of eight rooms. Each room contains either a patient requiring in-person assessment or a valve that must be repaired (Figure 2-4).

Goal State: All patients are assessed in-person by a medical crew. All valves are fixed by a mechanic. All rooms are inspected by a robot.

Constraints: There are two medical crews, red and blue (discrete resource constraint), one human mechanic (discrete resource constraint) and two robots, red and blue (discrete

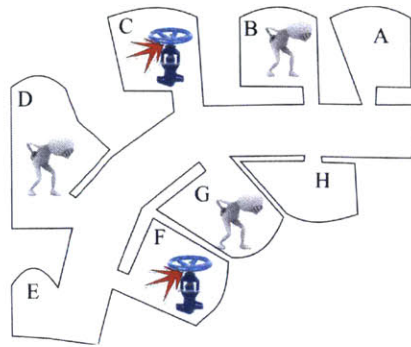


Figure 2-3: Radioactive material leakage scenario

resource constraint). For safety purposes, a robot must inspect the radioactivity of a room before human crews can be sent inside (sequence constraint).

Assumption: All tasks (e.g. inspecting a room, fixing a valve) take the same amount of time (one unit), and there are no hard temporal constraints. This assumption was made to conduct the initial proof-of-concept experimentation described in this thesis, and is relaxed in the scenario described in Section 2.5.2.

Announcement: During the planning session, the team receives a situation update that the red robot is now out of order, requiring the team to modify their previously discussed plan to only use one robot for deployment. The announcement triggers automatically once the team has exchanged 20 utterances. The purpose of this announcement is to increase task complexity for the team, to have at least two competing plans and to increase the level of noise in the conversation.

This scenario produces a large number of possible plans (more than 10^{12}), many of which are valid for achieving the goals without violating the constraints.

Second scenario: police incidents response

The second scenario involves a team of police officers and firefighters responding to a series of incidents occurring in different time frames. This scenario includes more complicated

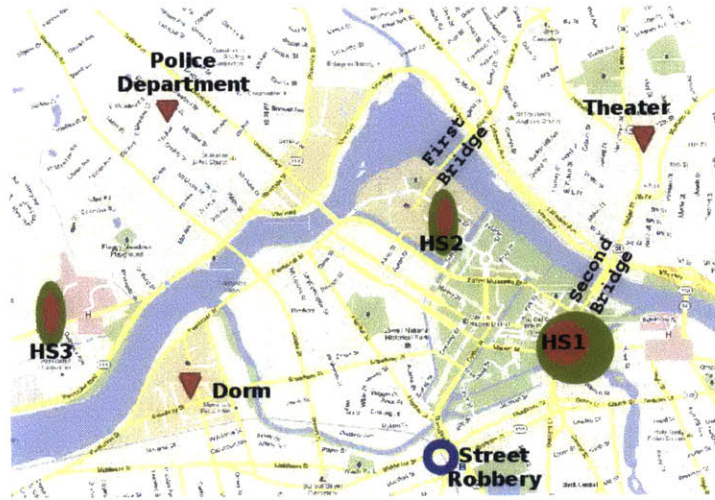


Figure 2-4: Police incident response scenario

time-durative actions than the first, as well as interdependency of tasks that has to be taken into account when planning. The current time is given as 8 p.m. Two fires have started at this time: one at a college dorm and another at a theater building, as shown in Figure 2-4. Also, three street corners, indicated as crime hot-spots (places predicted to experience serious crimes, based on prior data), become active between 8:30 p.m. and 9 p.m. There is also a report of a street robbery taking place at 8 p.m. No injury has occurred; however, a police officer must speak with the victim to file an incident report.

Goal State: Respond to as many incidents as possible given the resources listed in Table 2.2.

Constraints:

- Putting out a fire requires one fire truck and one police car equipped with a robot.
- A police car must stay with the robot until an evacuation is over.
- Only a robot can perform an evacuation.
- Each robot can only be used once.
- Successfully responding to a fire requires both evacuating the building and putting out the fire. Both actions can happen simultaneously.

Resources	Name	Function	Duration
Police teams with robots	Alpha Bravo Charlie	Patrol hotspot	Evacuate one building in: 30 min with one robot 15 min with two robots 10 min with three robots
		Deploy robot for evacuation	
		Respond to the street robbery	Talk to the victim in: 10 min with one police car
Fire trucks	Delta Echo Foxtrot	Put out fire	Put out fire in: 30 min with one fire truck 15 min with two fire trucks 10 min with three fire trucks (same for both dorm and theater)

Table 2.2: Resources available in police incident response scenario

- Responding to a hot-spot patrol requires one police car to be located at the site for a specified amount of time.
- Only one police car is necessary to respond to the street robbery.

Assumption and announcement: If no information about traffic is provided, the travel time from place to place is assumed to be negligible. During the planning session, the team receives the following announcement: “The traffic officer just contacted us, and said the First and Second bridges will experience heavy traffic at 8:15 pm. It will take at least 20 minutes for any car to get across a bridge. The travel time from the theater to any hot-spot is about 20 minutes without using the bridges.” Once this announcement is made, the team must account for the traffic in their plan.

2.6 Evaluation

In this section, I evaluate the performance of my approach for plan inference through initial proof-of-concept human subject experimentation, and show I am able to infer a human team’s final plan with 86% accuracy on average, where “accuracy” is defined as a composite measure of task allocation and plan sequence accuracy measures. The task allocation represents assignments of actions to agents, while the sequence represents the order in which the tasks are executed. I also describe a robot demonstration in which two

people plan and execute a first-response collaborative task with a PR2 robot.

2.6.1 Human Team Planning Data

As indicated previously, I designed a Web-based collaboration tool modeled after the NICS system [44] used by first-response teams, but with a modification that requires the team to communicate solely via text chat. For the radioactive material leakage scenario, before announcement, 13 teams of two (a total of 26 participants) were recruited through Amazon Mechanical Turk and from the greater Boston area. Recruitment was restricted to those located in the US to increase the probability that participants were fluent in English. For the radioactive material leakage scenario, after announcement, 21 teams of two (a total of 42 participants) were recruited through Amazon Mechanical Turk and from the greater Boston area. For the police incident response scenario, 14 teams of two (total 28 participants) were recruited from the greater Boston area. Participants were not required to have prior experience or expertise in emergency or disaster planning, and I note that there may be structural differences in the dialog of expert and novice planners. I leave this topic for future investigation.

Each team received one of the two fictional rescue scenarios described in Section 2.5.2, and was asked to collaboratively plan a rescue mission. Upon completion of the planning session, each participant was asked to summarize the final agreed-upon plan in the structured form described previously. An independent analyst reviewed the planning sessions to resolve discrepancies between the two members' descriptions when necessary.

Four independent analysts performed utterance tagging, with each team planning session tagged and reviewed by two of these four analysts. The resulting tagged utterances shows that 36% of predicates mentioned per data set did not end up in the final plan on average.

2.6.2 Method Implementation

This work is implemented in Python, and the VAL PDDL 2.1 plan validator [72] was used. I performed 2,000 Gibbs sampling steps on the data from each planning session. The

initial *plan* value was set to two to five moves (from MH proposal distribution) away from the true plan. The initial value for *s* variable was randomly set to any timestamp in the initial *plan* value.

Within one Gibbs sampling step, I performed 30 steps of the Metropolis-Hastings (MH) algorithm to sample the plan. Every 20 samples were selected to measure accuracy (median), after a burn-in period of 200 samples.

Results I assessed the quality of the final plan produced by my approach in terms of the accuracy of task allocation among agents (e.g. which medic travels to which room) and the accuracy of the plan sequence.

Two metrics for task allocation accuracy were evaluated: 1) The percent of inferred plan predicates appearing in the team's final plan [% Inferred], and 2) the percent noise rejection of extraneous predicates that were discussed but do not appear in the team's final plan [% Noise Rej].

I evaluated the accuracy of the plan sequence as follows: A pair of predicates is *correctly ordered* if it is consistent with the relative ordering in the true final plan. I measured the percent accuracy of sequencing [% Seq] by $\frac{\# \text{ correctly ordered pairs of correct predicates}}{\text{total \# of pairs of correct predicates}}$. Only correctly estimated predicates were compared, as there is no ground truth relation for predicates not included in the true final plan. I used this relative sequencing measure because it does not compound sequence errors, as an absolute difference measure would (e.g. where an error in the ordering of one predicate early in the plan shifts the position of all subsequent predicates).

Overall "composite" plan accuracy was computed as the arithmetic mean of the task allocation and plan sequence accuracy measures. This metric summarizes the two relevant accuracy measures so as to provide a single metric for comparison between conditions.

I evaluated my approach under four conditions: 1) perfect PDDL files, 2) PDDL problem file with missing goals/constants (e.g. delete available agents), 3) PDDL domain file missing a constraint (e.g. delete precondition), and 4) using an uninformative prior over possible plans.

The purpose of the second condition, PDDL problem file with missing goals/constants, was to test the robustness of my approach to incomplete problem information. This PDDL

problem specification was intentionally designed to omit information regarding one patient (pG) and one robot (blueR). It also omitted the following facts about the initial state: that pG was located at G, the blueR was available to perform inspections, and patient pG patient was not yet rescued. The goal state omitted that pG patient was to be rescued. This condition represented a significant degradation of the problem definition file, since the original planning problem involved only three patients and two robots.

The purpose of the third condition, PDDL domain file with a missing constant, was to test the robustness of my approach to missing constraints (or rules for successful execution). It is potentially easy for a person to miss specifying a rule that is often implicitly assumed. The third condition omitted the following constraint from the domain file: all the rooms are to be inspected prior to sending any medical crews. This condition represented a significant degradation of the domain file, since this constraint affected any action involving one of the medical crew teams.

Results shown in Tables 2.3-2.5 are produced by sampling *plan* and *s* variables and fixing $\beta = 5$ and $\omega_p = 0.8$. The tables report median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. I show that my approach infers final plans with greater than 86% composite accuracy on average. I also show that my approach is relatively robust to degraded PDDL specifications (i.e., PDDL with missing goals, constants and constraints). Further discussion of sampling hyper-parameters is found in Section 2.7.2.

2.6.3 Concept-of-Operations Robot Demonstration

I illustrate the use of my approach for plan inference through a robot demonstration in which two people plan and execute a first-response collaborative task with a PR2 robot. The participants plan an impending deployment using the Web-based collaborative tool I developed. Once the planning session is complete, the dialogue is tagged manually. The plan inferred from this data is confirmed with the human planners and provided to the robot for execution. The registration of predicates to robot actions, and room names to map locations, is performed offline in advance. While the first responders are on their

	Task Allocation		% Seq.	Composite % Acc.
	% Inferred	% Noise Rej.		
PDDL	61	100	97	86
PDDL with missing goals and constants	100	58	77	78
PDDL with missing constraint	70	100	87	86
No PDDL	70	58	66	65

Table 2.3: Radioactive material leakage scenario plan accuracy results, before announcement (13 teams / 26 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.

	Task Allocation		% Seq.	Composite % Acc.
	% Inferred	% Noise Rej.		
PDDL	77	100	83	87
PDDL with missing goals and constants	100	54	97	84
PDDL with missing constraint	72	100	90	87
No PDDL	100	54	81	78

Table 2.4: Radioactive material leakage scenario plan accuracy results, after announcement (21 teams / 42 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.

	Task Allocation		% Seq.	Composite % Acc.
	% Inferred	% Noise Rej.		
PDDL	97	89	97	86
PDDL with missing goals and constants	92	86	92	83
PDDL with missing constraint	97	89	97	85
No PDDL	81	95	81	82

Table 2.5: Police incidents response scenario plan accuracy results (14 teams / 28 subjects). The table reports median values for the percent of the inferred plan predicates appearing in the final plan [% Inferred], noise rejection [% Noise Rej.], and sequence accuracy [% Seq.]. Composite % Accuracy is calculated as the average of the previous three measures.

way to the accident scene, the PR2 autonomously navigates to each room, performing online localization, path planning and obstacle avoidance. The robot informs the rest of the team as it inspects each room and confirms it is safe for human team members to enter. Video of this demo can be found here: <http://tiny.cc/uxhcrw>.

2.7 Discussion

In this section I discuss the results and trends in Tables 2.3-2.5. I then discuss how sampling hyper-parameters improves inference accuracy, and provide an interpretation of inferred hyper-parameter values and how they relate to data characteristics. I also provide additional support for the use of PDDL by analyzing multiple Gibbs sampling runs. My rationale behind the i.i.d assumption on utterances made in the generative model is explained, and I show how a simple extension to my model can relax this assumption. Finally, I provide my rationale for designing the proposal distribution for the sampling algorithm.

2.7.1 Results

The average accuracy of the inferred final plan improved across all three scenarios with the use of perfect PDDL as compared to an uninformative prior over possible plans. The sequence accuracy also consistency improved with the use PDDL, regardless of noise level or the type of PDDL degradation. The three scenarios exhibited different levels of “noise,” defined as the percentage of utterances that did not end up in the finally agreed upon plan. The police incidents response scenario produced substantially higher noise (53%), as compared to the radioactive material leaking scenario before announcement (38%) and after announcement (17%). This is possibly because the police incidents scenario included durative-actions, whereas the others did not. Interestingly, perfect PDDL produced more substantial improvements in sequence accuracy when noise level was higher, in the radioactive material leaking scenario before announcement, and in police incidents scenario.

Accuracy in task allocation, on the other hand, did differ depending on the noise level

and the type of PDDL degradation. The noise rejection ratio was the same or better with PDDL or PDDL with a missing constraint, as compared to an uninformative prior, for scenarios with less noise (e.g. the radioactive material leaking scenarios before and after announcement). However, PDDL did not provide benefit to the noise rejection ratio for the police incidents scenario where the noise level was more than 50%. However, in this case PDDL did provide improvements in inferred task allocation.

The composite accuracy of PDDL and PDDL with missing constant are comparable, although they differ in task allocation and sequence accuracies. One of the possible explanations is that teams mostly did not violate the missing constraint (i.e., send robots before sending human), therefore, missing the constrain did not significantly impact the composite accuracy.

2.7.2 Sampling Hyper-parameters

This section discusses the results of hyper-parameter sampling. First, I show that each data point (i.e., each team's conversation) converges to different hyper-parameter values, then show that those values capture the characteristics of each data point. Second, I show how learning different sets of hyper-parameters improves different measures of accuracy, and describe how this is consistent with my interpretation of the hyper-parameters in my model.

Improvements in sequence accuracy versus noise rejection

The hyper-parameter β represents the noisiness in predicate ordering, while the hyper-parameter ω_p represents the noisiness in the assignment of predicates. Setting these parameters to a fixed value corresponds to an assumption about the noisiness of the data set. We can learn these parameters through Gibbs sampling, allowing these values to be adjusted according to different characteristics of each data set. The details of how I sample hyper-parameters were explained in Section 2.4.3. I performed 2,000 Gibbs sampling steps on the data from each planning session. The initial values of ω_p and β were sampled from their prior, where the parameters were set to the values described in Section 2.4.1.

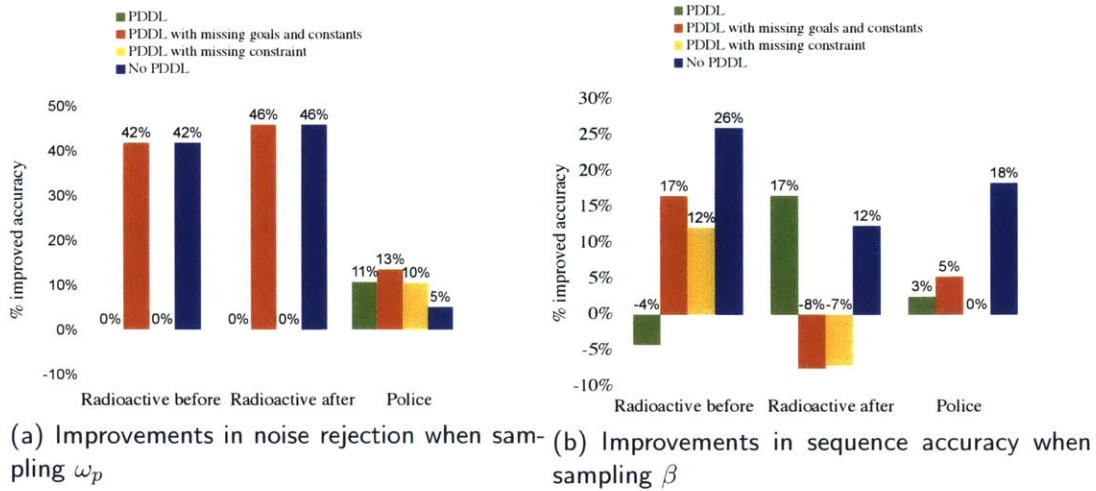


Figure 2-5: Percent improvements in median noise rejection and median sequence accuracy when sampling hyper-parameters versus setting $\omega_p = 0.8$ and $\beta = 5$.

I found that when I learned ω_p (with $\beta = 5$), the noise rejection rate improved compared with when I fixed $\omega_p = 0.8$. In the radioactive material leakage scenario, both before and after the mid-scenario announcement, the noise rejection ratio was improved by as much as 41% and 45%, respectively; in the police incident response scenario, I observed up to a 13% improvement (Figure 2-5a). Note that in all cases the median noise rejection ratio was maintained or improved with the sampling of ω_p .

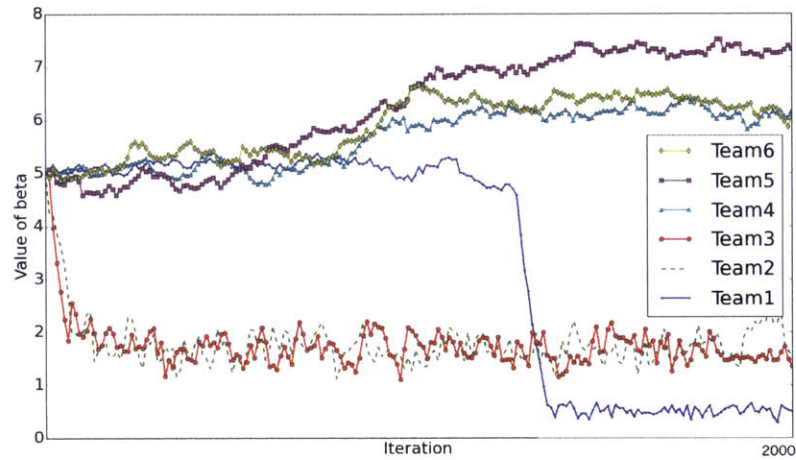
Similarly, when I learned β (with $\omega_p = 0.8$), sequence accuracies generally improved. In the radioactive material leakage scenario, before and after the announcement, the sequence accuracy improved by up to 26% and 16%, respectively; in the police incident response scenario, I observed up to an 18% improvement (Figure 2-5b). Note that three cases did see a degradation in accuracy of up to 4-8%. However in nine out of the twelve cases the sequence accuracy was maintained or improved with the sampling of β .

