Abstract of "Algorithms for the Personalization of AI for
Robots and the Smart Home" by Stephen Brawner, Ph.D., Brown University, May 2018.

Just as an interconnected-computerized world has produced large amounts of data resulting in exciting challenges for machine learning, connected households with robots and smart devices will provide developers with an opportunity to build technologies that learn from personalized household data. However, there exists a dilemma. When limited data is available for a user, for example when they initially procure a new smart device or robot, there will be a substantial burden placed on that user to personalize it to their household by the learner. At the outset, applying predictions learned from a general population to a user will provide better predictive success. But as the amount of data provided by the user increases, intelligent methods should choose predictions more heavily weighted by the individuals examples.

This work investigated three problems to find algorithms that learn from both the general population and specialize to the human individual. We developed a solution to reduce the interactive burden when telling a robot how to organize a kitchen by applying a context-aware recommender system. Also, using the paradigm of trigger-action programming made popular by IFTTT, we sought to improve the programming experience by learning to predict the creation of programs from the user's history. Finally we developed several methods to personalize grounding natural language to these trigger-action programs. In a smart home where a user can describe to an intelligent home automated system rules or programs they desire to be created, their utterances are highly context dependent. Multiple users may use similar utterances to mean different things. We present several methods that personalize the machine translation of these utterances to smart home programs.

This work presents several problems that show that learning algorithms that learn from both a

general population and from personalized interactions will perform better than either learning approach alone.

Algorithms for the Personalization of AI for Robots and the Smart Home

by

Stephen Brawner

B. S., Harvey Mudd College., 2007

Sc. M., Brown University, 2014

A dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2018

This dissertation by Stephen Brawner is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Michael L. Littman, Director

Recommended to the Graduate Council

Date _____ _____

Maya Cakmak, Reader
(University of Washington, Computer Science)

Date _____ _____

Eugene Charniak, Reader

Approved by the Graduate Council

Date _____ _____

Andrew G. Campbell
Dean of the Graduate School

Curriculum Vitae

Academic

- **Ph.D, Computer Science**. Brown University. Providence, RI 2018
- **Sc.M., Computer Science**. Brown University. Providence, RI 2014
- **B.S., Engineering**. Harvey Mudd College. Claremont, CA 2007

Professional

- Software Engineering Intern, iRobot Coorporation. Bedford, MA 2015
- Software Engineering Intern, Bot & Dolly, Inc. San Francisco, CA 2013
- Software Research Intern, Open Source Robotics Foundation, Inc. Mountain View, CA 2012-2013
- Software Engineering Intern, Willow Garage. Menlo Park, CA 2012
- Consulting Engineer. Los Angeles, CA 2010-2011
- Project Engineer. eSolar, Inc. Pasadena, CA 2008-2010

Teaching and Mentoring Experience

- Instructor, Intro to Computation for Humanities and Social Sciences. Brown University 2017
- Graduate Teaching Assistant, Introduction to Artificial Intelligence. Brown University 2014
- Graduate Teaching Assistant, Human Robot Interaction. Brown University 2013
- Machine Shop Proctor. Harvey Mudd College 2005-07
- Undegraduate Tutor/Grader, Introduction to Computer Science. Harvey Mudd College 2005

Publications and Presentations

- David Abel, Edward C. Williams, Stephen Brawner, Emily Reif, and Michael Littman. "Bandit-Based Solar Panel Control". *IAAI 2018*, New Orleans, LA
- Stephen Brawner, Michael L. Littman. "Learning Household Organization via Context-based Collaborative Filtering". *IntRS@RecSys 2016*, Boston, MA
- Mark K Ho, James MacGlashan, Amy Greenwald, Michael L. Littman, Elizabeth M. Hilliard, Carl Trimbach, Stephen Brawner, Joshua B. Tenenbaum, Max Kleiman-Weiner, Joseph L. Austerweil. "Feature-based Joint Planning and Norm Learning in Collaborative Games". *CogSci 2016*. Philadelphia, PA
- Blase Ur, Sarah Mennicken, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Noah Picard, Diane Schulze, Michael L. Littman. "Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes". *CHI 2016*. San Jose, CA
- David Abel, Gabriel Barth-Maron, David Hershkowitz, Stephen Brawner, Kevin O'Farrell, James Macglashan and Stefanie Tellex. "Goal-Based Action Priors". *ICAPS 2015*. Jerusalem, Israel
- Stephen Brawner, Kevin O'Farrell, Lee Painton, Stefanie Tellex, and Michael Littman. "Coplanning via Inverse Reinforcement Learning". *NERC 2014*. Providence RI
- Stephen Brawner. "Converting SolidWorks Parts and Assemblies to ROS Friendly Files". *ROSCON 2013*. Stuttgart, Germany.