Interestingly, most of the samples achieved the highest overall composite accuracy when only *plan* and *s* were learned, and the hyper-parameters were fixed. In particular, I observed an average 5% ($\pm 3\%$) decrease in composite accuracy when sampling all four variables together. One of the possible explanations for this finding is that, due to the limited amount of data, Gibbs sampling may require many more iterations to converge all the variables. This result suggests that one may choose the set of hyper-parameters to

learn based on which measure of accuracy is more important to the user.

Interpretation of inferred values of hyper-parameters

(a) Examples of β value convergences



(b) Examples of ω_p value convergence

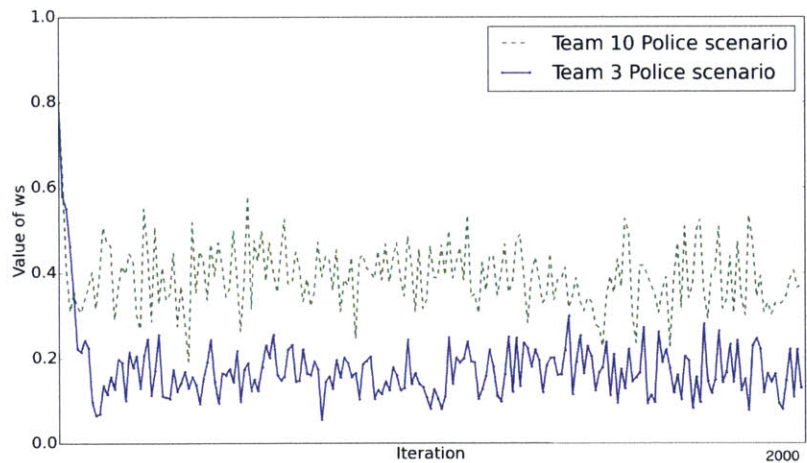


Figure 2-6: Inferred values of hyper-parameters (only showing subset of data set)

As described in Section 2.4.1, the ω_p parameter models the level of noise in predicates within the data. In other words, the ω_p parameter is designed to model how many

suggestions the team makes during the conversation that are subsequently included in the final plan. The high noise level means that a smaller number of predicates from the planning conversation are included in the final plan. A lower-valued ω_p will represent the characteristics of the conversation well, which may allow for better performance. (However, if the noise level is too high, the inference may still fail.)

To compare the learned value of ω_p with the characteristics of the conversation, I need a way to calculate how noisy the conversation is. The following is one way to compute the value of ω_p : First, count the utterances that contain any predicates. Then, count the utterances that contain predicates included in the final plan. The ratio between these two numbers can be interpreted as noisiness in the predicates; the lower the number, the more the team talked about many possible plans.

I performed this manual calculation for two teams' trials — Team 3 and 10 — to compare their values to the learned values. In Team 3's trial, only 19.4% of the suggestions made during the conversation were included in the final plan (i.e., almost 80% of suggestions were not relevant to the final plan). On the other hand, 68% of suggestions made in Team 10's trial were included in the final plan. Using this interpretation, Team 3's trial is more than twice as noisy as Team 10's trial.

The converged value of ω_p is lower in Team 3's trial than in Team 10's trial, reflecting the characteristics of each data set. Figure 2-6b shows the converged value of ω_p for each team's trial (sub-sampled, and for a subset of the dataset). The figure presents values of ω_p at each iteration of the Gibbs sampling step. Note that the samples from Team 3's trial converge to 20%, while the samples from Team 10's trial converge to 40%. The lower value of ω_p represents higher noise level, and matches my intuition.

However, there is no conclusive way to prove that these converged values are the true values. In theory, the Gibbs sampling algorithm only guarantees convergences to the true value with an infinite number of iterations. Therefore, I cannot prove that the converged ω_p variables shown in Figure 2-6 are the true values. In practice, a trace plot, such as that in Figure 2-6, is drawn in order to demonstrate convergence to a local optimum. The fact that the values appear to plateau after a burn-in period provides support of convergence to a local optimum point. Investigation of this potentially local optimum point suggests

that the ω_p value for each data point can be different, and that I can observe some relationship between the ω_p value and the characteristics of the data set. In addition, the manual calculation of ‘noisiness’ is only one way of interpreting the ‘noisiness’ of the data set. Therefore, this analysis should be considered as one possible way to gain insight into the learned values; not a rigorous proof of the relation between the learned value of the hyper-parameter and the characteristics of the data.

2.7.3 The Benefit of PDDL

This section provides additional evidence of the benefit of using PDDL by analyzing multiple runs using the same data and sampling algorithm. As explained in Section 2.4.3, Gibbs sampling is an approximate inference algorithm that can produce different results on each run.

In this section I evaluate runs over a wide range of different settings to show that the benefit of PDDL applies not just to a particular setting of parameters, but also to different settings. I analyzed three cases across a range of parameters: 1) learning both *plan* and *s*, 2) learning *plan*, *s* and ω_p and 3) learning *plan*, *s* and β . In the first case, I changed the value of α to range from 3 to 1,000, ω_p from 0.3 to 0.8, and β from 1 to 100. In the second case, in addition to α and β parameters, I varied the parameters for the prior distribution of ω_p — k_{ω_p} and θ_{ω_p} ; both ranging from 2 to 70. In the third case, in addition to α and ω_p parameters, I varied the parameters for the prior distribution of β — k_{β} and θ_{β} ; both ranging from 0.1 to 50. Values from the all ranges were selected randomly to produce a total of 613 runs.

Eighty-two percent of the 613 runs showed higher accuracy when PDDL was used than when PDDL was not used. This suggests that adding the structured prior improves accuracy over a wide range of parameter settings. Figure 2-7 presents the ratio of runs that saw benefit from the use of the PDDL, for each of the three scenarios.

Interestingly, the highest accuracy was not always achieved with perfect PDDL files; in some cases, the highest accuracy was achieved with imperfect PDDL files (e.g., PDDL file with missing goals/constraints, as described in Section 2.6). This observation may be

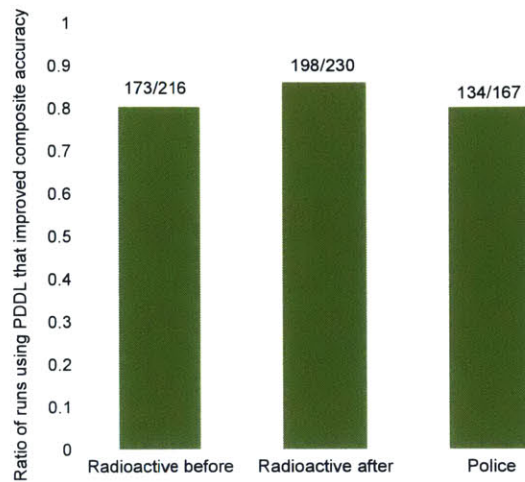


Figure 2-7: Ratio of runs that show the benefit of using PDDL

explained by the possibility that some finally agreed-upon plans 1) are not complete and/or 2) violate constraints (mostly due to participants' misunderstandings). For example: Prior to the announcement during the radioactive material leakage scenario, a number of teams had not finished building complete plans. Therefore, the final plans in these cases may have been better inferred with incomplete PDDL files (consistent with Table 2.4). In the police incident response scenario, however, a number of teams missed the constraint that the hot-spot patrolling task is only considered complete if that hot-spot is fully covered from 8:30 p.m. to 9 p.m. A number of teams dispatched police cars only for a portion of that time window, resulting in invalid plans with the perfect PDDL files (consistent with Table 2.5)

The improvements achieved by adding the structure to the prior using PDDL suggest that the structural information is beneficial to my inference problem. It would be interesting to systematically investigate the smallest set of structural information that achieves accuracy improvements, given a fixed computation budget, in future work.

2.7.4 The i.i.d Assumption on the Utterance in the Generative Model

My generative model considers that all utterances are independent and identically distributed samples from the plan variable. In other words, I consider that all utterances give equal evidence to a plan, regardless of the order in which they appear during the conversation. An alternative would be to have a different weight for each utterance, to take the ordering into account. In this section, I explain the reasons for the i.i.d. assumption, and how a simple extension to the current model can relax this assumption.

In the human subject data collected for this work, I did not observe a clear relationship between the order of an utterance and whether the suggestion is included in the final plan. For example, a number of teams decided to include parts of a plan that were discussed at the beginning of the conversation within their final plan, after discussing many other possibilities. The distribution of the utterances included in the final plan shown in Figure 2-8 suggests that a high portion of utterances discussed at the beginning of the meeting are included in the final plan.

An additional reason for the i.i.d. assumption is that, when a team discusses plans under time pressure, the planning sessions often consist of a small number of succinct communications. For example, the average number of predicates in all utterances in a planning session is 90, whereas the average number of predicates in a final plan is 12. A succinct conversation yields less available data for the inference; therefore, a complicated model may fail to correctly infer all the latent variables. A time series model, wherein the ordering is taken into account and the weight of each utterance is a latent variable that needs to be learned from the data, is an example of such a model.

However, a simple extension of the current model can relax this assumption and incorporate the different importance of each utterance. One way to decide an utterance's importance is to integrate human cognitive models. Human cognitive architectures [10] model human cognitive operations, such as the memory model [11]. For example, I can decrease the importance of each utterance as time proceeds in the planning session by applying varying weights to each utterance. A simple extension to the current model can

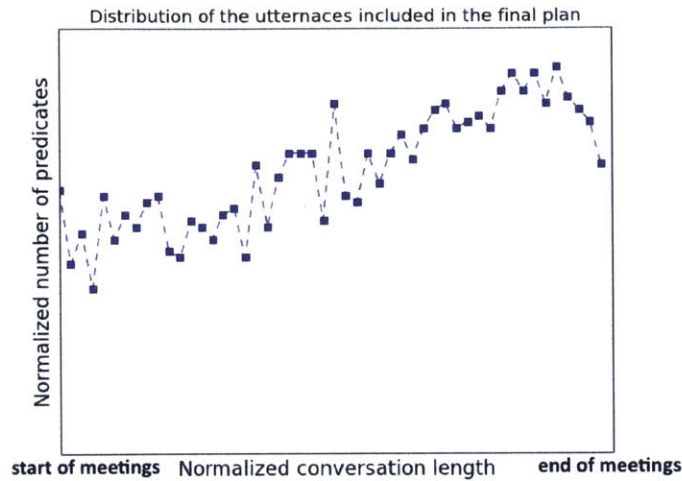


Figure 2-8: The distribution of the utterances included in the final plan (normalized)

be made to incorporate the memory model. Specifically, the variables ω_p and β can be modified to be vectors that represent weight or activation level of each utterance from the human cognition model [11]. The vector of ω_p will have the length of the utterances, $\omega_p = \{\omega_{p,1}, \omega_{p,2}, \dots, \omega_{p,T}\}$, where each $\omega_{p,t}$ represents the activation level of each utterance. Similarly, I can extend β to be a vector, where each β_t can represent how noisy each utterance is, weighing it accordingly. However, while these cognitive models are empirically well-verified, Whitehill et al. [133] pointed out that there is no structured way to set parameters for these models. In addition, it is unclear how the human memory model would differ depending on the characteristics of a given task. For example, the memory model may differ significantly for short, succinct conversations conducted under time pressure.

2.7.5 Engineering the Proposal Distribution in the Metropolis-Hastings Sampling Algorithm

This section describes the impact of different proposal distributions in the MH sampling step, and our rationale for designing the proposal distribution as described in Section 2.4.3.

There have been numerous studies conducted on selecting a family of candidate-

generating density functions [97, 66, 56, 54]. However, as pointed out by Chib et al. [34], there is no structured way to choose the proposal distribution. It becomes more challenging when the sampled object is not a simple scalar variable, but a more complicated object, such as the *plan* variable (i.e., tuples of sets of grounded predicates) in this work. For such an object, there are larger spaces of potential proposal distributions to choose from.

However, a good choice for proposal distribution can improve performance. Figure 2-9 shows the results of the two different proposal distributions used for this work. The preliminary version of this work [80] applied the following distribution:

- Select a predicate from the set of possible predicates. If it is in the current *plan*, move it to either: 1) the next set of predicates or 2) the previous set, or 3) remove it from the current *plan*. If it is not in the current *plan*, move it to one of the existing sets.
- Select two sets in the current *plan* and switch their orders.

One difference between the proposal distribution above and the one outlined in Section 2.4.3 is the set of allowed timestamps that a selected predicate can move to at each iteration. The above proposed distribution allows a predicate to move to any timestamp, whereas the one in Section 2.4.3 only allows a predicate to move to an adjacent timestamp.

The key insight into proposal distribution in this work is gained by investigating sequences of MH sampling steps and observing when a proposal distribution fails to propose a good move. In other words, I identify what moves are necessary to move a proposed value (i.e., proposed new plan) to the true value of the latent variable (i.e., true plan) when they are close to each other. Often, a predicate is one timestamp off from the true timestamp (i.e., one timestamp after or before), and the proposal distribution as contained in the preliminary work [80] often fails to suggest a better proposed point. This motivated us to create a proposal distribution enabling more frequent moves between adjacent timestamps than between any two timestamps. As a result, I observed a substantial improvement to accuracy in all scenarios, as shown in Figure 2-9.

While this particular proposal distribution cannot be applied to all cases, this insight suggests that the following approach could be useful when designing a proposal distribution

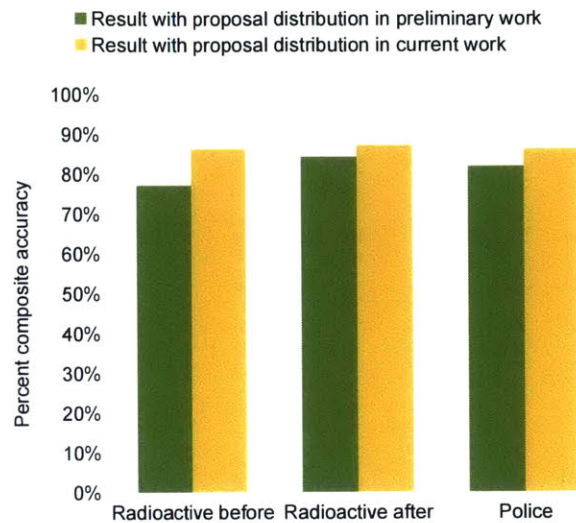


Figure 2-9: The impact of different proposal distributions (The highest accuracy with perfect PDDL files)

for non-scalar valued variables: First, a distance metric is defined between the two non-scalar valued variables. In my case, this step included defining the distance between two tuples of sets of predicates (i.e., *plan* variables). For example, the distance could be the average number of missing or extraneous predicates or the number of predicates that have incorrect timestamps. Second, starting from an initial proposed distribution, the distance between each sample and the true value is measured. Third, I can filter sample sequences when the distance is short, and visualize them. The shorter distance indicates moments when sampling could have almost reached the true value, but did not. Finally, the proposed distribution is modified to include the move that converts the samples in the third step to the true value within one or two moves. This process allows for insight into designing the proposal distribution. I leave further investigation of the systematic approach to future work.

2.8 Conclusion and Future Work

In this chapter, I have formulated the novel problem of performing inference to extract a finally agreed-upon plan from a human team's planning conversation. I presented my

approach that combines a probabilistic approach with logical plan validation, used to compute a highly structured prior over possible plans. My approach infers team plans without the need for historical data, using only situational information and data from a single planning session. I do not require the development or addition of a plan library to infer the plan, and demonstrate my solution is robust to incomplete knowledge of the planning problem. I demonstrated the benefit of this approach using human team meeting data collected from large-scale human subject experiments (total 96 subjects) and were able to infer the human teams' final plans with 86% accuracy on average.

Chapter 3

The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification

The human team decision-making process requires exchange of ideas. In order to build a machine that collaborates with humans in the decision-making process, machine learning algorithms must communicate to humans by providing output in forms that make sense to humans and are easily incorporated into the human-decision making process. While Chapter 2 demonstrated a machine learning model that “interpret decisions of humans,” this chapter presents models that can “makes sense to humans” by exploring and communicating patterns and structure in data to support human decision-making. I propose and validate a new unsupervised clustering model that provides interpretable explanations about machine learning results to humans, who use examples as a fundamental part of humans’ decision-making strategies.

Bayesian Case Model (BCM) is a general framework for Bayesian case-based reasoning (CBR) and prototype classification and clustering. BCM brings the intuitive power of CBR to a Bayesian generative framework. The BCM learns *prototypes*, the “quintessential”

observations that best represent clusters in a dataset, by performing joint inference on cluster labels, prototypes and important features. Simultaneously, BCM pursues sparsity by learning *subspaces*, the sets of features that play important roles in the characterization of the prototypes. The prototype and subspace representation provides quantitative benefits in interpretability while preserving classification accuracy. Human subject experiments verify statistically significant improvements to participants' understanding when using explanations produced by BCM, compared to those given by prior art.

3.1 Introduction

People like to look at examples. Through advertising, marketers present examples of people we might want to emulate in order to lure us into making a purchase. We might ignore recommendations made by Amazon.com and look instead at an Amazon customer's Listmania to find an example of a customer like us. We might ignore medical guidelines computed from a large number of patients in favor of medical blogs where we can get examples of individual patients' experiences.

Numerous studies have demonstrated that exemplar-based reasoning, involving various forms of matching and prototyping, is fundamental to humans' most effective strategies for tactical decision-making ([101, 35, 83]). For example, naturalistic studies have shown that skilled decision makers in the fire service use *recognition-primed decision making*, in which new situations are matched to typical cases where certain actions are appropriate and usually successful [83]. To assist humans in leveraging data sources to make better decisions, we desire that machine learning algorithms provide output in forms that are easily incorporated into the human decision-making process.

Studies of human decision-making and cognition provided the key inspiration for artificial intelligence CBR approaches [2, 122]. CBR relies on the idea that a new situation can be well-represented by the summarized experience of previously solved problems [122]. CBR has been used in important real-world applications [91, 19], but it does not learn the underlying complex structure of data in an unsupervised fashion and may not scale to datasets with high-dimensional feature spaces (as discussed in [123]).

In this thesis, I introduce a new Bayesian model, called the Bayesian Case Model (BCM), for prototype clustering and subspace learning. In this model, the prototype is the exemplar that is most representative of the cluster. The subspace representation is a useful output of the model because we neither need nor want the best exemplar to be similar to the current situation in all possible ways: for instance, a moviegoer who likes the same horror films as we do might be useful for identifying good horror films, regardless of their cartoon preferences. I model the underlying data using a mixture model, and infer sets of features that are important within each cluster (i.e., subspace). This type of model can help to bridge the gap between machine learning methods and humans, who use examples as a fundamental part of their decision-making strategies.

I show that BCM produces prediction accuracy comparable to or better than prior art for standard datasets. I also verify through human subject experiments that the prototypes and subspaces provide meaningful feedback for the characterization of important aspects of a dataset. In these experiments, the exemplar-based output of BCM resulted in statistically significant improvements to participants' performance of a task requiring an understanding of clusters within a dataset, as compared to outputs produced by prior art.

3.2 Background and Related Work

People organize and interpret information through exemplar-based reasoning, particularly when they are solving problems ([101, 27, 35, 83]). AI Case-Based Reasoning approaches are motivated by this insight, and provide example cases along with the machine-generated solution. Studies show that example cases significantly improve user confidence in the resulting solutions, as compared to providing the solution alone or by also displaying a rule that was used to find the solution [41]. However, CBR requires labels (i.e. solutions) for previous cases, and does not learn the underlying structure of the data in an unsupervised fashion. Maintaining transparency in complex situations also remains a challenge [123]. CBR models designed explicitly to produce explanations [1] rely on the backward chaining of the causal relation from a solution, which may not scale as complexity increases. The cognitive load of the user also increases with the complexity of the similarity measure used

for comparing cases [50]. Other CBR models for explanations require the model to be manually crafted in advance by experts [99].

An alternative way to organize information is using clustering. The mixture model is a powerful tool for discovering cluster distributions in an unsupervised fashion. However, this approach does not provide intuitive explanations for the learned clusters (as pointed out in [32]). Sparse topic models are designed to improve interpretability by reducing the number of words per topic [134, 45]. However, using the number of features as a proxy for interpretability is problematic, as sparsity is often not a good or complete measure of interpretability [50]. Explanations produced by mixture models are typically presented as distributions over features. Even users with technical expertise in machine learning may have a difficult time interpreting such output, especially when the cluster is distributed over a large number of features [50].

My approach, the BCM, simultaneously performs unsupervised clustering and learns both the most representative cases (i.e., prototypes) and important features (i.e., subspaces). BCM preserves the power of CBR in generating interpretable output, where interpretability comes not only from sparsity but from the prototype exemplars.

In my view, there are at least three widely known types of interpretable models: sparse linear classifiers ([126, 32, 129]); discretization methods, such as decision trees and decision lists (e.g., [43, 134, 45, 90, 60]); and prototype- or case-based classifiers (e.g., nearest neighbors [39] or a supervised optimization-based method [20]). (See [50] for a review of interpretable classification.) BCM is intended as the third model type, but uses unsupervised generative mechanisms to explain clusters, rather than supervised approaches [61] or by focusing myopically on neighboring points [13].

3.3 The Bayesian Case Model

Intuitively, BCM generates each observation using the important pieces of related prototypes. The model might generate a movie profile made of the horror movies from a quintessential horror movie watcher, and action movies from a quintessential action moviegoer.

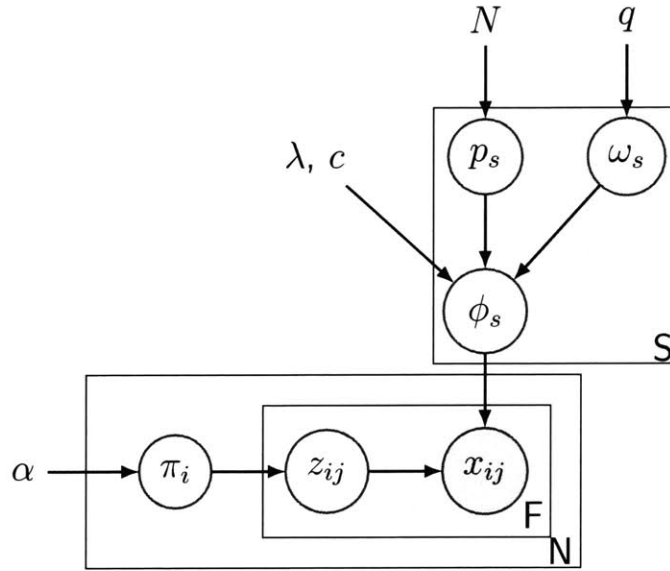


Figure 3-1: Graphical model for the Bayesian Case Model

BCM begins with a standard discrete mixture model [69, 23] to represent the underlying structure of the observations. It augments the standard mixture model with *prototypes* and *subspace feature indicators* that characterize the clusters. I show in Section 3.4.2 that prototypes and subspace feature indicators improve human interpretability as compared to the standard mixture model output. The graphical model for BCM is depicted in Figure 4-1.

I start with N observations, denoted by $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$, with each x_i represented as a random mixture over clusters. There are S clusters, where S is assumed to be known in advance. (This assumption can easily be relaxed through extension to a non-parametric mixture model.) Vector π_i are the mixture weights over these clusters for the i^{th} observation x_i , $\pi_i \in \mathbb{R}_+^S$. Each observation has P features, and I denote the j^{th} feature of the i^{th} observation as x_{ij} . Each feature j of the observation x_i comes from one of the clusters, the index of the cluster for x_{ij} is denoted by z_{ij} and the full set of cluster assignments for observation-feature pairs is denoted by \mathbf{z} . Each z_{ij} takes on the value of a cluster index between 1 and S . Hyperparameters q , λ , c , and α are assumed to be fixed.

The explanatory power of BCM results from how the clusters are characterized. While a standard mixture model assumes that each cluster take the form of a predefined parametric

distribution (e.g., normal), BCM characterizes each cluster by a *prototype*, p_s , and a *subspace feature indicator*, ω_s . Intuitively, the *subspace feature indicator* selects only a few features that play an important role in identifying the cluster and prototype (hence, BCM clusters are *subspace clusters*). I intuitively define these latent variables below.

Prototype, p_s : The prototype p_s for cluster s is defined as one observation in \mathbf{x} that maximizes $p(p_s|\omega_s, \mathbf{z}, \mathbf{x})$, with the probability density and ω_s as defined below. My notation for element j of p_s is p_{sj} . Since p_s is a prototype, it is one of the observations, so $p_{sj} = x_{ij}$ for some i . Note that more than one maximum may exist per cluster; in this case, one prototype is arbitrarily chosen. Intuitively, the prototype is the “quintessential” observation that best represents the cluster.

Subspace feature indicator ω_s : Intuitively, ω_s ‘turns on’ the features that are important for characterizing cluster s and selecting the prototype, p_s . Here, $\omega_s \in \{0, 1\}^P$ is an indicator variable that is 1 on the subset of features that maximizes $p(\omega_s|p_s, \mathbf{z}, \mathbf{x})$, with the probability for ω_s as defined below. Here, ω_s is a binary vector of size P , where each element is an indicator of whether or not feature j belongs to subspace s .

The generative process for BCM is as follows: First, I generate the subspace clusters. A subspace cluster can be fully described by three components: 1) a prototype, p_s , generated by sampling uniformly over all observations, $1 \dots N$; 2) a feature indicator vector, ω_s , that indicates important features for that subspace cluster, where each element of the feature indicator (ω_{sj}) is generated according to a Bernoulli distribution with hyperparameter q ; and 3) the distribution of feature outcomes for each feature, ϕ_s , for subspace s , which I now describe.

Distribution of feature outcomes ϕ_s for cluster s : Here, ϕ_s is a data structure where each “row” ϕ_{sj} is a discrete probability distribution of possible outcomes for feature j . Explicitly, ϕ_{sj} is a vector of length V_j , where V_j is the number of possible outcomes of feature j . Let us define Θ as a vector of the possible outcomes of feature j (e.g., for feature ‘color’, $\Theta = [\text{red}, \text{blue}, \text{yellow}]$), where Θ_v represents a particular outcome for that feature (e.g., $\Theta_v = \text{blue}$). I will generate ϕ_s so that it mostly takes outcomes from the prototype p_s for the important dimensions of the cluster. I do this by considering the

vector g , indexed by possible outcomes v , as follows:

$$g_{p_{sj}, \omega_{sj}, \lambda}(v) = \lambda(1 + c\mathbb{1}_{\{w_{sj}=1 \text{ and } p_{sj}=\Theta_v\}}),$$

where c and λ are constant hyperparameters that indicate how much we will copy the prototype in order to generate the observations. The distribution of feature outcomes will be determined by g through $\phi_{sj} \sim \text{Dirichlet}(g_{p_{sj}, \omega_{sj}, \lambda})$. To explain at an intuitive level: First, consider the irrelevant dimensions j in subspace s , which have $w_{sj} = 0$. In that case, ϕ_{sj} will look like a uniform distribution over all possible outcomes for features j ; the feature values for the unimportant dimensions are generated arbitrarily according to the prior. Next, consider relevant dimensions where $w_{sj} = 1$. In this case, ϕ_{sj} will generally take on a larger value $\lambda + c$ for the feature value that prototype p_s has on feature j , which is called Θ_v . All of the other possible outcomes are taken with lower probability λ . As a result, we will be more likely to select the outcome Θ_v , that agrees with the prototype p_s . In the extreme case where c is very large, we can copy the cluster's prototype directly within the cluster's relevant subspace and assign the rest of the feature values randomly.

An observation is then a mix of different prototypes, wherein we take the most important pieces of each prototype. To do this, mixture weights π_i are generated according to a Dirichlet distribution, parameterized by hyperparameter α . From there, to select a cluster and obtain the cluster index z_{ij} for each x_{ij} , I sample from a multinomial distribution with parameters π_i . Finally, each feature for an observation, x_{ij} , is sampled from the feature distribution of the assigned subspace cluster ($\phi_{z_{ij}}$). (Note that Latent Dirichlet Allocation (LDA) [23] also begins with a standard mixture model, though feature values in my model exist in a discrete set that is not necessarily binary.) Here is the full model, with hyperparameters c , λ , q , and α :

$$\begin{aligned} \omega_{sj} &\sim \text{Bernoulli}(q) \quad \forall s, j & p_s &\sim \text{Uniform}(1, N) \quad \forall s \\ \phi_{sj} &\sim \text{Dirichlet}(g_{p_{sj}, \omega_{sj}, \lambda}) \quad \forall s, j & \text{where } g_{p_{sj}, \omega_{sj}, \lambda}(v) &= \lambda(1 + c\mathbb{1}_{\{w_{sj}=1 \text{ and } p_{sj}=\Theta_v\}}) \\ \pi_i &\sim \text{Dirichlet}(\alpha) \quad \forall i & z_{ij} &\sim \text{Multinomial}(\pi_i) \quad \forall i, j & x_{ij} &\sim \text{Multinomial}(\phi_{z_{ij}}) \quad \forall i, j. \end{aligned}$$

My model can be readily extended to different similarity measures, such as standard kernel methods or domain specific similarity measures, by modifying the function g . For example, I can use the least squares loss i.e., for fixed threshold ϵ , $g_{p_{sj}, \omega_{sj}, \lambda}(v) = \lambda(1 + c\mathbb{1}_{[w_{sj}=1 \text{ and } (p_{sj} - \Theta_v)^2 \leq \epsilon]})$; or, more generally, $g_{p_{sj}, \omega_{sj}, \lambda}(v) = \lambda(1 + c\mathbb{1}_{[w_{sj}=1 \text{ and } \ell(p_{sj}, \Theta_v) \leq \epsilon]})$.

In terms of setting hyperparameters, there are natural settings for α (all entries being 1). This means that there are three real-valued parameters to set, which can be done through cross-validation, another layer of hierarchy with more diffuse hyperparameters, or plain intuition. To use BCM for classification, vector π_i is used as S features for a classifier, such as SVM.

3.3.1 Motivating example

This section provides an illustrative example for prototypes, subspace feature indicators and subspace clusters, using a dataset composed of a mixture of smiley faces. The feature set for a smiley face is composed of types, shapes and colors of eyes and mouths. For the purpose of this example, assume that the ground truth is that there are three clusters, each of which has two features that are important for defining that cluster. In Table 3.1, I show the first cluster, with a subspace defined by the color (green) and shape (square) of the face; the rest of the features are not important for defining the cluster. For the second cluster, color (orange) and eye shape define the subspace. I generated 240 smiley faces from BCM's prior with $\alpha = 0.1$ for all entries, and $q = 0.5$, $\lambda = 1$ and $c = 50$.

BCM works differently from Latent Dirichlet Allocation (LDA) [23], which presents its output in a very different form. Table 3.1 depicts the representation of clusters in both LDA (middle column) and BCM (right column). This dataset is particularly simple, and I chose this comparison because the two most important features that both LDA and BCM learn are identical for each cluster. However, LDA does not learn prototypes, and represents information differently. To convey cluster information using LDA (i.e., to define a topic), I must record several probability distributions – one for each feature. For BCM, I need only to record a prototype (e.g., the green face depicted in the top row, right column of the figure), and state which features were important for that cluster's

	Data in assigned to cluster	LDA	BCM	
		Top 3 words and probabilities	Prototype	Subspaces
1		 0.26 0.23 0.12		color () and shape () are important.
2		 0.26 0.24 0.16		color () and eye () are important.
3		 0.35 0.27 0.15		eye () and mouth () are important.

Table 3.1: The mixture of smiley faces for LDA and BCM

subspace (e.g., shape and color). For this reason, BCM is more succinct than LDA with regard to what information must be recorded in order to define the clusters. One could define a “special” constrained version of LDA with topics having uniform weights over a subset of features, and with “word” distributions centered around a particular value. This would require a similar amount of memory; however, it loses information, with respect to the fact that BCM carries a full prototype within it for each cluster.

A major benefit of BCM over LDA is that the “words” in each topic (the choice of feature values) are coupled and not assumed to be independent – correlations can be controlled depending on the choice of parameters. The independence assumption of LDA can be very strong, and this may be crippling for its use in many important applications. Given my example of images, one could easily generate an image with eyes and a nose that cannot physically occur on a single person (perhaps overlapping). BCM can also generate this image, but it would be unlikely, as the model would generally prefer to copy the important features from a prototype.

BCM performs joint inference on prototypes, subspace feature indicators and cluster labels for observations. This encourages the inference step to achieve solutions where clusters are better represented by prototypes. I will show that this is beneficial in terms of predictive accuracy in Section 3.4.1. I will also show through an experiment involving human subjects that BCM’s succinct representation is very effective for communicating

the characteristics of clusters in Section 3.4.2.

3.3.2 Inference: collapsed Gibbs sampling

I use collapsed Gibbs sampling to perform inference, as this has been observed to converge quickly, particularly in mixture models [62]. I sample ω_{sj} , z_{ij} , and p_s , where ϕ and π are integrated out. Note that I can recover ϕ by simply counting the number of feature values assigned to each subspace. Integrating out ϕ and π results in the following expression for sampling z_{ij} :

$$p(z_{ij} = s | z_{i \neq j}, \mathbf{x}, p, \omega, \alpha, \lambda) \propto \frac{\alpha/S + n_{(s,i,-j,\cdot)}}{\alpha + n} \times \frac{g(p_{sj}, \omega_{sj}, \lambda) + n_{(s,\cdot,j,x_{ij})}}{\sum_s g(p_{sj}, \omega_{sj}, \lambda) + n_{(s,\cdot,j,\cdot)}}, \quad (3.1)$$

where $n_{(s,i,j,v)} = \mathbb{1}(z_{ij} = s, x_{ij} = v)$. In other words, if x_{ij} takes feature value v for feature j and is assigned to cluster s , then $n_{(s,i,j,v)} = 1$, or 0 otherwise. Notation $n_{(s,\cdot,j,v)}$ is the number of times that the j^{th} feature of an observation takes feature value v and that observation is assigned to subspace cluster s (i.e., $n_{(s,\cdot,j,v)} = \sum_i \mathbb{1}(z_{ij} = s, x_{ij} = v)$). Notation $n_{(s,\cdot,j,\cdot)}$ means sum over i and v . I use $n_{(s,i,-j,v)}$ to denote a count that does not include the feature j . The derivation is similar to the standard collapsed Gibbs sampling for LDA mixture models [62].

Similarly, integrating out ϕ results in the following expression for sampling ω_{sj} :

$$p(\omega_{sj} = b | q, p_{sj}, \lambda, \phi, \mathbf{x}, \mathbf{z}, \alpha) \propto \begin{cases} q \times \frac{\mathbf{B}(g(p_{sj}, 1, \lambda) + n_{(s,\cdot,j,\cdot)})}{\mathbf{B}(g(p_{sj}, 1, \lambda))} & b = 1 \\ 1 - q \times \frac{\mathbf{B}(g(p_{sj}, 0, \lambda) + n_{(s,\cdot,j,\cdot)})}{\mathbf{B}(g(p_{sj}, 0, \lambda))} & b = 0, \end{cases} \quad (3.2)$$

where \mathbf{B} is the Beta function and comes from integrating out ϕ variables, which are sampled from Dirichlet distributions.