Open Source Software

- SW2URDF: bitbucket.org/brawner/sw2urdf
- Baxter H2R Packages: github.com/h2r/baxter_h2r_packages
- Collision Map Creator: bitbucket.org/brawner/collision_map_creator_plugin

Awards

- Brown Venture Prize Finalist 2018
- ICRA Mobile Manipulation Challenge. Accepted Participant 2012
- Brown University Fellowship. Brown University 2010-11
- Patent: WO 2010093876 A2, “Heliostat Field Cleaning System”, First Inventor 2010
- Deans List. Harvey Mudd College 2003-07
- NASA Reduced Gravity Student Flight Program 2005-06

Preface and Acknowledgements

It is an ambitious task to acknowledge all the people who have made this work possible. Research is a collaborative process and there are numerous people and groups who have helped me along the way. To all those that have helped in both big and small ways toward my research progress, Thank You.

The person most singularly helpful toward my research has been my advisor, Michael L. Littman. In his role, he has helped me understand this meandering process called computer science research. He has also been an excellent sounding board for ideas (good and bad), and has been deeply helpful thinking through the small details as well as the larger picture. I have observed time and time again Michael illustrate how to not only be a productive researcher, but a friendly, caring presence among his collaborators. Lastly, it cannot be overstated the amount of direct mentoring he has provided specifically with regard to puns.

The people most responsible for my capacity to even endure the burdensome process of the PhD have been my family: my brother Eric, my sister Christie and my parents John and Lorrie. Eric and Christie have long served as role models, whom I strive every day to live up to the examples they have set. My parents have been unhesitatingly supportive of me throughout this marathon and provided overwhelming support and love.

I can credit the wonderful group of friends I have made at Brown to the high quality of life I have enjoyed while in Providence, without which this degree would not have been possible. My first-year

roommates and the first friends I made in Providence, Jon Mace and Patrick Heck, have become some of my best. Without a doubt, living with them opened me to some of the best people that came through Brown. Chris Tanner, John Oberlin, Nakul Gopalan, Carl Trimbach, Jun Ki Lee and countless others made for great support inside the CS department. Andrew Lynn, Brittany Baxter, Lexi Kriss, Sara Miller, Jacob Paul, Cara Kuczarski, Megan Reilly, David Mély, and many more made for great support outside of the department.

Research is of course not a singular endeavor, and robotics is especially collaborative. It is frustratingly difficult to get robots to do anything. Without the shared knowledge and support provided by all the people of Brown University Robotics, this research would not have been successful. Stefanie Tellex, and Michael Littman and more recently George Konidaris have built a maximally productive and collaborative lab, with great people and robots. I am especially grateful to experienced lab sages that shared what they knew: Chris Crick, James MacGlashan, Peter Haas and Tom Sgouros.

Perhaps the group that I can never thank enough are all the members of the staff of Brown Computer Science. Though their names never appear on an author list, they have made my life in this department so much easier. Lauren Clarke, Eugenia DeGouveia, Lori Agresti, Suzanne Alden, have all been especially helping with my various departmental needs and navigating the university bureaucracy. Likewise, the technical staff have all been instrumental in ensuring that we all have technical resources we need and that they work flawlessly.

I can credit the numerous internships I have been fortunate enough to do as providing the software development and industrial experience that has given me a huge advantage now that I am re-joining the real world. At the Open Source Robotics Foundation and also Willow Garage, I want to thank John Hsu, Brian Gerkey and Nathan Koenig for providing a one of a kind opportunity to a robot novice. At Bot & Dolly, I am thankful to be able to work with Kendra Byrne and Ian Sherman where I discovered a completely different side and potential for robots in our lives. At iRobot, I would like to thank Timothy Field, John Wang and Jarad Cannon for an excellent experience learning how to

develop on production-level robots (i.e. robots that actually work).