3.4 Results

In this section, I show that BCM produces prediction accuracy comparable to or better than LDA for standard datasets. I also verify the interpretability of BCM through human

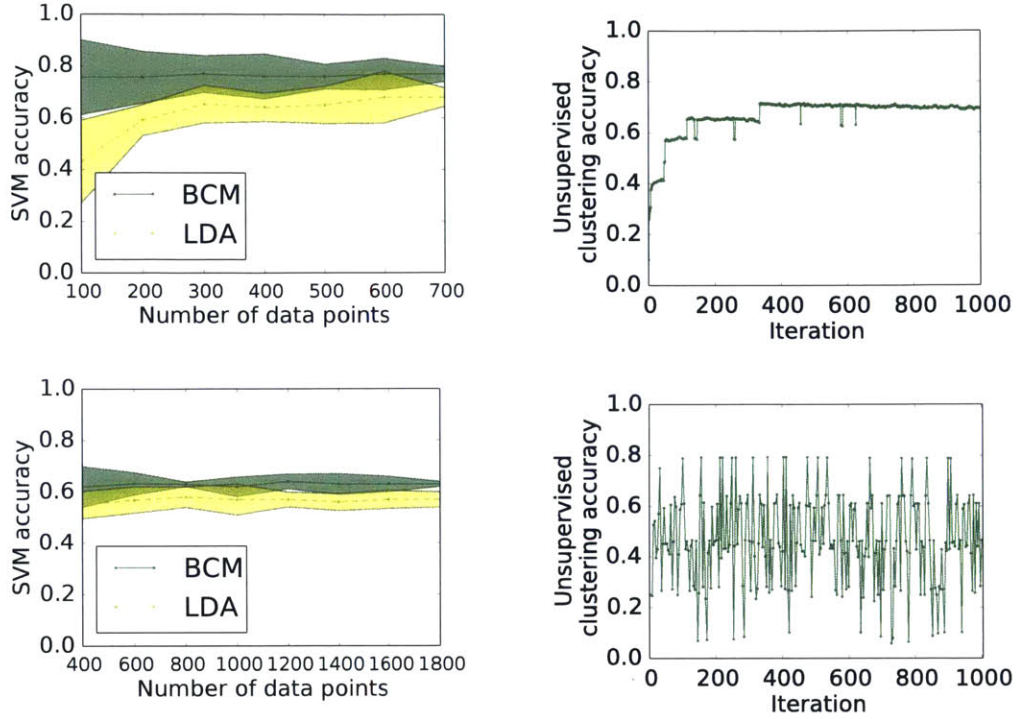
subject experiments involving a task that requires an understanding of clusters within a dataset. I show statistically significant improvements in objective measures of task performance using prototypes produced by BCM, compared to output of LDA. Finally, I visually illustrate that the learned prototypes and subspaces present as meaningful feedback for the characterization of important aspects of the dataset.

3.4.1 BCM maintains prediction accuracy.

I show that BCM output produces prediction accuracy comparable to or better than LDA, which uses the same mixture model (Section 3.3) to learn the underlying structure but does not learn explanations (i.e., prototypes and subspaces). I validate this through use of two standard datasets: *Handwritten Digit* [73] and *20 Newsgroups* [88]. I use the implementation of LDA available from [107], which incorporates Gibbs sampling, the same inference technique used for BCM.

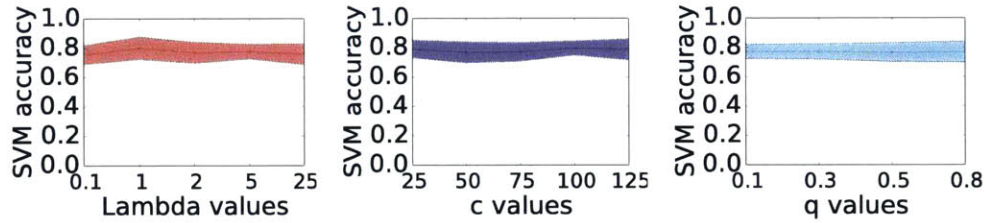
Figure 3-2a depicts the ratio of correctly assigned cluster labels for BCM and LDA. In order to compare the prediction accuracy with LDA, the learned cluster labels are provided as features to a support vector machine (SVM) with linear kernel, as is often done in the LDA literature on clustering [23]. The improved accuracy of BCM over LDA, as depicted in the figures, is explained in part by the ability of BCM to capture dependencies among features via prototypes, as described in Section 3.3. I also note that prediction accuracy when using the full *20 Newsgroups* dataset acquired by LDA (accuracy: 0.68 ± 0.01) matches that reported previously for this dataset when using a combined LDA and SVM approach [136]. Also, LDA accuracy for the full *Handwritten Digit* dataset (accuracy: 0.76 ± 0.017) is comparable to that produced by BCM using the subsampled dataset (70 samples per digit, accuracy: 0.77 ± 0.03).

As indicated by Figure 3-2b, BCM achieves high unsupervised clustering accuracy as a function of iterations (e.g., about 70% for the *Handwritten Digit* dataset). I can compute this measure for BCM because each cluster is characterized by a prototype – a particular data point with a label in the given datasets. (Note that this is not possible for LDA.) I set α to prefer each π_i to be sparse, so only one prototype generates each observation, and I



(a) Accuracy and standard deviation with SVM

(b) Unsupervised accuracy for BCM



(c) Sensitivity analysis for BCM

Figure 3-2: Prediction test accuracy reported for the *Handwritten Digit* [73] and *20 Newsgroups* datasets [88]. (a) applies SVM for both LDA and BCM, (b) presents the unsupervised accuracy of BCM for *Handwritten Digit* (top) and *20 Newsgroups* (bottom) and (c) depicts the sensitivity analysis conducted for hyperparameters for *Handwritten Digit* dataset. Datasets were produced by randomly sampling 10 to 70 observations of each digit for the *Handwritten Digit* dataset, and 100-450 documents per document class for the *20 Newsgroups* dataset. The *Handwritten Digit* pixel values (range from 0 to 255) were rescaled into seven bins (range from 0 to 6). Each 16-by-16 pixel picture was represented as a 1D vector of pixel values, with a length of 256. Both BCM and LDA were randomly initialized with the same seed (one half of the labels were incorrect and randomly mixed), The number of iterations was set at 1,000. $S = 4$ for *20 Newsgroups* and $S = 10$ for *Handwritten Digit*. $\alpha = 0.01$, $\lambda = 1$, $c = 50$, $q = 0.8$.

use that prototype's label for the observation. Sensitivity analysis in Figure 3-2c indicates that the additional parameters introduced to learn prototypes and subspaces (i.e., q , λ and c) are not too sensitive within the range of reasonable choices.

3.4.2 Verifying the interpretability of BCM

I verified the interpretability of BCM by performing human subject experiments that incorporated a task requiring an understanding of clusters within a dataset. This task required each participant to assign 16 recipes, described only by a set of required ingredients (recipe names and instructions were withheld), to one cluster representation out of a set of four to six. (This approach is similar to those used in prior work to measure comprehensibility [74].) I chose a recipe dataset¹ for this task because such a dataset requires clusters to be well-explained in order for subjects to be able to perform classification, but does not require special expertise or training.

My experiment incorporated a within-subjects design, which allowed for more powerful statistical testing and mitigated the effects of inter-participant variability. To account for possible learning effects, I blocked the questions with different conditions and balanced the assignment of participants into the two ordering groups: Half of the subjects were presented with all eight questions in condition 1 (e.g., BCM) first, while the other half first saw the eight questions in condition 2 (e.g., LDA). Subjects were encouraged to answer the questions as quickly and accurately as possible, but were instructed to take a 5-second break every four questions in order to mitigate the potential effects of fatigue.

Examples from BCM v.s LDA

Cluster representations (i.e., explanations) from LDA were presented as the set of top ingredients for each recipe topic cluster. For BCM I presented the ingredients of the prototype without the name of the recipe and without subspaces. The number of top ingredients shown for LDA was set as the number of ingredients from the corresponding BCM prototype and ran Gibbs sampling for LDA with different initializations until the

¹Computer Cooking Contest: <http://liris.cnrs.fr/ccc/ccc2014/>

The interface is divided into several sections:

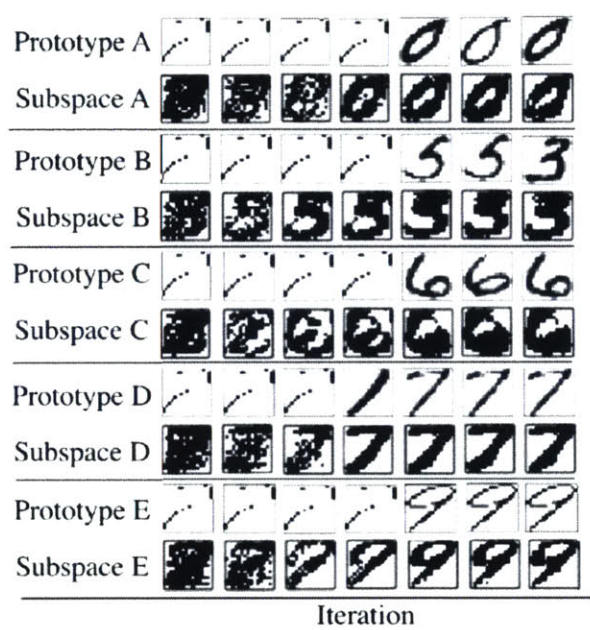
- New dish:** A list of ingredients including salt, water, vanilla, sugar, peach, flour, egg, cream cheese, sour cream, whipped cream, baking powder, and four radio buttons labeled Category 1, Category 2, Category 3, and Category 4. A 'Submit' button is at the bottom.
- Categories:** A list of dish names including brownie, cheesecake, chili, chutney, cornbread, crepe, curry, gingerbread, meatloaf, pancake, pasta, pizza, potato, salad, punch, quiche, and salsa.
- Descriptions of unnamed categories:** A header for the table below.
- Representative examples of categories:** A table with four columns: Category 1 Example, Category 2 Example, Category 3 Example, and Category 4 Example. The rows list various ingredients like powder sugar, vinegar, chili powder, etc.

Category 1 Example	Category 2 Example	Category 3 Example	Category 4 Example
powder sugar	vinegar	chili powder	vinegar
cracker	apple	broth	potato
sour cream	cinnamon	cumin	bell pepper
cream cheese	cranberry	garlic	carrot
egg	ginger	onion	hot sauce
pie filling	raisin	pepper	olive
sugar	sugar	tomato juice	onion
vanilla		tomato paste	sugar
butter		oil	oil
		turkey	chili powder
		black bean	corn
		corn	salt

Figure 3-3: Web-interface for the human subject experiment

ground truth clusters were visually identifiable. Given a new recipe (i.e., given by as a set of ingredients) and cluster representations either from BCM or LDA, participants are asked to classify the new recipe. Twenty-four participants (10 females, 14 males, average age 27 years) performed the task, answering a total of 384 questions.

Using explanations from BCM, the average classification accuracy was 85.9%, which was statistically significantly higher ($\chi^2(1, N = 24) = 12.15, p \ll 0.001$) than that of LDA, (71.3%). For both LDA and BCM, each ground truth label was manually coded by two domain experts: the first author and one independent analyst (kappa coefficient: 1). These manually-produced ground truth labels were identical to those that LDA and BCM predicted for each recipe. There was no statistically significant difference between BCM and LDA in the amount of time spent on each question ($t(24) = 0.89, p = 0.37$); the overall average was 32 seconds per question, with 3% more time spent on BCM than on LDA. Subjective evaluation using Likert-style questionnaires produced no statistically significant differences between reported preferences for LDA versus BCM.



(a) *Handwritten Digit* dataset

Cluster	Prototype (Recipe names)	Ingredients (Subspaces)
1	<i>Herbs and Tomato in Pasta</i>	basil, garlic, Italian seasoning, oil, pasta, pepper, salt, tomato
2	<i>Generic chili recipe</i>	beer, chili powder, cumin, garlic, meat, oil, onion, pepper, salt, tomato
3	<i>Microwave brownies</i>	baking powder, sugar, butter, chocolate, chopped pecans, eggs, flour, salt, vanilla
4	<i>Spiced-punch</i>	cinnamon, stick, lemon juice, orange juice, pineapple juice, sugar, water, whole cloves

(b) *Recipe* dataset

Figure 3-4: Learned prototypes and subspaces for the *Handwritten Digit* and *Recipe* datasets.

3.4.3 Learning subspaces

Figure 3-4a illustrates the learned prototypes and subspaces as a function of sampling iterations for the *Handwritten Digit* dataset. For the later iterations, shown on the right of the figure, the BCM output effectively characterizes the important aspects of the data. In particular, the subspaces learned by BCM are pixels that define the digit for the cluster's prototype.

Interestingly, the subspace highlights the *absence* of writing in certain areas. This makes sense: For example, one can define a '7' by showing the absence of pixels on the left of the image where the loop of a '9' might otherwise appear. The pixels located where there is variability among digits of the same cluster are not part of the defining subspace for the cluster.

Because I initialized randomly, in early iterations, the subspaces tend to identify features common to the observations that were randomly initialized to the cluster. This is because ω_s assigns higher likelihood to features with the most similar values across observations within a given cluster. For example, most digits 'agree' (i.e., have the same zero pixel value) near the borders; thus, these are the first areas that are refined, as shown in Figure 3-4a. Over iterations, the third row of Figure 3-4a shows that BCM learns a cluster of the digits "5" and "3," which tend to share many pixel values in similar locations. Note that the sparsity of the subspaces can be customized by hyperparameter q .

Next, I show results for BCM using the Computer Cooking Contest dataset in Figure 3-4b. Each prototype consists of a set of ingredients for a recipe, and the subspace is a set of important ingredients that define that cluster, highlighted in red boxes. For instance, BCM found a "chili" cluster defined by the subspace "beer," "chili powder," and "tomato." A recipe called "Generic Chili Recipe" was chosen as the prototype for the cluster. (Note that beer is indeed a typical ingredient in chili recipes.)

3.5 Conclusion

The Bayesian Case Model provides a generative framework for case-based reasoning and prototype-based modeling. Its clusters come with natural explanations; namely, a pro-

prototype (a quintessential exemplar for the cluster) and a set of defining features for that cluster. I showed the quantitative advantages in prediction quality and interpretability resulting from the use of BCM. Exemplar-based modeling (nearest-neighbors, case-based reasoning) has historical roots dating back to the beginning of artificial intelligence; this method offers a fresh perspective on this topic, and a new way of thinking about the balance of accuracy and interpretability in predictive modeling.

Chapter 4

iBCM: Interactive Bayesian Case Model — Empowering Humans via Intuitive Interaction

Building on BCM in Chapter 3, this chapter presents the *interactive* Bayesian Case Model (iBCM), which enables the user to interact with the clustering model. Users provide direct input to iBCM in order to achieve effective clustering results, and iBCM optimizes the clustering by achieving a balance between what the data indicate and what the user indicates as useful.

Clustering methods optimize the partitioning of data points with respect to an internal metric, such as likelihood, in order to approximate the goodness of clustering. However, this internal metric does not necessarily translate into effective clustering from the user's perspective. This chapter presents the iBCM, a model that opens a communication channel between the clustering model and the user. Users can provide direct input to iBCM in order to achieve effective clustering results, and iBCM optimizes the clustering by creating a balance between what the data indicate and what makes the most sense to the user. This model provides feedback for users and does not assume any prior knowledge of machine learning on their part. I provide quantitative evidence that users are able to obtain more satisfactory clustering results through iBCM than without an interactive model. I also demonstrate the use of this method in a real-world setting where

computer language class teachers utilize iBCM to cluster students' coding assignments for grading.

4.1 Introduction

The ultimate goal of any clustering method is to partition data points in the most effective and useful way for the user. A number of existing methods [49] are able to group a large number of data points according to certain internal metrics. Unlike classification methods where prediction accuracy often serves as an evaluation metric, clustering metrics that directly optimize for a particular application are difficult to articulate and can be domain-specific. In clustering, there might be multiple ways to cluster data points that are equally good according to the internal metric of the clustering method, but some of these methods may better align with a user's knowledge or preference. For example, if a user in a retail setting wants to cluster customer data in order to build a recommendation engine and pair sales staff with appropriate customers using purchase records, clustering with respect to item categories or region of residence could yield equally good results. However, clustering with respect to region may offer a more cost-effective implementation of the engine by matching the recommendation plans with the existing team structure of the company. Hard-coding such information does not translate to other domains, and also requires in-depth knowledge of machine learning.

The iBCM empowers users via direct, intuitive interaction with a clustering model, in order to incorporate their expert domain knowledge and preference while simultaneously optimizing the internal goodness of clustering metric. In this work, we define 'interaction' to be humans being able to modify prototypes and subspaces. Moreover, the communication channel is bi-directional: iBCM communicates back to its users by providing explanations in natural language, offering insight into potential conflicts between user feedback and data point patterns and allowing for more fluent collaboration between the user and machine learning models.

The communication channel does not assume that a user has prior knowledge of machine learning. Instead, users provide examples (i.e., data points) and features of these

examples in order to articulate their feedback. For example, when building a recommendation engine for films, users may think that the length of a specific movie is not an important feature for clustering, even if its use would yield good performance. Instead, the user may think that genre or the number of awards each film has received are more important features. iBCM incorporates the user's feedback and propagates it to the inference engine in order to achieve better clustering. If user-suggested features conflict with what the data indicates, iBCM then provides feedback using the examples.

Desiderata Ultimately, iBCM aims to enable efficient and intuitive use of clustering models, without requiring in-depth knowledge about machine learning. However, establishing intuitive interactions with a complex machine learning system while assuming no prior user knowledge introduces a number of challenges. Even if the interaction is simplified to the level of performance metrics (confusion matrix) [77], it still requires the user to understand the confusion matrix. Some domains, such as computer vision, offer a readily intuitive interaction medium [131]; however, this is not typically the case. iBCM combines Case-Based Reasoning (CBR) approaches [2, 122] with a Bayesian generative model to both generate effective clustering and provide an intuitive medium for interaction. Specifically, iBCM uses examples (data points provided by the user) for communication, enabling interaction between the model and any user with some degree of familiarity with the data. The use of examples as a medium for interaction is well-grounded in several cognitive science studies that demonstrated that exemplar-based reasoning, involving various forms of matching and prototyping, is fundamental to humans' most effective strategies for tactical decision-making ([101, 35, 83]). In order to assist user decision-making by leveraging large data sources, we desire machine learning algorithms that communicate in forms that are easily incorporated into the human decision-making process.

Once intuitive interaction has been established, the next challenge is to determine the correct way to incorporate user feedback in order to achieve a balance between the user's input and solutions that optimize the internal metrics of the clustering method. If there is a conflict between the suggestions made by the user and the output from the clustering model, there must be a way for the model to both provide feedback to the user

and intuitively explain its internal states. Communication with the user is essential to a successful interactive system — it improves transparency, which has been identified as a major factor in establishing user trust in adaptive agents [111, 58] and improving system acceptance [68, 127].