Lastly, I would like to give a large amount of credit and acknowledgment to my mixed-breed yellow lab, Taylor. The PhD process is a long journey and is sometimes lonely. Finding Taylor in my third year helped me immensely through it. Introducing a daily routine of walks, exercise and training helped me find a productive outlet far outside of the university. She has been an excellent companion and there is nothing more helpful to solving your research quandaries than a nice long walk with a dog companion. It is the recommendation of this researcher that everyone find themselves a companion animal.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The accelerating progress of robotics and household devices will lead to a period of high adoption of smart machines in the home. Just as an interconnected computerized world has produced large amounts of data resulting in exciting challenges for machine learning, connected households with various robots and smart devices will provide developers with an opportunity to build technologies that learn from our households.

The challenge inherent in the abundance of personalized information is that much of this information is not digitized. In household tasks, this non-digital context includes not only the household's complete state, but also the physical and mental state of all the household members. Even by placing sensors throughout our smart home, systems can observe only a limited set of these hidden states. Designing smart devices and robots for the home will likely require user input and feedback to properly adapt to the household.

Smart devices and robots have three approaches to adapting to a household: they can learn, they can be trained, and they can be programmed. On one end, devices that learn significantly reduce the burden on the user, but allow for less direct corrections. On the other end, programming requires the user to understand the language of the device. Training devices is a middle ground that compromises between these two sides.

Systems that learn fully autonomously are generally difficult to develop for robots. They require the robot to collect proper inputs and accurate feedback. If either the input or the feedback are perceived incorrectly, then the learning algorithm will likely build an incorrect model. For a commercial product, there may also be security concerns that also exist in autonomously collecting data around a household.

Programming smart devices and robots on the other hand significantly reduces the amount of information about the user that has to be shared. End users can explicitly input their preferences and modify the programming and configurations until they achieve the desired behavior. However, there is a trade-off between the expressibility of a programming language and the ease with which it can be learned and used. End-user programming typically focuses heavily on simplifying the adaptability of the language by using only a limited high-level programming functionality.

Training robots and smart devices is a compromise between learning and programming. Both the user and device take an active roll in transferring the intention of the user to the device, resulting in a combination of the best and worst of learning and programming. The device takes up a substantial burden of learning the user's intentions, however, either a poor trainer or a poor learner can degrade knowledge transfer.

1.1 Generalized Learning in the Home

The uses of machine-learning systems in everyday living are numerous. Asking for directions from a modern smartphone's embedded AI presence (Siri, Google, Cortana) may utilize a broad swath of machine-learning algorithms from fields like Natural Language Processing and Computer Vision. Many internet-connected devices continuously collect examples to improve the underlying prediction models. However, their focus is typically on improving prediction success broadly instead of improving success for any specific end user.

For example, at the time this dissertation was prepared, Amazon's Echo has been leading a small,

but growing product category of household personal assistants. Google, Microsoft and Apple are other companies producing a similar product. Essentially, these devices are wireless speakers with an embodied artificial intelligence agent (Alexa, Google, Cortana, Siri) thoroughly integrated with the internet and the company's special services.

These devices provide a semi-personal assistant, which is supplied with some personally identifying information at the initial setup. Even though a home user might interact with the device through numerous interactions over a long time period, the devices do not currently learn from these interactions and all queries are processed by a central server.

Amazon's Echo processes language in a general pattern-matching format. Amazon's NLP processing servers first recognize that a query like "Play Baba O'Riley by the Who" matches a familiar pattern of query: "Play $SONG by $ARTIST". It will then select the phrase in between the query words 'play' and 'by' and match that to songs under the collection of songs of the artist specified by words in the $ARTIST phrase. These systems work remarkably consistently, but lack the ability to understand other contexts. For example, the query "Play my favorite song" throws these systems into a tailspin. They just do not learn from repeated interactions.

These devices are continuously learning and their AI is constantly improving. This learning happens only to the generalized machine-learning models they are using to understand utterances and commands. They do not currently learn to understand their own user.

While these sorts of limitations are currently acceptable for devices that have limited functionality in interacting with one's household, the introduction of highly capable household robots and household devices will require the ability to learn from previous experience to allow for shorthand commands and abbreviated interactions. It might be acceptable to tell a robot where each individual item is placed the first time it cleans the living room, but this will be unacceptably redundant even by the second time.

(a) (b)

Figure 1.1: **a**: The original setback thermostat developed by Minneapolis Heat Regulator Company in 1906. With just a mercury thermometer, a temperature setting and a clock, the user set the time at which the central heating of a house was to change its setpoint temperature. The user manually turned down the temperature at night to reset. Image credit: [8] **b**: The Honeywell Chronotherm III. The first digitally programmable thermostat. Three decades of confusing programming settings has since followed.

1.2 Programmable Smart Household Devices and Cautionary Tales

People have had the opportunity to interact with and program their 'smart' devices in their household for quite some time. Originally, a smart device was something that included a bit of preprogrammed automation. Programmable remote controls, programmable thermostats, and programmable VCRs are three examples of 'smart devices' that all serve to illustrate the point that including the ability to personalize a device is not sufficient. Additionally, it must also not be burdensome to individualize the device.

Programmable Thermostats

The first programmable thermostats were developed by Minneapolis Heat Regulator Company (eventually merging with Honeywell) in 1906 (Figure 1.1a). These thermostats featured a mercury thermometer, a temperature-setting adjustment and a mechanical clock. The thermostat user would turn down the temperature at night, and the thermostat would turn it up to the programmed temperature at the set time. Not long after, they added a second clock to automatically turn down the temperature at night. The mechanically programmed thermostat was perfected in design with the Round developed by the Minneapolis-Honeywell Regulator company and introduced to the public in 1953 (though originally designed in 1941) [5].

Since then, we have seen the development of digital programmable thermostats, which have been called "smart" since the introduction of Honeywell's Chronotherm III in 1986 (Figure 1.1b). Though various marketing efforts have been promising great reductions in heating bills with programmable thermostats, recent studies have raised doubt as to whether people save more money on utilities by programming a thermostat or by manually changing the temperature in their house [52]. It turns out that the challenging nature of programming these thermostats negates much of the significant benefit to the automatic temperature changes.

Programmable Videocassette Recorders

During the much remembered Betamax/VHS war of the late 1970s and 1980s, a revolution to television viewing was introduced. A year after the original Betamax was sold in Japan and the US, Sony brought out the SL-7200 Betamax deck with optional timer (Figure 1.2). This allowed people, for the first time, to record a television program without being physically present at their TV, enabling people to watch shows on their own schedule and not at the time prescribed by the television networks. What followed was a lesson in interface design for the ages[2].

Most people who owned a VCR have experienced the 'Blinking 12:00' problem. VCRs with a digital internal clock allowed home users to program a set recording time with the press of a couple buttons. However, any loss in electricity to the home reset the clock back to 12:00 and required people to reset the clock and timer. Despite the simple one or two button interaction, people struggled to remember the interaction or just rarely bothered to fix the problem. Thus, one of the most revolutionary ideas introduced in these devices was rendered powerless by households that were themselves rendered powerless.

Figure 1.2: An advertisement for the Betamax SL-7200, the first VCR capable of recording at a programmed time. Image credit: [1]

Programmable Universal Remote Controls

Though remote controls had been around since the early 1950s, the idea of a single remote to control all your audio and visual devices was not introduced until the preprogrammed universal remote was produced by Magnavox. With a marketing slogan of "Magnavox controls Sony", their product had a large influence on universal remotes to this day [9].

The first programmable universal remote, called CORE, was introduced by the company CL9 founded by Steve Wozniak in 1987 [6] (Figure 1.3). It allowed a lot more flexibility for the home user by allowing them to program the remote to control any device controlled by IR signals. Unfortunately, it was not much of a commercial success, and people have been struggling to program universal remotes ever since.

Programming a universal remote controller required pointing the remote that is to be mimicked at the programmable remote. The user then assigned a single button's infrared signature to one of the universal buttons. It was a long process and required assigning each button on each remote one

Figure 1.3: The inside of the product box for the CL 9 CORE. One of the first programmable universal remotes. This remote could remember 4096 different infrared signals and the user could also program buttons to remember sequences of signals. It did not achieve much commercial success, in part due to its difficult usability. Image credit: [3]

at a time.

Universal remotes continue to solve the problem of managing too many physical remote controls. Universal remotes come in two flavors, the preprogrammed remote with a large but ultimately limited database of programs and programmable remotes that learn from existing remote controls. Today, though, if your phone supports IR, you can just install a smartphone application [10] to achieve the same functionality. In a way, a smartphone is the ultimate universal controller.

Consumers have had access to programmable devices now for decades, so we can carry forward lessons learned during these trying times to household learning devices and robots. If these devices do not have high reliability and accessibility to a diverse end-user population, adoption will be weak (universal remotes), adoption of their 'smart' features will be weak (VCRs) or efficiency savings will not be obtained (thermostats).