Contributions The key contribution of this thesis chapter is to introduce a unified framework for clustering, user interaction and explanation generation. This unified framework for clustering enables the output representation to use the same representation as the parameters of the model without loss of information. The output that is presented to users is thus directly interpretable in terms of the generative process of the model. iBCM also offers for users to interact directly with a representation that maps to parameters of the model (i.e., examples and features of data points) without loss of information. This approach enables the model to incorporate user feedback and articulate its internal states without a loss of information. While explanations using highly sophisticated language of statistics may offer in-depth understanding for expert users [93], iBCM provides intuitive explanations for non-expert users.

4.2 Related work

Types of prior art that are relevant to the described approach include interface designs intended to aid user workflow [105], the interactive visualization of clustering results [89, 65, 8] and interactive machine learning systems [5, 14, 7]. Interfaces that improve user workflow when modifying model settings are intended to reduce the burden of repeated processes to achieve the desired clustering results [105]. Instead of modifying model settings, some systems aim to improve the user’s internalization of the current clustering through visualization [89, 65, 8].

One of the most commonly used methods of improving clustering is to explore multiple model parameter settings. To better support this iterative workflow, there are interfaces that allow for the tracking of model changes or comparing different models of machine learning [105]. There have also been works with the goal of designing a smart interface

that can adjust to user behavior during highly dynamic tasks [8]. Although these systems can improve the efficiency of users' iterative workflows, indirect interaction with models is time-consuming. Also, this approach requires users to have in-depth knowledge of machine learning or only offers domain-specific solutions.

Another way to help users better understand clustering results is through visualization, rather than changing the clustering itself. Visualization techniques are often developed for specific domains (e.g., topic model visualization [31] or geographical data visualization [65]). Some also introduce new metrics to help users internalize and conceptualize clustering results [63]. Some of these combine their functionalities to change model parameters in the interface [65]. However, this approach assumes that the current clustering is somewhat aligned with effective clustering results from the user's perspective.

In order to overcome the limitations of indirect interaction with machine learning methods, there has been growing interest in the development of interactive machine learning systems [5], ranging from studying the theoretical aspect of interactive clustering [14] to building systems for use in specific domains [131, 89]. Some works simplified the medium of interaction to the level of performance metrics. For example, in classification tasks, [77] users specified their preferred confusion matrix for classifications. However, even this simplified method of interaction requires the user to understand the performance metrics and to know how to assign a numerical score between 0 and 1 to their input. For some domains, such as computer vision, the medium for intuitive interaction is naturally available — users can directly manipulate decision boundaries in pictures [131], for example. However, this is not typically true for high-dimensional complex data points. In more complex domains, such as clustering documents, interactive clustering systems are available [18, 89]; however, the medium of interaction assumes an expert level of user knowledge, such as n-grams for machine learning systems [18] or keyword weights for topic clusters [89]. Some works had users communicate their feedback through data points [7, 6], such as by providing positive and negative examples to train a classifier [6]. However, the difference between machine representation (e.g., the averaged features of data points) and representation of information obtained from human feedback (i.e., the examples) may lead to inconsistencies in how user feedback is incorporated into the machine.

iBCM's unified framework for interactive machine learning addresses the challenges listed above by incorporating user feedback and articulating its internal states without the loss of information [85].

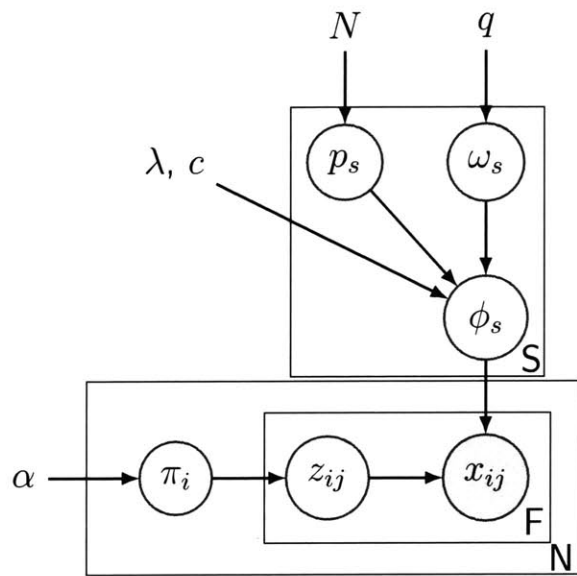
4.3 Interactive Bayesian Case Models (iBCM)

iBCM is an interactive version of the Bayesian Case Model (BCM), which combines case-based reasoning and a Bayesian generative model to perform clustering. I showed that BCM yielded statistically significant improvements to objective measures of interpretability in Chapter 3. The interpretability of BCM is one of the essential ingredients for building an intuitive interactive machine learning system, especially with regard to a bi-directional communication channel between the user and the model. This section reviews BCM and presents the internal mechanisms that allow unified framework for interactive machine learning.

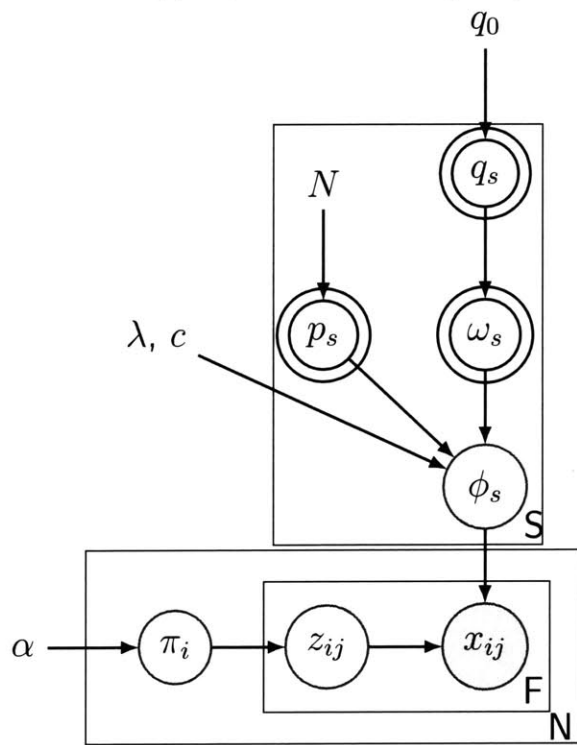
4.3.1 Bayesian Case Model

BCM, as introduced in 3, begins with a standard discrete mixture model [69, 23] in order to represent the underlying structure of the observations. It augments the standard mixture model with *prototypes* and *subspace feature indicators* that characterize the clusters. A graphical model of BCM is depicted in Figure 4-1a.

BCM begins with N observations, denoted by $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$, with each x_i represented as a random mixture over clusters. There are S clusters, where S is assumed to be known in advance. (This assumption can easily be relaxed through extension to a non-parametric mixture model, such as [22].) Vector π_i are the mixture weights over these clusters for the i^{th} observation x_i , $\pi_i \in \mathbb{R}_+^S$. Each observation has P features, and I denote the j^{th} feature of the i^{th} observation as x_{ij} . Each feature j of the observation x_i comes from one of the clusters, the index of the cluster for x_{ij} is denoted by z_{ij} and the full set of cluster assignments for observation-feature pairs is denoted by \mathbf{z} . Each z_{ij} takes on the value of a cluster index between 1 and S . Hyperparameters q, λ, c , and α are assumed to be fixed.



(a) Bayesian Case Model (BCM)



(b) interactive Bayesian Case Model (iBCM)

Figure 4-1: Graphical model depicting the BCM and iBCM. Double circle nodes represent *interacted latent variables*.

The explanatory power of BCM results from how the clusters are characterized. While a standard mixture model assumes that each cluster takes the form of a predefined parametric distribution (e.g., normal), BCM characterizes each cluster by a *prototype*, p_s , and a *subspace feature indicator*, ω_s . Intuitively, the *subspace feature indicator* selects only a few features that play an important role in identifying the cluster and prototype (hence, BCM clusters are *subspace clusters*). BCM defines these latent variables as follows:

Prototype, p_s : The prototype p_s for cluster s is defined as one observation in \mathbf{x} that maximizes $p(p_s|\omega_s, \mathbf{z}, \mathbf{x})$, with the probability density and ω_s as defined below. The notation for element j of p_s is p_{sj} . Since p_s is a prototype, it is equal to one of the observations, such that $p_{sj} = x_{ij}$ for some i .

Subspace feature indicator ω_s : Intuitively, ω_s 'turns on' the features that are important for characterizing cluster s and selecting the prototype, p_s . Here, $\omega_s \in \{0, 1\}^P$ is an indicator variable that is 1 on the subset of features that maximizes $p(\omega_s|p_s, \mathbf{z}, \mathbf{x})$, with the probability for ω_s as defined below. Here, ω_s is a binary vector of size P , where each element is an indicator of whether or not feature j belongs to subspace s .

I showed in Chapter 3 that BCM produces prediction accuracy comparable to or better than prior art for standard datasets. They also verify through human subject experiments that the prototypes and subspaces present as meaningful feedback for the characterization of important aspects of a dataset. In these experiments, the exemplar-based output of BCM resulted in statistically significant improvements to participants' performance of a task that required an understanding of clusters within a dataset, as compared to outputs produced by prior art.

4.3.2 Interactive Bayesian Case Model (iBCM)

The main difference between iBCM and BCM is that iBCM introduces *interacted latent variables* that represent a variable inferred through both user feedback and the data – p , ω and q (Figure 4-1b). Here, I introduce a new notation (double-circled nodes) for graphical models in order to represent human-*interacted latent variables*. *Interacted latent variables* are learned through user feedback and information obtained from data points. These

variables differ from observed variables (shaded nodes) because they may change their value dynamically through interaction. I also distinguish them from pure latent variables (unfilled nodes) because I may want to incorporate high-confidence user feedback, even if it conflicts with inference results obtained without user interaction. Figure 4-1 shows the graphical models of BCM and the iBCM side-by-side. In addition to interacted latent variables, q in iBCM is a matrix, where, as in BCM, q is a scalar hyperparameter. The benefit of this augmentation is explained in Section 4.3.2.

One of the challenges of incorporating user feedback into a machine learning system is balancing this feedback with information obtained from the data. For example, there may be a way to cluster data points that is not ideal according to the model's internal metric, but is more useful to a user. However, incorporating user feedback must be done cautiously — I must inform users when their feedback conflicts with information obtained from the data, while still maintaining a positive user experience.

This section presents two mechanisms of iBCM designed to address these challenges: confidence-based user feedback and the explanation given to users by the machine upon receiving this feedback.

Intuitive interaction medium

The interaction in this unified framework allows direct use of the internal representation of the model as the medium for interaction. Users interact with iBCM using a set of data points that they have provided: representative data points from each cluster (i.e., prototypes) and a set of their important features (i.e., subspaces). The only assumption this interaction makes on users is some degree of familiarity with the data points that they provide. This type of medium mimics the way that humans develop effective strategies for tactical decision-making ([101, 35, 83]).

Using data points and their features as a medium for interaction is particularly beneficial when operating within a domain with high-dimensional complex data points; for example, patent documentation, where investigating a single data point can become a daunting, time-consuming task. The iBCM interaction medium makes this simple. For example, if a user is more familiar with a particular data point or considers it to be espe-

cially important (such as a milestone patent), they can incorporate their domain knowledge by suggesting that data point as a prototype, thereby anchoring clustering with this point. The user can then suggest which features are important within that prototypical document. The ability to control prototypes and subspaces together allows for a simple yet atomic level of user control.

Communication from iBCM to users also benefits from this intuitive interaction medium (i.e., prototypes and subspaces). iBCM uses the same representation for its communication with users upon receiving feedback (Section 4.3.2).

Incorporating user feedback into iBCM

The inference pipeline for incorporating user feedback involves the following three steps: 1) listening to users — updating cluster information variables ω or q according to user feedback; 2) propagating user feedback — rearranging cluster label variables z ; and 3) listening to data — resampling cluster labels z and/or w .

iBCM uses a simple mechanism to incorporate user feedback: asking the user about the degree of confidence they have in their feedback. This extra information is simple enough to not burden the interaction, yet provides additional granularity for user control. Internally, the confidence-specific inference pipeline enables iBCM to adjust the impact of the feedback on the clustering results.

A. Listen to users: update interacted latent variables This section describes the different inference pipelines that iBCM applies depending on the user’s confidence in their feedback. For each feedback they provide, users have a choice of selecting ‘very confident’ (i.e., high confidence) or ‘somewhat confident’ (i.e., low confidence). The Low confident feedback influences the interacted latent variable, q , which then impacts its child interacted latent variables (i.e., prototypes, p and subspaces ω) when they are resampled. Higher-confidence feedback directly influences the child interacted latent variables.

When a user provides feedback with low confidence, iBCM propagates this information

to the prior level of interacted latent variable, q , as follows:

$$q_{s,f} = \begin{cases} \min(1, Hq_0) & \text{if low-confidence feedback on } \omega_{s,f} \\ q_0 & \text{otherwise,} \end{cases}$$

where H is a constant greater than 1, representing higher probability for $q_{s,f}$, and q_0 represents a constant hyperparameter (set to 0.8 in this work). Note that for BCM, this q variable is a scalar hyperparameter that governs the sparsity of the subspaces (e.g., the proportion of important features of a prototype). In the iBCM, q is a matrix and represents the sparsity of the subspaces of each prototype within a cluster. In the generative story, $q_{s,f}$ represents user feedback about how likely it is for feature f of cluster s to be important — and, therefore, contained within subspaces. During the inference step, variables $q_{s,f}$, once updated by user feedback, become the pseudo counts when updating the multinomial distribution, which then makes $\omega_{s,f}$ more likely to be part of subspaces.

In the case of high-confidence user feedback (i.e., a user reports that he/she is 'very confident' about their feedback), iBCM honors feedback completely by updating the ω interacted latent variable in Figure 4-1b as follows:

$$\omega_{s,f} = \begin{cases} \text{user specified value} & \text{if high-confidence feedback on } \omega_{s,f} \\ \omega_{s,f} & \text{else.} \end{cases}$$

In BCM, $\omega_{s,f}$ is 1 if feature f is an important feature for cluster s , and 0 otherwise. In iBCM, $\omega_{s,f}$ is updated to be identical to high-confidence feedback submitted by the user. Doing so updates the characteristics of cluster s , and therefore the clustering labels, z , in the inference pipeline. In combination with re-inferring the z variable (as described in Section C), this enables iBCM to learn a new clustering based on features that have been identified as important by the user.

Note that if a user provides careless feedback with overconfidence or has different understanding of the domain, a hard reset of the interacted latent variable ω may result poor clustering. For example, if a user selects the length of a specific movie to be an important feature for clustering movies, but there is no good way to cluster data

points based on the length of movies, the clustering results may be different from what a user expects. In this case, iBCM provides an explanation as to why the re-inferred clustering labels, z , is different from what users expected (described in Section 4.3.2). This explanation is intended to offer insight for users and to improve future interaction.

B. Propagate user feedback to accelerate inference One way to propagate user feedback is to update the interacted latent variables and re-infer all other variables in the model. However, unless the inference technique guarantees a globally optimal solution, doing so only offers locally minimal solutions that may not reflect the user’s feedback. In order to make sure that the new solution reflects the feedback, I require a way to quickly move the current solution to a solution space closer to the local minimum that reflects the user’s feedback.

In order to move to a solution space closer to the user’s feedback, iBCM propagates the feedback to the cluster labels, z . In sampling-based inference, doing so effectively moves solutions for all latent and interacted latent variables to a more ‘favorable’ solution space for the user (i.e., closer to a solution space that matches user’s feedback). The rationale behind this procedure is that, unlike with an inference procedure with a random start, we are not hoping for a random walk through the solution space — we know where in the solution space we want to be. Therefore, I can simulate the process of restarting Gibbs sampling multiple times with random initialization and choosing the most favorable instance with regard to the user’s feedback.

To do this, iBCM updates cluster labels, z , as follows:

$$z_{i,f} = \begin{cases} s & \text{if } x_{i,f} = p_{s,f} \text{ for interacted } \omega_{s,f} \\ \text{Uniform}(1, S) & \text{otherwise,} \end{cases}$$

where $p_{s,f}$ is the value of feature f of the prototype of cluster s . For example, if user feedback indicates that feature 1 of cluster A is important (i.e., $\omega_{s,f}$ is interacted), then iBCM propagates this information by setting $z_{i,f}$ to be s for all $x_{i,f}$ that has the same value of $p_{s,f}$. All other $z_{i,f}$ are sampled uniformly over all possible cluster labels, $1, \dots, S$.

Note that this step serves to create a balance between user feedback and patterns within the data. When re-inferring variables (as described in Section C), iBCM starts from a solution space that is closer to what the user wants, but moves toward a solution that also reflects the data patterns.

C. Listen to data: re-infer all other variables When a user provides low-confidence feedback, iBCM re-infers subspaces, ω , and clustering labels, z , by performing Gibbs sampling for only these variables. When the re-inferred interacted variable, ω , is different from what users indicated in a low-confidence case, iBCM points out where the inconsistency comes from. For example, the model might display: "You suggested that feature 1 is not important, but I think it is important. If you feel strongly about this feedback, please provide the same feedback with high confidence".

For high-confidence user feedback, iBCM only re-infers the clustering labels and completely honors the subspaces specified by users.

Note that the above approach is one of many possible ways to update interacted latent variables and re-infer other variables, and the approach can be customized. In Section 4.4, we show that this approach successfully incorporates user feedback.

Delivering feedback from the iBCM to users

The goal of providing explanations is to allow iBCM to give feedback to users when the information from patterns in the data conflicts with user-submitted feedback. Delivering these explanations opens up a communication channel between iBCM and the user, allowing for more fluent collaboration.

iBCM generates explanations using information from two sources: 1) cluster labels, z , and 2) the likelihood of prototypes and subspaces. Both types of explanation are triggered by the same condition: inconsistency between user feedback and the resulting subspaces and cluster memberships. For example, the likelihood of prototypes and subspaces are calculated to select the most likely prototypes and subspaces, then are compared to the user feedback. If they are different, iBCM describes the discrepancy.

Explanations using cluster labels One way to generate explanations is to base them on the cluster assignments, z , where $z_{i,f}$ represents the cluster assignment of feature f of i -th data point. Note that in a mixture model, one data point may have multiple cluster assignments for each feature. This distribution of assignments is represented by π_i , a vector of length S .