1.3 The Burden of Personalization

As discussed in Section 1.2, programming household devices is burdensome and can inhibit adoption. This same issue exists from learning devices that require any input from the user. Providing the capability of personalization is not sufficient. Even if a household device or robot is capable of learning everything it must know from the user, the size of burden on the user will be pivotal in whether the user chooses to adopt the technology.

The effort required on the part of the user to tailor a device to their own household or person is referred to as the *burden of personalization* throughout this dissertation. Users who are overwhelmingly burdened by the process of personalization will default to generic and out-of-the-box starting configuration of their smart devices. These are the perpetually 'Blinking 12:00' users.

It is one of the main goals of this dissertation to offer solutions to reducing the burden of personalization. While specific solutions are unique to the problem they belong to, the solutions we offer provide insights into solving this problem in a broad set of contexts.

The key strategy to reducing this burden is to blend population-level learning and personalized learning. Without any information about user, the system should utilize what it knows from the general population. As it learns, it can blend in the information about the user.

The goal is to start from the general solution, but use the provided instance corrections to better understand user in the space of users we already understand. Can their examples be better predicted by exploiting the examples provided by a specific cluster of users?

The goal for any smart household device or robot is that its learning needs to adapt and to converge to 100% reliability. Devices that learn from general populations and do not adapt to the household will remain frustrating to use. The only solution for these devices is to learn from interactions in the home—post deployment.

1.4 A Combined Model of Personalized and Population Level Learning

Picture a household robot that is programmed to clean and organize a household. Fresh out of the box, it will have no idea where objects are located around the home. A smart robot might first drive around the home, analyze each object and memorize its location. But, the location of an object at any given time may or may not be the user's desired location. Also, its precise location may or may not need to be so precise. The coffee cup that was a gift from a friend several years ago has an appropriate cupboard and shelf, but it does not necessarily need to be placed in front of that new Star Wars mug and to the right of the NPR mug. If the robot were to ask the user the precise location of every object in the household, it would encumber the user to the point of frustration.

A smarter robot can learn natural patterns of organization from general data. Mugs are placed on a shelf together, plates of the same kind are stacked on top of one another. In this manner, the robot does not need to query the user for all objects in a household, but just those that do not fit the learned general patterns. In Chapter 3, we discuss the household organization problem in greater detail. There, methods commonly used in recommender systems are adapted to the grouping of objects in the household.

For the other research in this dissertation, we look into the problem of simplifying and aiding the programming of automated households. Trigger-action programming is a model of end-user programming that breaks up programming tasks into an event trigger and an action performed by the device. The website ifttt.com (IFTTT) is widely used for connecting and programming smart devices in the home and interfacing internet services. While there are over one million different recipes on the site, prior research has found that only a small subset are actually built and used by the site's users [65]. In fact, only a small fraction of the created recipes shared on this website are actually unique.

Chapter 4 presents results from applying various collaborative filtering methods to recommend

trigger-action programs to the end-user from their programming history. Program suggestions and code completion is a widely used tool for typical programmers. Our work explores the ability to provide recommendations to household users for TAP programs.

To further explore solutions to combining the benefits of personalized and generalized learning, Chapter 5 focuses on translating natural language utterances into actual household TAP programs.

Programs to automate the household require context specific to the home in which they operate. They manipulate specific devices particular to the human user, and each household will consist of a different set of devices. However, users in two entirely different households may give very similar natural language utterances to write different programs. Two users using two different automated lighting devices, Philips Hue and WeMo for example, would likely prefer to tell such a system "Turn on the lights when I arrive home". However, the household system needs to understand which room of lights to turn on, and even which specific lighting service to use. A personalized system would not have sufficient information to understand common language groundings, but a generalized one might learn groundings specific to other users.

There exists a significant burden to personalize devices in the home so we as technology developers need to imbue our devices with as much common sense as possible when attempting to learn from individual users. Common sense has to be learned from a broad dataset covering many users. My goal for this dissertation is to provide solutions to several problems in this space as well as spur interest and direction into other problems of this type.

Learning algorithms for robots and smart homes perform better when they trade-off learning from both generalization of a large population and individualization through repeated user interactions than either strategy alone.