Explanations are generated by analyzing data points with the same maximum element of π_i , as well as commonly shared or not shared features among those points. For example, if all assessed data points have the same value for feature B, but different values for feature A, then the explanation generated could read as follows: “101 items in group 1 have different feature A values. However, most of the items in this group have the same feature B value”. This type of explanation provides common characteristics of each cluster using data points within that cluster in a “feed-forward” fashion, and represents how user input has propagated to clustering labels. However, this method does not directly offer “feedback” from iBCM — representations of internal states of the model indicating why certain data points are assigned to certain clusters.

Explanations using examples and important features In order to provide deeper insight into the internal state of iBCM, explanations can also be provided using interacted latent variables (i.e., prototypes and subspaces). Utilizing variables from the model to generate explanations offers more sophisticated information about the internal states of the model and the clustering results. These interacted variables have been shown to be effective when communicating clustering results to humans, as shown in Chapter 3. To generate explanations in this fashion, likelihoods are used as “scores” to indicate how likely each feature is to belong to the subspace. For example: “In group 1, important features are currently marked as feature A and feature B. However, feature C seems to be more

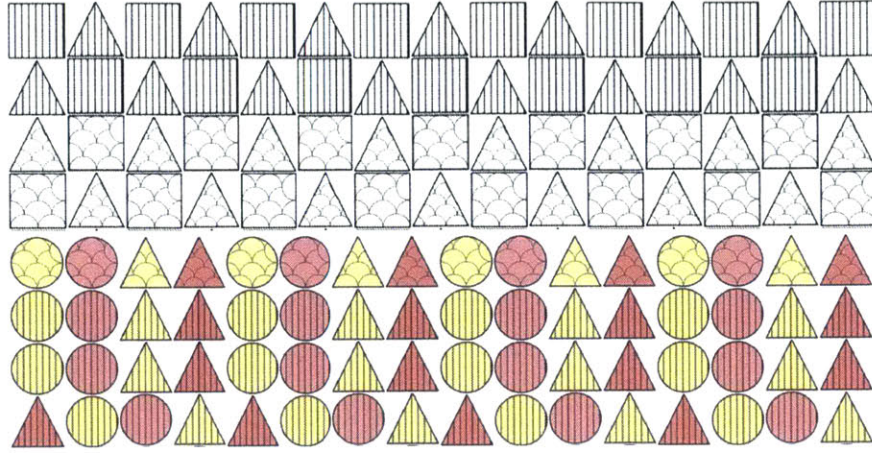


Figure 4-2: A subset of data points. Each data point has two features (top) or three features (bottom): shape, color and pattern.

important in this group". Likelihoods are defined as follows (directly from iBCM):

$$p(\omega_{sj} = b | q_{sj}, p_{sj}, \lambda, \phi, \mathbf{x}, \mathbf{z}, \alpha) \propto \begin{cases} q_{sj} \times \frac{\mathbf{B}(g(p_{sj}, 1, \lambda) + n_{(s, \cdot, j, \cdot)})}{\mathbf{B}(g(p_{sj}, 1, \lambda))} & b = 1 \\ 1 - q_{sj} \times \frac{\mathbf{B}(g(p_{sj}, 0, \lambda) + n_{(s, \cdot, j, \cdot)})}{\mathbf{B}(g(p_{sj}, 0, \lambda))} & b = 0, \end{cases}$$

where $g_{p_{sj}, \omega_{sj}, \lambda}(v) = \lambda(1 + c \mathbb{1}_{[w_{sj}=1 \text{ and } p_{sj}=\Theta_v]})$, Θ_v is a particular feature value and \mathbf{B} is the Beta function. $n_{(s, \cdot, j, v)}$ is the number of times that the j^{th} feature of an observation takes feature value v and that observation is assigned to subspace cluster s (i.e., $n_{(s, \cdot, j, v)} = \sum_i \mathbb{1}(z_{ij} = s, x_{ij} = v)$). Hyperparameter λ, c and α are assumed to be fixed. These definitions follow BCM.

Note that providing this type of explanation is only possible with this unified framework, in which internal clustering, user interaction and explanations are all done in the same representation without loss of information. This enables iBCM to articulate its internal states to users in the same intuitive and consistent way.

Question	1	2	3	4	5	6	7	8
# of features	2	2	2	2	3	3	3	3
is balanced	True	False	True	False	True	False	True	False
# of clusters	2	2	3	3	4	4	5	5

Table 4.1: Experiment design. Participants were randomly assigned to one of four groups, with four participants per group. "Is balanced" means that there were an equal number of data points with each feature. (e.g., equal number of red squares and yellow squares).

4.4 Evaluation

I performed human experiments to validate my approach. The first experiment was designed to measure quantifiable improvements in how well final clustering performed with iBCM matches the intent of the subject compared with clustering with BCM by using dataset with ground truth. Then in the second experiment, I implemented to validate the approach in a real-world setting (online computer science education).

4.4.1 Human subject experiment for a domain with ground truth

I designed an experiment to evaluate the quality of clustering obtained through interaction while controlling for data characteristics that can affect clustering results. In order to do this, I constructed datasets that could be clustered in multiple ways, all of which were equal in terms of likelihoods, with the optimal ways to cluster according to internal metrics also known a priori.

There were eight conditions in the experiment (shown in Table 4.1), each of which had three variables: the number of features, number of clusters and balance. The number of features reflects the complexity of the domain which is fairly modest in this experiment. The data points in the first four conditions had two features (shape and pattern, shown on the top of Figure 4-2), while points in the remaining four conditions each had three features (shape, pattern and color, depicted on the bottom of Figure 4-2). Each feature had two possible values; for example, the color could be either red or yellow. The number of clusters reflects the complexity of interaction between iBCM and the user. When the number of clusters is even, clustering can be performed such that the number of data

points falling into each cluster is exactly the same. If the number of clusters is odd, some clusters must contain fewer data points than others. The phrase “is balanced” was defined as the presence of an equal number of data points with the same feature values. For example, if there are an equal number of red circles, yellow circles, red triangles and yellow triangles, then “is balanced” is true. This factor was intended to test how subjects’ interaction with iBCM is influenced by the complexity of a dataset. Note that a balanced dataset can have a different number of data points that fall into an individual cluster due to the number of clusters.

The experiment was designed using a Latin square to assign the conditions to the 24 participants in a balanced manner. Each subject answered nine questions, with the first considered a practice question that was not included in the analysis.

The subjects performed the following steps for each question: First, subjects were asked how they wanted to cluster the data; for example, according to shape or color. Randomly ordered data points were shown to the subjects at this point. This step collected the ground truth for subjects’ desired clustering, which would later be compared with the clustering results following interaction with iBCM. The user was then showed the clustering results of BCM, which selects one of the several optimal ways to cluster the data points, as the dataset was designed to have multiple, equally likely optimal solutions. At the third step, subjects rated how well the clustering they had been shown matched their preferred clustering on a five-point Likert scale, with 1 indicating they strongly disagreed and 5 that they strongly agreed that the two clustering attempts matched. Next, subjects interacted with iBCM to provide their feedback and customize the clustering. Finally, when subjects indicated that they were done, they were asked to rate how well the final clustering matched with their preferred clustering on the same five-point Likert scale. All participants took mandatory breaks after every other question in order to manage fatigue. The goal of iBCM is to achieve stronger subject agreement that their clustering results matched their preference following interaction with the model. Note that according to the model’s internal metric, the quality of clustering at step 2 is likely to be equal to that of the final clustering.

Figure 4-3 shows the interface used for this experiment. In this interface, each row

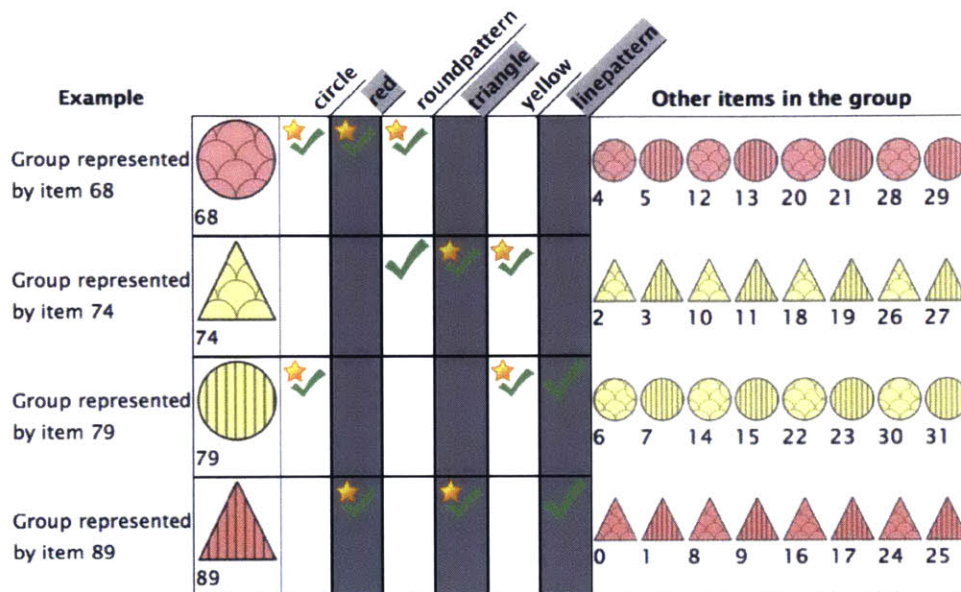


Figure 4-3: Graphical user interface for interaction. Each row represents a cluster. Prototypes of each cluster are shown on the left. The numbers below them are data ID. Subspaces are marked as stars. Each data point has three features: shape, color and pattern.

represents a cluster, where the example that best represents the cluster (i.e., prototype) is shown on the left of each row. The name of each data point is simply a number depicted below each point. Features (denoted with a check mark) and important features (denoted with a check mark and star) are depicted to the right of each prototype. For example, on the second row, the data number 74 has the 'round pattern', 'triangle' and 'yellow' features, but the important features for characterizing this cluster are 'yellow' and 'triangle.'

Subjects were able to provide feedback using this interface in two ways: by clicking a check mark to designate a feature and the value as important (when clicked, a check mark changed to a check mark with a star) or unimportant, and by clicking one of the items marked as "other items in the group" in order to promote it to become a prototype of a cluster. When an item was clicked, subjects were prompted with options to designate it as a prototype of any of the clusters.

When subjects provided either type of feedback, the interface asked them to indicate

their degree of confidence in that feedback (i.e., 'somewhat confident' or 'very confident'). All feedback was cached in iBCM, which incorporated and recomputed clustering data points only when subjects clicked the 'rerun' button. Subjects were allowed to provide as much feedback as they wanted before hitting 'rerun.' They were able to click the 'rerun' button a maximum of 50 times; however, all subjects were satisfied with their clustering results before reaching this number of interactions (maximum number of clicks are 31).

Question 1	$z = 6.10, p < .001$
Question 2	$z = 6.09, p < .001$
Question 3	$z = 6.13, p < .001$
Question 4	$z = 6.08, p < .001$
Question 5	$z = 6.09, p < .001$
Question 6	$z = 6.08, p < .001$
Question 7	$z = 6.11, p < .001$
Question 8	$z = 6.11, p < .001$

Table 4.2: Human subject experiment results

For all conditions, regardless of the number of features, number of clusters or whether the dataset was identified as balanced or not, iBCM achieved significantly better agreement with the users' desired clusterings, as shown in Table 4.2. The complexity of the domain, the interaction with iBCM or of the dataset did not influence the effectiveness of the model. I used the two-sided Wilcoxon signed rank test for paired observations to assess the statistical significance of the observed differences in agreement before and after interaction with iBCM. Unlike a t-test, the Wilcoxon signed rank test does not assume normal data distribution. This experiment validates that the subjects achieved significantly more satisfying results using iBCM compared with BCM, even though the results from BCM provided equally good solutions according to the model's internal metric. In the next section, I present a real-world implementation of a system utilizing iBCM for further validation.

4.4.2 Implementing and validating iBCM system for online education

In this section, I present an implementation and validation of iBCM for a real-world application: introductory programming education. Twelve teachers who previously taught introductory Python computer language classes (i.e., domain experts) were invited to interact with the iBCM system. Based on a Likert questionnaire, teachers strongly agreed (with statistical significance) that iBCM better supported them in exploring the variations in hundreds of student submissions, to prepare homework review sessions. In this experiment, twelve teachers explored 554 solutions from 48 homework assignments collected over the course of years at my institute.

Many colleges offer several courses that teach basic programming to hundreds of students simultaneously. It can take hours for teachers to grade exams for courses of this size and many teachers generate grading rubrics for each problem based on a small random sample of student solutions. Rather than depend on such a sample, I implemented a system for introductory Python programming teachers that uses iBCM as the main computational engine for grouping. The system utilizes iBCM to help teachers group their students' assignments provide helpful comments for the students.

Education is a particularly appropriate domain for iBCM, because the clustering results must reflect both patterns within the data and the knowledge of domain experts. Experts (i.e., teachers in this experiment) have accumulated years of knowledge that the clustering algorithm can leverage to make the clustering results more effective for their tasks. For example, each teacher may have a different style of grading and providing feedback — some may focus on the key concepts presented in the class while others may focus on students' coding practices (e.g., the use of good variable names and comments) as much as understanding the key concepts. Systems that can simplify the grouping of assignments are particularly useful for massive open online courses (MOOCs). Reviewing thousands of student assignments is a very time-consuming task, especially when also trying to provide constructive feedback on an individual level. iBCM allows teachers to incorporate their domain knowledge to achieve the grouping of assignments that is most effective for their

```

def dotProduct(listA, listB):
    '''
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    '''
    res = 0
    for j in range(len(listA)):
        res = res + listA[j] * listB[j]
    return res

def dotProduct(listA, listB):
    '''
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    '''
    combo_list = []
    for number in range(len(listA)):
        combo_list.append(listA[number] * listB[number])
    return sum(combo_list)

def dotProduct(listA, listB):
    '''
    listA: a list of numbers
    listB: a list of numbers of the same length as listA
    '''
    dot_product = sum(listA[i] * listB[i] for i in range(len(listA)))
    return dot_product

```

Figure 4-4: The raw data

task.

However, building a complete system for use in engineering education introduces an additional challenge into the interactive machine learning system: processing raw data. I use a method called OverCode [59] to extract relevant features for iBCM. OverCode uses both static and dynamic analysis to combine similar solutions that perform the same computations, but may use different variable names or statement orders. OverCode was developed to help MOOC instructors explore variations among student solutions and provide appropriate feedback to a large number of students. It has been shown to allow teachers to more quickly develop a high-level view of students' understanding and misconceptions as well as provide feedback that is relevant for more students [59]. Note that the output from OverCode is both human-readable and executable (the student's raw submission is shown in Figure 4-4 and snippets of OverCode are shown in Figure 4-5). OverCode renames variables to reflect their behavior. It finds common variables that behave the same way in many solutions run on the same test case, and then renames those common variables to most popular name.

Inputs to iBCM are binary vectors indicating the existence of the features, including renamed variables and language-specific keywords, such as listA, assert, while.

Tool A

dot product

Ready for Input

Cluster Prototypes and Subspaces

```
def dotProduct(listA, listB):
    total=0
    iB=0
    while iB<len(listA):
        product=listA[iB]*listB[iB]
        total+=product
        iB+=1
    return total

def dotProduct(listA, listB):
    total=0
    for (a,b) in zip(listA, listB):
        product=a*b
        total+=product
    return total

def dotProduct(listA, listB):
    if len(listA)!=len(listB):
        print 'length of A and B need to be the same'
        return None
```

Cluster members

Show all stacks

Promote to Prototype

```
def dotProduct(listA, listB):
    length=len(listA)
    iB=0
    total=0
    while iB<length:
        total=total+listA[iB]*listB[iB]
        iB+=1
    return total
```

Promote to Prototype

```
def dotProduct(listA, listB):
    iB=0
    total=0
    while iB<len(listA):
        product=listA[iB]*listB[iB]
        total=total+product
        iB=iB+1
    return total
```

(a) The first group is highlighted in blue, and the submissions in the group are shown on the right.

Tool A

dot product

Ready for Input

Cluster Prototypes and Subspaces

```
def dotProduct(listA, listB):
    total=0
    iB=0
    while iB<len(listA):
        product=listA[iB]*listB[iB]
        total+=product
        iB+=1
    return total

def dotProduct(listA, listB):
    total=0
    for (a,b) in zip(listA, listB):
        product=a*b
        total+=product
    return total

def dotProduct(listA, listB):
    if len(listA)=len(listB):
        print 'length of A and B need to be the same'
        return None
```

Cluster members

Show all stacks

Promote to Prototype

```
def dotProduct(listA, listB):
    length=len(listA)
    iB=0
    total=0
    while iB<length:
        total=total+listA[iB]*listB[iB]
        iB+=1
    return total
```


Promote to Prototype

```
def dotProduct(listA, listB):
    iB=0
    total=0
    while iB<len(listA):
        product=listA[iB]*listB[iB]
        total=total+product
        iB=iB+1
    return total
```

(b) When a user hovers their cursor over a keyword (`len`), it is highlighted in a blue rectangle, indicating that it can be interacted with (i.e., clicked).

Figure 4-5: Experiment interface for iBCM

Tool B

dot product 

[Click here to get new grouping](#)

Cluster Prototypes and Subspaces

```
def dotProduct(listA, listB):
    total=0
    for(a,b) in zip(listA, listB):
        product=a*b
        total+=product
    return total
```

Cluster members

[Show all stacks](#)

```
def dotProduct(listA, listB):
    total=0
    for i in xrange(0, len(listA), 1):
        total=listA[i]*listB[i]+total
    return total
```

```
def dotProduct(listA, listB):
    total=0
    iB=0
    while iB<len(listA):
        product=listA[iB]*listB[iB]
        total+=product
        iB+=1
    return total
```


```
def dotProduct(listA, listB):
    if len(listA)!=len(listB):
        print 'length of A and B need to be the same'
```

```
def dotProduct(listA, listB):
    total=0
    for x in range(len(listA)):
        return sum(listA[x]*listB[x] for x in range(0, len(listA)))
```

```
def dotProduct(listA, listB):
    total=0
    for i in range(0, len(listA)):
        total+=listA[i]*listB[i]
    return total
```

(a) The first group is highlighted in blue and the submissions in the group are shown on the right.