Chapter 2

Background to Personalization in AI and Robotics

2.1 Personalization in Household Robotics

The personalization and tailoring of a robot's behavior to its user's household will be an important component in future household robots. Though we do not currently see many robots that could potentially make use of this style of learning, the research community has begun to look at fundamental personalization problems that will enable these capabilities.

Human-robot interaction has been an important field of research for roboticists that draws from human-computer interaction, human factors engineering, and psychology. However, it has long studied singular interactions with robots. Leite et al. [38] found that only in about the last 10 to 15 years have researchers been capable of performing longitudinal studies of repeated interactions of humans and their robots. Therefore, exploring the question of how to design robots to best learn and adapt to their human users has only recently been investigated.

Dautenhahn [24] proposed a framework that may allow household robots may be able to personalize and adapt in the home. Building on our understanding of how we have domesticated canines and their ability to adapt to different households and different work tasks, the author suggests that it may be beneficial to design our household robots to adapt in a similar manner. Their framework consists of an imprinting step, similar to a puppy's first several months of development, where the key behaviors are developed and strongly ingrained.

Utilizing such a framework, different stages of imprinting could happen between the factory construction and later stages in the household. The first stage could feasibly be completed at its production facility, to train the robot with essential first-stage behaviors. Later stages of development, where the robot learns and adapts to its household and adoptive family, would occur in its household setting. This makes the important realization that much of a robot's understanding of the world will likely come from real-world interaction, but it is unclear from the paper why first-stage learning cannot be learned in one master robot, and copied to new robots.

Prior research in robotics personalization has generally fallen into three overlapping categories: making interactive systems friendlier and more social by providing them with the appearance of a personalized or social component, customizability of a robot's behavior or appearance to the user's preferences, and the personalization of interactions over repeated encounters [37].

Researchers at Carnegie Mellon Univesity, led by Manuela Veloso, have developed CoBots, which are a class of robot with a strictly limited set of capabilities that require interaction with and interventions by humans to accomplish the robot's own goals [66]. A design principle in this project is that robots robots may never be able to fully autonomously navigate a human world, they instead enable the researchers to investigate the sorts of interactions necessary to illicit the best response from human partners. This line of HRI research has led to important studies of personalized robot behavior.

In the context of CoBots, Lee et al. [37] found that a robot that adapts its own interactions with a human will be received better. These researchers programmed snack-delivering CoBots to

personalize their dialogue with human users based on previous interactions. Such interactions helped users feel much closer to the robot. Users that experienced the personalized interactions used the robot's name more often and some even presented the robot with gifts.

Baraka and Veloso [15] enabled a similar snack-delivering CoBot to personalize its behavior according to user input. In this work, the robot was provided explicit ratings from the user about the predictive snack choices it made, and the robot was able to fit the user to a specific model, which traded off the amount of novelty used to suggest a snack choice.

Frequently, personalization is built completely from scratch through repeated interactions with users, resulting in duplicated efforts, as different users are required to train the same base skills. For example, Mason and Lopes [44] developed a robotic system that accomplished tasks according to instructions from users and eventually learns how to perform the task semi-autonomously. This system personalizes to the preferences of the user by building user profiles with regard to how the human user prefers specific tasks to be performed. These user profiles were built as a database of good and bad end states. However, the original task, still had to be taught de novo by all users.

Leyzberg et al. [39] found that personalizing tutoring robots can lead to improved learning outcomes over non-personalized tutors. Their robot tutor helped participants solve a grid-based logic puzzle by teaching them puzzle-solving strategies. It helped personalize the tutoring by providing relevant strategies at set times throughout the session. Beyond choosing relevant strategies to the game state at the time they were delivered, the assessment algorithm in their system chose the relevant game state strategies that were most helpful for the participant's skill level. They found over a 50% reduction in solving time by the participant's fourth puzzle over the control conditions. Gordon et al. [29] also studied personalizing a robot tutor, in this case to teach pre-K students vocabulary from a second language. They were able to significantly improve engagement using a personalized tutor over a non-personalized version.

To enable rich interactions to train robots and smart devices, new methods will need to learn from large populations to provide complex responses and behaviors. However, there do not exist

many algorithms that leverage learning from large populations that also provide personalized and tailored interactions.