Tool B

dot product 

[Click here to get new grouping](#)

Cluster Prototypes and Subspaces

```
def dotProduct(listA, listB):
    if len(listA)!=len(listB):
        print 'length of A and B need to be the same'
        return None
    else:
        total=0
        for i in range(len(listA)):
            total+=listA[i]*listB[i]
        return total
```

```
def dotProduct(listA, listB):
    listCB=[a*b for(a,b) in zip(listA, listB)]
    ans=sum(listCB)
    return ans
```

```
def dotProduct(listA, listB):
    return sum([listA[i]*listB[i] for i in range(len(listA))])
```

Cluster members

[Show all stacks](#)

```
def dotProduct(listA, listB):
    total=0
    for(a,b) in zip(listA, listB):
        product=a*b
        total+=product
    return total
```

```
def dotProduct(listA, listB):
    total=0
    for i in xrange(0, len(listA), 1):
        total=listA[i]*listB[i]+total
    return total
```

```
def dotProduct(listA, listB):
    for x in range(len(listA)):
        return sum(listA[x]*listB[x] for x in range(0, len(listA)))
```

(b) When 'Click here to get new groupings' button is clicked, a new grouping is shown.

Figure 4-6: Experiment interface for BCM with random restarts

Figure 4-5 depicts the interface that subjects used to interact with iBCM. On the left column, it shows prototypical examples (one of the students' submissions) for each cluster. The parts surrounded by red rectangles indicate subspaces (important features). When a prototype is selected — highlighted in blue, as in the first row in Figure 4-5 — assignments that fall into that cluster are depicted on the right.

Conditions

In this experiment, subjects were given two conditions: iBCM (Figure 4-5) and BCM with an option to regroup the submissions (referred below as BCM with re-grouping, shown in Figure 4-6). In the iBCM condition, subjects were instructed to modify subspaces and prototypes (Figure 4-5b) in any way that is helpful for their tasks. The BCM with re-grouping condition offered a pre-selected grouping with an option to click a button to see a new grouping. These new groupings were prepared prior to the experiment by restarting Gibbs sampling inference from different initializations. Having an option to see a new grouping may engage users more by allowing interaction with the system as compared to a static system that provides a single grouping option to the users. When compared to iBCM, BCM with regrouping is a stronger basis of comparison than static BCM without an option to see new groupings.

Tasks and procedure

Subjects were told that they are preparing a homework review session of an introductory Python class and were asked to write their list of “discoveries”. The list of discoveries is an exhaustive list of approaches that students took. Subjects were told that their goal is to explore the full spectrum of approaches. The stated purpose of the homework review sessions was to give feedback to students about their homework assignments to improve students' coding abilities. Emphasis was placed on the fact that subjects were not to group or grade student submissions.

Each subject performed this task for five homework assignments (the first problem was used to train the subjects and was not included in the analysis). Prior to the exploration task of each problem, each subject was given the problem descriptions (i.e., homework

assignment that students are given) and two minutes to brainstorm solutions of their own before exploring student submissions. They were then given ten minutes to write down their list of discoveries using either iBCM or BCM with re-grouping. After a pair of iBCM and BCM with re-grouping conditions, subjects filled out a survey. Each subject filled out two surveys over the course of the experiment.

The experiment was designed using a Latin square to assign the homework assignments to the twelve participants in a balanced manner. The order in which participants were assigned to conditions was randomly selected, but the total number of times both conditions were shown was balanced.

In iBCM condition, subjects could click words from the programming assignment to place a red square around them - this denoted they were to be included in the subspace of the cluster. Similarly subjects clicked words in red rectangles to remove the red rectangle to denote that the words were to be excluded from the subspaces. In order to modify prototypes, subjects could click the "Promote to Prototype" button above each student's submission. The "Demote from prototype" buttons then appeared above all the current prototypes on the left column. Subjects could replace one of the current prototypes with the new choice of prototype using the "Promote to Prototype" and "Demote from Prototype" buttons. During this interaction, a visual indication of the computation process was provided to subjects on the left top corner. The left top banner changed to "Please hold. Thinking..." while iBCM was performing computation. When iBCM completed computation, the interface was automatically refreshed to reflect changes in groupings made upon the subject's feedback.

In the BCM with re-groupings condition, subjects could click the "Click here to get new grouping" button to view a new grouping. These new groupings were prepared prior to the experiment by restarting Gibbs sampling inference from different initializations. Pilot studies suggested that thirty different groups was a sufficient number of different groups to provide users with ample groupings given the limited experiment time. This button was clicked a maximum of 12 times.

Participants were asked 15 survey questions comparing the two conditions (questions shown in Table 4.3). Participants responded to the questions using a 5-point Likert scale

p value	Questions
$p < 0.01$	Comparing A to B, I like tool A _____ than tool B.
$p < 0.01$	Comparing A to B, when using tool A, I was _____ satisfied by the overall performance of myself than tool B.
$p = 0.09$	Comparing A to B, The tool A increased the productivity of the task _____ than tool B.
$p < 0.04$	Comparing A to B, The tool A contributed for the completion the task _____ than tool B.
$p < 0.001$	Comparing A to B, The tool A helped me _____ than tool B to explore the full spectrum of students' submissions.
$p < 0.004$	Comparing A to B, The tool A helped me _____ than tool B to discover different types of students' submissions.
$p = 0.085$	Comparing A to B, I relied on the tool A _____ than tool B to complete the task.
$p < 0.008$	Tool A was _____ useful than B in identifying important features to expand my list of discoveries.
$p = 0.35$	Comparing A to B, when using tool A, I saw _____ different types of submissions students made than using tool B.
$p < 0.001$	Being able to modify important features (i.e., red rectangles) was helpful to explore students submissions.
$p < 0.008$	Being able to modify prototypes was helpful to explore students submissions.

Table 4.3: Survey questions (tool A: iBCM, tool B: BCM with re-groupings).

ranging from “much less” to “much more” for the first eight questions, from “much fewer” to “much more” for the ninth question, and from “strongly disagree” to “strongly agree” for the last two questions.

Results

p -values and corresponding questions are shown in Table 4.3. Questions were designed to evaluate users' satisfaction with the tools, users' perception of the usefulness of tools in achieving the goal (i.e., exploring the full spectrum of student submissions) and in completing the task (i.e., expanding discovery list). Twelve out of fifteen questions asked in the survey yielded statistically significant results, showing that participants preferred and were satisfied with iBCM in exploring student submissions. Participants also agreed that being able to modify features and prototypes was useful for their task (statistically significant). I used a Wilcoxon signed-rank test and tested the null hypothesis that the distribution of the one sampled case (i.e., collected Likert-scale values) is symmetric about 3 (i.e., 'neutral' in Likert-scale).

The survey also included open-ended comments about the situations in which each tool was useful. A number of participants pointed out that interacting with iBCM allowed them to select keywords and prototypes in the way that is interesting and helpful for their tasks. Related comments included: “I was able to pull out examples that were really representative of a lot of the solutions and then skim through each solution set to find the minor differences.” One also mentioned that iBCM allowed them to explore “in depth as to how students could do.” A few also noted that “[iBCM] is useful with large datasets

where brute-force would not be practical". On the other hand, participants noted that BCM with re-groupings was useful "with small data sets and when they can brute-force through the submissions". One noted "It [BCM] really forced me to look through all the different solution groups and really read the code as opposed to skim it." Another noted that "with Tool B [BCM with re-grouping] I had fewer insights (but I was more sure of them)," and "I didn't find Tool B [BCM with re-grouping] more useful than Tool A in any situation. I didn't use the automatic reordering because I didn't like having no control over the groupings." The experiment shows the potential benefit of using iBCM for a real-world setting.

4.5 Conclusion

In this work, I introduced an interactive clustering model that can provide effective results for the user. iBCM communicates with users to incorporate their feedback into the clustering model and share its internal states. I showed that users were statistically significantly agreed that the final clustering matched their preferred way to cluster data points when using iBCM during human experiments. I also showed iBCM's potential for real-world use by implementing and validating a system to help computer science teachers to explore student homework submissions. Based on a Likert questionnaire, teachers strongly agreed (with statistical significance) that iBCM better supported them in exploring the variations in hundreds of student submissions, for the purpose of preparing homework review sessions.

Chapter 5

Conclusion and Future Work

In this thesis, I developed models and built a system to demonstrate how to enable humans to interact effectively with machine learning models to leverage the relative strengths of humans and machine learning models. The challenge addressed in this thesis is that traditional machine learning systems are not designed to extract domain expert knowledge from their natural workflow or to provide pathways for the human domain expert to directly interact with the algorithm to inject their knowledge or to better understand system output. I bridge this gap by building human-in-the-loop machine learning models and systems that compute and communicate machine learning results in ways that are compatible with human decision-making processes and that can readily incorporate human domain expert knowledge. In Chapter 2, I developed and validated a model that can infer human teams' planning decisions from the structured form of natural language of team meetings. In Chapter 3, I designed and validated an interpretable ML model that "makes sense to humans" by exploring and communicating patterns and structures in data to support human decision-making. Finally, in Chapter 4, I designed a machine learning model that supports transparent interaction with humans without requiring that a user has expert knowledge of machine learning techniques. I built a human-in-the-loop machine learning system that incorporates human feedback and communicates its internal states to humans using an intuitive medium for interaction with the ML model.

BCM extensions — hierarchical, multiple prototypes per cluster and non-parametric

For datasets that are better clustered and explained using a larger number of clusters, extending BCM to be hierarchical may offer better interpretability. Hierarchical BCM can learn sets of a prototype and corresponding subspaces for meta-clusters, then learn sets of prototypes and subspaces for the subsequent clusters. We can also apply non-parametric Bayesian techniques to BCM in order to infer the number of meta-clusters and sub-clusters. Another extension for BCM is learning multiple prototypes per cluster. Multiple prototypes would be useful if they can explain different aspects of the cluster. To learn multiple prototypes, BCM can include an additional criteria that encourages prototypes in the same cluster to be diverse, for example, using determinantal point process [87]. Learning a diverse set of prototypes may further improve humans' understanding of the cluster.

Potentially more scalable inference techniques for BCM

I apply Gibbs sampling inference technique to perform inference on BCM. However, there are other techniques that may scale better for larger datasets (e.g., variational inference [130], tensor decomposition methods [9]). Since not all distributions used in BCM are within the exponential families, the current BCM model does not offer closed-form updates when using variational inference. Modification or approximation of the BCM model may ease the derivation of variational inference.

Varying number of clusters and caching user's feedback in iBCM

The decomposition of Gibbs sampling steps used in iBCM could be further extended to vary the number of clusters in the model. For example, when a user suggests adding a cluster with a particular prototype, BCM can be updated to add the new cluster with the suggested prototype. Then, the cluster labels of data points that resemble the new prototype could be assigned to this new cluster while others remain unchanged. One of the challenges is to add the new cluster efficiently, especially when datasets are large. In this

case, subsampling the data points (i.e., randomly sample half of data points to investigate and rearrange their cluster labels) at the start may help to speed up the process.

The current iBCM framework updates the current solution to reflect user feedback. However, a user may want their feedback to be preserved for a longer period of time, not only for the current solution. This may be the case when a user wants to tailor iBCM's behavior through interaction for a specific application and repeatedly use iBCM. This can be done by either partially fixing a part of interacted latent variables to maintain the preference (e.g., if users indicated a particular feature to be a permanent part of a subspace, honor this by fixing the feature as such), or having an additional interacted latent variable that records the user's preference.

Make other models to make sense to humans

One future direction for research is to extend the insights learned from designing Bayesian methods that “makes sense to humans” to non-Bayesian methods and/or other widely used machine learning methods to make them more interpretable and interactable. Insights from the methods presented here that are inspired by the way humans think could applied to make an analogous module for other machine learning models. For example, a module can be added to existing models to learn explanations, by explaining the relationships between input data points and outputs (e.g., clustering results). This mirrors one of many possible ways humans try to understand highly complicated systems. Focusing on the relationships between inputs and outputs may offer a way to bypass the non-linearity of models (e.g., non-linear decision boundaries).

An extra component that learns explanations can be also added for existing models. For example, in addition to maximizing the margins around the decision boundary in support vector machines, an additional component could be added to its cost function that learns explanations such as prototypes and/or subspaces. One can also optimize in finding prototypes and subspaces that can minimize human confusion. For example, the model can find prototypes that could be explained using linear functions (i.e., the decision boundary is non-linear, but a linear function can be used to explain learned prototypes).

Build structure to models using previously developed techniques

The idea of guiding inference by combining traditionally different methods (using a logic-based approach in Chapter 2 and case-based reasoning in Chapter 3) shown in this thesis could be extended to other methods. Combining different methods is not only useful for harnessing the relative strength of different algorithms, but is also especially important for building “structures” for difficult inference problems (i.e., problems where solution space is big). For example, one can incorporate ontologies that are studied in classical artificial intelligence for knowledge representation into probabilistic models. Ontologies may offer a way to reflect relationships between variables (e.g., causal effect), building more structure in the model to aid inference while leveraging accumulated domain knowledge. This structure helps inference steps to potentially provide more informed decisions and potentially be more efficient. Utilizing tools previously developed by researchers in different field also enables us to build on each other’s work.

Interaction for debugging machine learning models

The experiments with iBCM suggest that there is subjective benefit in eliciting and utilizing user input to guide the solutions produced by machine learning models. This interaction could be extended to guide a machine learning model’s behavior to achieve desired performance. Currently, this is manually done by humans by adjusting the parameter settings of models and rerunning the machine learning model. An iBCM-like method could be used to help users explore the model’s behavior, for example, by investigating and interacting with examples of each class, or misclassified data points. Then a subspace-like interpretable medium could be used by users to modify the classifier’s behavior, ultimately to achieve the desired performance (e.g., low false positives). As more opportunities in data mining arise, more people may soon work with machine learning models regularly. Just as software debuggers are essential for boosting the productivity of software engineers, I believe that debugging tools for machine learning models will be developed and used widely in the future. Debuggers for machine learning models must overcome a number of challenges, such as stochasticity, model complexity, and the scale of the data. Enabling intuitive and

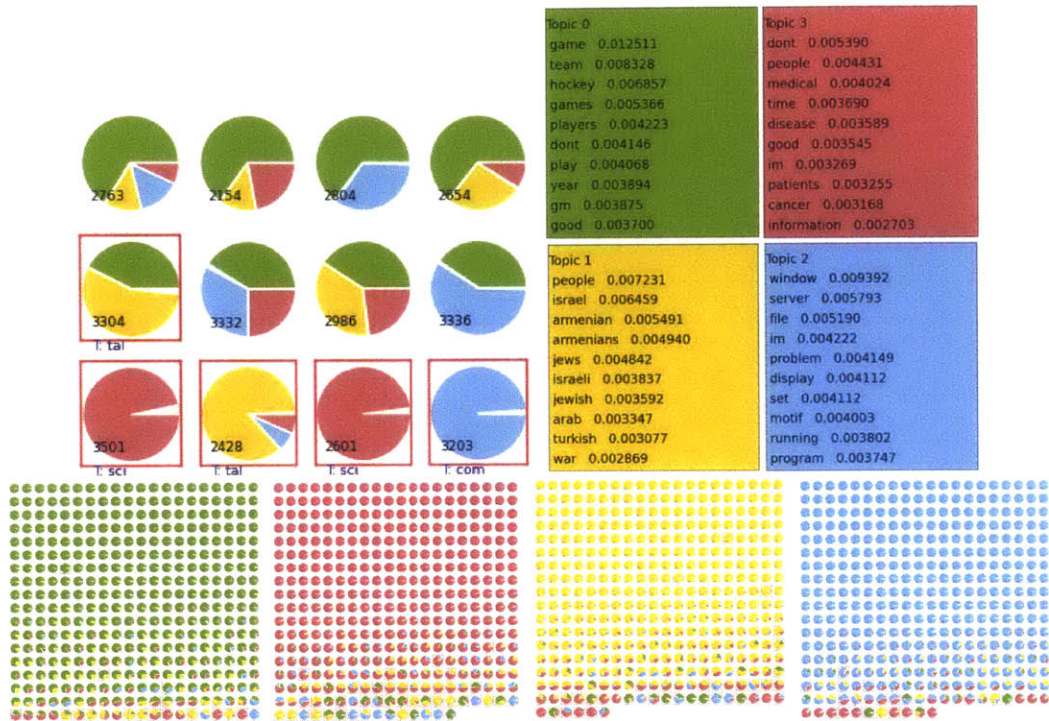


Figure 5-1: A portion of the analysis report using the data representation studied in this work. Each data point is compressed using topic-based feature compression. Zoomed in (top left), where a pie chart inside of a red square indicates incorrectly classified data points. The document ID number and true labels are also shown as texts. The top 10 words within each color-coded topic are shown at the top right. The bottom shows reports in a bird's-eye view.

informative interaction with machine learning models will be one of the key components for developing successful debugging tools.

Data visualization and representation

When working with real-world applications, a good medium of interaction is crucial and cannot be decoupled when working with interactive systems. In work related to this thesis, I have investigated data representations that aids users to interpret, evaluate and debug machine learning models. The work is part of an effort to provide analysis reports for those who use ML as a problem-solving tool and require an efficient way to interpret, evaluate and debug ML problems. The report provides information about individual data

points and their relationships with one another and with class labels. The information is presented in a way that allows for quick investigation of the status of classification tasks. Figure 5-1 presents an example report, where each pie chart represents one document (i.e. one datapoint). The bottom portion of Figure 5-1 shows reports in a bird's-eye view. A detailed view of a subset of points is shown in the top-left of the report, where misclassified data points are indicated by a red box. The compressed feature space representation allows users to more easily diagnose problems such as incorrect ground truth labels in cases when the misclassified point really resembles other points that belong to the predicted class. In order for such a report to be practical in a real-world setting, the data representation in question should not only be interpretable and scale with the number of data points, but also be capable of working with high-dimensional features. In this work ([81]), I showed that I can quantitatively and qualitatively verify an efficient, accurate and scalable feature-compression method using latent Dirichlet allocation that effectively communicates the characteristics of high-dimensional, complex data points. As shown in this work, the choice of data representation influences how accurately and efficiently humans can perceive the information in data. Developing an efficient and interactive data representation that is optimized for interactive setting is an important direction to pursue.