2.2 Recommender Systems

Perhaps the most relevant research area when discussing melding personalized and generalized learning is the field of recommender systems. Not specific to any one particular algorithm or class algorithms, recommender systems describe a broad category of tools that draw the user's attention to a subset of items from a larger set of items based on that user's historic usage. Depending on the input and methods employed, recommender systems may be divided into three categories:

- Content-based: Recommended items are selected based on their contextual similarity to the user's history
- Collaborative: Items recommendations are informed by other users with similar usage history.
- Hybrid: A combination of content-based and collaborative

Though recommender systems share goals similar to the fields of information retrieval, they emerged as a separate field in the 1990s when recommendations could be easily quantified through the use of numerical ratings [14].

Today, recommender systems enjoy immense popularity primarily to encourage exploration of web and social media services. Amazon recommends products based on a user's purchase and browsing history. Media companies, like Netflix, Hulu and Spotify, recommend new TV shows, films, or music from a user's viewing, listening and rating history. Social media apps like Facebook, Instagram, LinkedIn, recommend new connections that have close network ties to the user's current connections. News companies, like Google News, and New York Times, are willing to trade at least some editorial content control for a user's demonstrated news topic interests.

2.2.1 Content-based Recommender Systems

With content-based recommender systems, it is assumed that some contextual information exists with each item being recommended. In the case of movies, it might include the director, actors, genre or, in the case of Netflix, a highly specific subgenre ('Movies for ages 0 to 2' [7], 'Deep Sea Horror Movies' [4], 'Understated Independent Coming-of-age Movies' [11]).

Content-based recommender systems share a strong resemblance to information retrieval (IR) and in fact are somewhat born out of that field. In one popular method borrowed from IR, frequencies of keywords are measured across documents and normalized according to each keyword's highest frequency. Keywords that appear in a large fraction of documents are downweighted as they are not useful for distinguishing between documents. For each user, a vector of weights are calculated from their own document history. A similarity score (for example, cosine similarity) of a user's weight vector and a document's weight vector then serves as a rating of that document's utility for the user [14].

Though these methods are suitable for finding relevant items related to a user's history, they do not score the quality of the document or item. So items of low quality may be recommended because of their high relevancy to the user's history. These methods also suffer with regard to novelty. They cannot recommend items with very different keywords or features regardless of how widely appreciated they are. The new Hawaiian Poké restaurant opening up across from your department building would not be highly recommended unless you had enjoyed Poké in another city [14].

As is an inherent issue to most recommender systems, a user with a short or nonexistent history would not have an accurate profile, and cannot receive accurate recommendations. However, because content-based recommenders do not generally account for the quality or general likability of an item, it cannot even recommend a generic set of items without sufficient number of ratings from that user [14].

2.2.2 Collaborative Filtering

Collaborative filtering is a category of methods for recommender systems that seek to predict ratings for items novel to a user from ratings of items from other users. The typical input is a rating matrix $R \in \mathbb{R}^{M \times N}$, where rows are associated with the items and the columns are associated with users.

There exist two main types of collaborative filtering techniques, memory based and model based. Memory-based methods compute recommendation ratings from the entire collection of ratings provided by previous users. Though statistically satisfying, the large amount of computation and memory required to compute these recommendations is difficult to scale to large numbers of users and items [14].

Model-based solutions attempt to derive an approximate model from the provided ratings. For example, a simple but reasonably performing model computes rating for a user u for item i from a combined sum of biases:

$$r_{u,i} = \mu + b_i + c_u \tag{2.1}$$

where μ is an average across all ratings R

$$\mu = \frac{1}{R} \sum_{u=1}^{U} \sum_{i=1}^{I} r_{u,i} \tag{2.2}$$

b_i is *bias* for item i

$$b_i = \frac{1}{U} \sum_{u=1}^{U} (r_{u,i} - \mu) \tag{2.3}$$

c_j is then the bias for user u

$$c_j = \frac{1}{I} \sum_{i=1}^{I} (r_{u,i} - \mu - b_i) \tag{2.4}$$

As with content-based methods, there still exists a new user problem and a new item problem. However, collaborative filtering methods will revert to the mean without sufficient data for either a user or an item.

Factorization methods attempt to generate a lower dimensional representation to improve generalization. Singular value decomposition is one example that performs well with typical recommending problems [17].

2.2.3 Hybrid Recommender Systems

Hybrid recommender systems are a class of methods designed to alleviate the problems encountered by content-based and collaborative filtering based methods. By combining the benefits of both models, one can link items together with the content-based contributions and link together the users with the collaborative approaches [14].