User's workflow and interface designs

Another important part of interactive systems is designing user workflow and the interfaces that aid interaction. Investigating how to optimize user workflow when using interactive machine learning systems may also be an important factor in determining the success of the system adaptation. Rigorous user testing would not only require carefully designing the interface, but also optimizing for the most informative degree of granularity and the form of data representation that is informative to the user.

Appendix A

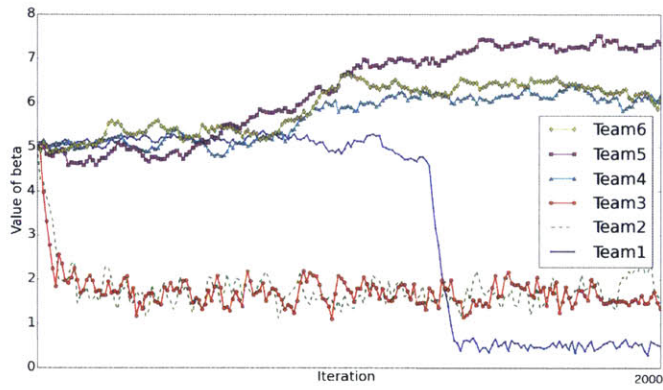
The visualization of Gibbs Sampling

Convergence: Trace Plot

It is known that there is no conclusive way to determine whether the Markov chain of the Gibbs sampling has reached its stationary, or the desired posterior, distribution [40]. Many available diagnostic tools are designed to test for necessary but insufficient conditions for convergence, such as Gelman-Rubin [54], Geweke [57], Heidelberger-Welch [67] and Raftery-Lewis [114], to mention a few. In this work we utilize a much simpler yet still informative approach, which is to visually check whether convergence has been reached using the trace plot.

A trace plot is simply a scatter plot of the statistics of successive parameter estimates (e.g., the estimated values) with respect to the iteration steps. These statistics can be means, variances or covariance. A trace plot is most informative when the scalar variables are plotted. Figure A-1 shows examples trace plots for the β and ω_p variables.

(a) Examples of β value convergences



(b) Examples of ω_p value convergence

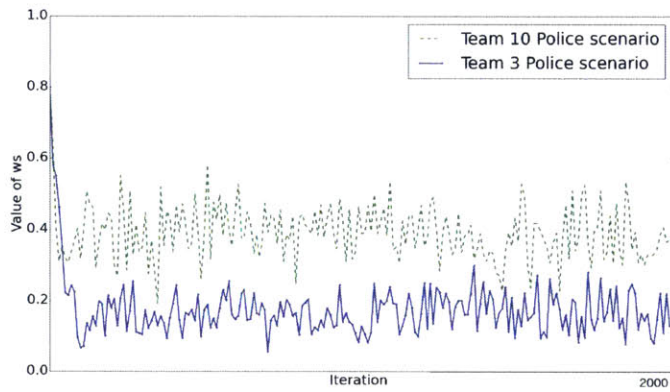


Figure A-1: Trace Plots (only showing a subset of the data set)

Bibliography

- [1] A. Aamodt. A knowledge-intensive, integrated approach to problem solving and sustained learning. *Knowledge Engineering and Image Processing Group. University of Trondheim*, 1991.
- [2] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 1994.
- [3] T.R. Addis. Towards an 'expert' diagnostic system. *ICL Technical Journal*, 1980.
- [4] D. W. Albrecht, I. Zuckerman, A.E. Nicholson, and A. Bud. Towards a Bayesian model for keyhole plan recognition in large domains. In *ICUM*, 1997.
- [5] S. Amershi, M. Cakmak, B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 2014.
- [6] S. Amershi, J. Fogarty, A. Kapoor, and D.S. Tan. Effective end-user interaction with machine learning. In *AAAI*, 2011.
- [7] S. Amershi, J. Fogarty, and D. Weld. Regroup: Interactive machine learning for on-demand group creation in social networks. In *SIGCHI*. ACM, 2012.
- [8] S. Amershi, B. Lee, A. Kapoor, R. Mahajan, and B. Christian. CueT: human-guided fast and accurate network alarm triage. In *CHI*. ACM, 2011.
- [9] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *JMLR*, 2014.
- [10] J. R. Anderson. A spreading activation theory of memory. *JVLVB*, 1983.
- [11] J. R. Anderson, D. Bothell, C. Lebiere, and M. Matessa. An integrated theory of list memory. *JOMAL*, 1998.
- [12] C. Andrieu, N. De Freitas, A. Doucet, and M.I. Jordan. An introduction to mcmc for machine learning. *ML*, 2003.
- [13] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.R. Müller. How to explain individual classification decisions. *JMLR*, 2010.
- [14] M.F. Balcan and V. Blum. Clustering with interactive feedback. In *ALT*. Springer, 2008.

- [15] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *KDD*. ACM, 2004.
- [16] S. Basu, D. Fisher, S. M. Drucker, and H. Lu. Assisting users with clustering tasks by combining metric learning and classification. In *AAAI*, 2010.
- [17] M. Bauer, S. Biundo, D. Dengler, J. Koehler, and G. Paul. PHI: a logic-based tool for intelligent help systems. *IJCAI*, 2011.
- [18] R. Bekkerman, H. Raghavan, J. Allan, and K. Eguchi. Interactive clustering of text collections according to a user-specified criterion. In *IJCAI*, 2007.
- [19] I. Bichindaritz and C. Marling. Case-based reasoning in the health sciences: What's next? *AI in medicine*, 2006.
- [20] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *AOAS*, 2011.
- [21] M. Bilenko, S. Basu, and R.J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML*. ACM, 2004.
- [22] D.M. Blei, T.L. Griffiths, and M.I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *JACM*, 2010.
- [23] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *JMLR*, 2003.
- [24] J.S. Brown and R.R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. *COGSCI*, 1978.
- [25] S. Carberry. *Plan recognition in natural language dialogue*. The MIT Press, 1990.
- [26] G. Carenini, V.O. Mittal, and J.D. Moore. Generating patient-specific interactive natural language explanations. In *SCAMC*, 1994.
- [27] J.S. Carroll. Analyzing decision behavior: The magician's audience. *Cognitive processes in choice and decision behavior*, 1980.
- [28] A. Cawsey. Generating interactive explanations. In *AAAI*, 1991.
- [29] A. Cawsey. User modelling in interactive explanations. *UMUAI*, 1993.
- [30] A. Cawsey, J. Galliers, B. Logan, S. Reece, and K.S. Jones. Revising beliefs and intentions: a unified framework for agent interaction. In *Prospects for Artificial Intelligence*, 1993.
- [31] A.J. Chaney and D.M. Blei. Visualizing topic models. In *ICWSM*, 2012.
- [32] J. Chang, J.L. Boyd-Graber, S. Gerrish, C. Wang, and D.M. Blei. Reading tea leaves: How humans interpret topic models. In *NIPS*, 2009.

- [33] E. Charniak and R.P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 1993.
- [34] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings algorithm. *The American Statistician*, 1995.
- [35] M.S. Cohen, J.T. Freeman, and S. Wolf. Metarecognition in time-stressed decision making: Recognizing, critiquing, and correcting. *Human Factors*, 1996.
- [36] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, 2003.
- [37] D. Cohn, Z. Ghahramani, and M.I. Jordan. Active learning with statistical models. *JAIR*, 1996.
- [38] A. Coles, M. Fox, K. Halsey, D. Long, and A. Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 2009.
- [39] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory*, 1967.
- [40] M.K. Cowles and B.P. Carlin. Markov chain Monte Carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association*, 1996.
- [41] P. Cunningham, D. Doyle, and J. Loughrey. An evaluation of the usefulness of case-based explanation. In *CBRRD*. Springer, 2003.
- [42] R. Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 1979.
- [43] G. De'ath and K.E. Fabricius. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology*, 2000.
- [44] R. Di Ciaccio, J. Pullen, and P. Breimyer. Enabling distributed command and control with standards-based geospatial collaboration. *HST*, 2011.
- [45] J. Eisenstein, A. Ahmed, and E. Xing. Sparse additive generative models of text. In *ICML*, 2011.
- [46] FEMA. Federal emergency management agency, 2014. [Online; accessed 3-December-2014].
- [47] A. Fiedler. Dialog-driven adaptation of explanations of proofs. In *IJCAI*, 2001.
- [48] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1972.
- [49] C. Fraley and A.E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal*, 1998.

- [50] A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations*, 2014.
- [51] Y. Gal, S. Reddy, S.M. Shieber, A. Rubin, and B.J. Grosz. Plan recognition in exploratory domains. *Artificial Intelligence*, 2012.
- [52] C.W. Geib and R. P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 2009.
- [53] C.W. Geib, J. Maraist, and R.P. Goldman. A new probabilistic plan recognition algorithm based on string rewriting. In *ICAPS*, 2008.
- [54] A. Gelman and D.B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 1992.
- [55] L. Getoor and L. Mihalkova. Learning statistical models from relational data. In *SIGMOD*. ACM, 2011.
- [56] J. Geweke. Bayesian inference in econometric models using Monte Carlo integration. *Econometrica: Journal of the Econometric Society*, 1989.
- [57] J. Geweke. *Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments*. Federal Reserve Bank of Minneapolis, Research Department, 1991.
- [58] A. Glass, D.L. McGuinness, and M. Wolverton. Toward establishing trust in adaptive agents. In *IUI*. ACM, 2008.
- [59] E.L. Glassman, J. Scott, R. Singh, and R.C. Miller. OverCode: visualizing variation in student solutions to programming problems at scale. In *ACM TOCHI*, 2014.
- [60] S. Goh and C. Rudin. Box drawings for learning with imbalanced data. In *KDD*, 2014.
- [61] A. Graf, O. Bousquet, G. Rätsch, and B. Schölkopf. Prototype classification: Insights from machine learning. *Neural computation*, 2009.
- [62] T.L. Griffiths and M. Steyvers. Finding scientific topics. *PNAS*, 2004.
- [63] J. Grimmer and G. King. General purpose computer-assisted clustering and conceptualization. *PNAS*, 2011.
- [64] B. J. Grosz and C. L. Sidner. Plans for discourse. In *Intentions in Communication*. MIT Press, 1990.
- [65] D. Guo, D.J. Pequet, and M. Gahegan. ICEAGE: Interactive clustering and exploration of large and high-dimensional geodata. *Geoinformatica*, 2003.
- [66] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 1970.

- [67] P. Heidelberger and P.D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 1981.
- [68] J.L. Herlocker, J.A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *CSCW*, 2000.
- [69] T. Hofmann. Probabilistic latent semantic indexing. In *ACM SIGIR*, 1999.
- [70] A. Holovaty and J. Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.
- [71] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. *UAI*, 1998.
- [72] R. Howey, D. Long, and M. Fox. VAL: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *ICTAI*. IEEE, 2004.
- [73] J.J. Hull. A database for handwritten text recognition research. *TPAMI*, 1994.
- [74] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *DSS*, 2011.
- [75] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1998.
- [76] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, 2000.
- [77] A. Kapoor, B. Lee, D. Tan, and E. Horvitz. Interactive optimization for steering machine classification. In *CHI*. ACM, 2010.
- [78] H. A. Kautz, R. N. Pelavin, J. D. Tenenbergh, and Morgan Kaufmann. A formal theory of plan recognition and its implementation. *Reasoning about Plans*, 1991.
- [79] H.A. Kautz. *A formal theory of plan recognition*. PhD thesis, Bell Laboratories, 1987.
- [80] B. Kim, C. M. Chacha, and J. Shah. Inferring robot task plans from human team meetings: A generative modeling approach with logic-based prior. *AAAI*, 2013.
- [81] B. Kim, K. Patel, A. Rostamizadeh, and J. Shah. Scalable and interpretable data representation for high-dimensional complex data. In *AAAI*, 2015.
- [82] J. Kim and J.A. Shah. Automatic prediction of consistency among team members' understanding of group decisions in meetings. In *SMC*. IEEE, 2014.
- [83] G.A. Klein. Do decision biases explain too much. *HFES*, 1989.

- [84] P. Koomen, V. Punyakanok, D. Roth, and W. Yih. Generalized inference with multiple semantic role labeling systems. *CoNLL*, 2005.
- [85] P. Krafft, J. Moore, B. Desmarais, and H.M. Wallach. Topic-partitioned multinet-work embeddings. In *NIPS*, 2012.
- [86] G.J.M. Kruijff, M. Janicek, and P. Lison. Continual processing of situated dialogue in human-robot collaborative activities. In *IEEE Ro-Man*, 2010.
- [87] J.T. Kwok and R.P. Adams. Priors for diversity in generative latent variable models. In *NIPS*, 2012.
- [88] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.
- [89] H. Lee, J. Kihm, J. Choo, J. Stasko, and H. Park. iVisClustering: An interactive visual document clustering via topic modeling. In *Computer Graphics Forum*. Wiley Online Library, 2012.
- [90] B. Letham, C. Rudin, T. McCormick, and D. Madigan. Interpretable classifiers using rules and Bayesian analysis. Technical report, University of Washington, 2014.
- [91] H. Li and J. Sun. Ranking-order case-based reasoning for financial distress prediction. *KBSI*, 2008.
- [92] L. Lin and M.A. Goodrich. A bayesian approach to modeling lost person behaviors based on terrain features in wilderness search and rescue. *Computational and Mathematical Organization Theory*, 2010.
- [93] J.R. Lloyd, D. Duvenaud, R. Grosse, J.B. Tenenbaum, and Z. Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *AAAI*, 2014.
- [94] K.E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 1998.
- [95] J. Mayfield. Controlling inference in plan recognition. *UMUAI*, 1992.
- [96] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL-the planning domain definition language. 1998.
- [97] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 1953.
- [98] J.D. Moore. *Participating in explanatory dialogues: interpreting and responding to questions in context*. MIT press, 1995.
- [99] J.W. Murdock, D.W. Aha, and L.A. Breslow. Assessing elaborated hypotheses: An interpretive case-based reasoning approach. In *ICCBR*. Springer, 2003.

- [100] Radford M. N. Slice sampling. *Annals of Statistics*, 2003.
- [101] A. Newell and H.A. Simon. *Human problem solving*. Prentice-Hall Englewood Cliffs, 1972.
- [102] H.T. Nguyen and A. Smeulders. Active learning using pre-clustering. In *ICML*. ACM, 2004.
- [103] T.A. Nguyen, S. Kambhampati, and M. Do. Synthesizing robust plans under incomplete domain models. *NIPS*, 2013.
- [104] M. Palmer, D. Gildea, and N. Xue. Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, 2010.
- [105] K. Patel, N. Bancroft, S.M. Drucker, J. Fogarty, A.J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *UIST*. ACM, 2010.
- [106] E.P.D. Pednault. Formulating Multi-Agent Dynamic-World Problems in the Classical Planning Framework. In *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann Publishers, 1987.
- [107] X. Phan and C. Nguyen. GibbsLDA++, AC/C++ implementation of latent dirichlet allocation using gibbs sampling for parameter estimation and inference, 2013.
- [108] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, 2006.
- [109] H. Poon and P. Domingos. Unsupervised semantic parsing. In *EMNLP*, 2009.
- [110] S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky. Shallow semantic parsing using support vector machines. In *NAACL-HLT*, page 233, 2004.
- [111] P. Pu and L. Chen. Trust building with explanation interfaces. In *IUI*. ACM Press, 2006.
- [112] D.V. Pynadath and M.P. Wellman. Probabilistic state-dependent grammars for plan recognition. *UAI*, 2000.
- [113] L.D. Raedt. Probabilistic logic learning. *Logical and Relational Learning*, 2008.
- [114] A.E. Raftery and S.M. Lewis. The number of iterations, convergence diagnostics and generic metropolis algorithms. In *Practical Markov Chain Monte Carlo*, 1995.
- [115] M. Ramirez and H. Geffner. Plan recognition as planning. *IJCAI*, 2009.
- [116] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 2006.
- [117] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. *UIST*, 1997.

- [118] A. Sadilek and H.A. Kautz. Recognizing multi-agent activities from GPS data. *AAAI*, 2010.
- [119] S. Schaal. Learning from demonstration. *NIPS*, 1997.
- [120] E.H. Shortliffe, R. Davis, S.G. Axline, B.G. Buchanan, C.C. Green, and S.N. Cohen. Computer-based consultations in clinical therapeutics: Explanation and rule acquisition capabilities of the MYCIN system. *Computers and Biomedical Research*, 1975.
- [121] P. Singla and P. Domingos. Markov logic in infinite domains. In *UAI*, 2007.
- [122] S. Slade. Case-based reasoning: A research paradigm. *AI magazine*, 1991.
- [123] F. Sørmo, J. Cassens, and A. Aamodt. Explanation in case-based reasoning—perspectives and goals. *AI Review*, 2005.
- [124] S. Tellex, T. Kollar, S. Dickerson, M.R. Walter, A.G. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.
- [125] A.L. Thomaz. *Socially guided machine learning*. PhD thesis, Massachusetts Institute of Technology, 2006.
- [126] R. Tibshirani. Regression shrinkage and selection via the lasso. *JRSS*, 1996.
- [127] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*. Springer, 2011.
- [128] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *JMLR*, 2002.
- [129] B. Ustun and C. Rudin. Methods and models for interpretable linear classification. *ArXiv*, 2014.
- [130] M.J. Wainwright and M.I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008.
- [131] M. Ware, E. Frank, G. Holmes, M. Hall, and I.H. Witten. Interactive machine learning: letting users build classifiers. *International Journal of Human-Computer Studies*, 2001.
- [132] R. Weida and D. Litman. *Terminological Reasoning with Constraint Networks and an Application to Plan Recognition*. 1992.
- [133] J. Whitehill. Understanding ACT-R - an outsider's perspective. *CoRR*, 2013.
- [134] S. Williamson, C. Wang, K. Heller, and D. Blei. The IBP compound dirichlet process and its application to focused topic modeling. 2010.

- [135] W.T. Wipke, H. Braun, G. Smith, F. Choplin, and W. Sieber. Secs-simulation and evaluation of chemical synthesis: strategy and planning. In *ACS Symp. Ser*, 1977.
- [136] J. Zhu, A. Ahmed, and E.P. Xing. MedLDA: maximum margin supervised topic models. *JMLR*, 2012.
- [137] H. H. Zhuo, Q. Yang, and S. Kambhampati. Action-model based multi-agent plan recognition. *NIPS*, 2012.