Hybrid methods typically take one of four approaches to combining content-based and collaborative based methods:

- Combine the ratings from a content-based recommender and a collaborative recommender
- Incorporate content-based characteristics into a collaborative recommender
- Incorporate collaborative-based characteristics into a content-based recommender
- Construct a general model employing both content-based and collaborative based characteristics

2.3 Recommender Systems and the Cold Start Problem

As discussed in the previous section about recommender systems, an inherent drawback is the difficulty in providing suitable recommendations for new users or recommending novel items to existing users. This challenge is known in the field of recommender systems as the *cold-start problem*.

The cold-start problem is especially relevant in the field of household robotics as they are introduced into homes.

This dissertation argues that personalization is a core necessity to household robotics and smart home systems, so the cold-start problem is an inherent hurdle to personalization in robotics. In fact, if we could solve the cold-start problem in its entirety, then personalization would not be necessary: we could treat every user as a new user and expect the same performance. But naturally, there will always be some effort required to personalize our robots and smart devices, but the goal of this work is to minimize that effort by leveraging learning from the general population.

The goal in resolving the cold-start problem is providing recommendations to new users that quickly tailor themselves, using the short history of interaction, to the preferences of the user. This problem can be addressed in several ways. Treating the population of users as belonging to separate clusters can allow methods to quickly assign a user to specialized recommendations. Alternatively, if some demographic or contextual information is provided about the user, a system can associate that user with the populations of users to which they belong. With memory-based collaborative methods, several proposed solutions involve using an user-similarity metric that better handles novel users than baseline methods.

2.4 Personalization in Medicine

The field of medicine has perhaps been one of the oldest research areas to benefit from scientific thinking. It has long been leveraging the study of populations to benefit the individual. Medical research evaluates the benefit of drugs, interventions, and procedures on a selected population of individuals, but the goal of clinical treatment has been to maximize the outcome for each individual.

This focus on clinical trials has resulted in a profound understanding of the human body, but has left the actual practical clinical treatment to be somewhat of learned art. Recently, the advent of personalized medicine has gained a strong research focus and it provides interesting lessons for

the hope of improving the personalization of AI in the home.

I have had similar difficulty finding AI algorithms for personalized medicine that balance the trade-off between personalized learning and general learning. As this field evolves there might be a large number of insights that can be integrated across disciplines.

Often in personalized medicine, the focus is on developing a treatment strategy as opposed to singular treatment instances. Similar to how personalized robotics focuses on back-and-forth interactions with robots, these personalized medicine studies try to improve the patient's outcome over a series of treatments. Due to the sequential nature of these problems, framing these problems in the context of reinforcement learning (RL) has been shown to be particularly effective. Many algorithmic solutions in RL in particular can effectively balance the trade-off between a knowledge gathering action (does a X ml dosage of Y drug have a positive or negative response) and the currently best known response (A ml dosage of B drug), or as it is known in the RL community, exploring and exploiting.

Marivate et al. [43] has carefully evaluated the ability of batch reinforcement learning to identify treatments and evaluate their potential effectiveness on individuals. Similarly, Ernst et al. [27] use reinforcement learning to develop drug therapy treatment schedules for HIV patients, which bypass the complications of developing sophisticated ODE models of HIV infection dynamics.

Pineau et al. [53] developed adaptive treatments strategies for substancesabuse patients using instance based reinforcement learning.

Zhao et al. [68] found that an optimal policy can be extracted from a set of patient treatments and their responses. They utilize Q-learning, a temporal-difference reinforcement learning algorithm to identify the optimal policies from clinical data. This style of learning handles a fundamentally difficult challenge in personalized medicine: giving weight to interventions that may have either or both short-term and long-term effects. Unfortunately, Q-learning lacks the ability to easily transfer from one environment to another, which means that this particular solution cannot incorporate learning from other patient data to improve learning for a given individual.

Utilizing reinforcement learning, Hochberg et al. [31] find they can improve activity levels in diabetes patients through personalizing the delivery of reminder messages. In this system, the messages fell into several categories, negative feedback, positive feedback related to the patient's own exercise plan, and positive feedback comparing their exercise results to other patients. They chose to use a contextual-bandit algorithm, which finds the optimal actions given different contextual information but cannot find more complicated sequence of actions like Q-learning.

Personalizing medicine through the use of artificial intelligence, and in particular reinforcement learning, provides promising benefits to patients' clinical treatment. However, general knowledge is currently incorporated only explicitly through the design and formation of the possible solution set. This field remains an interesting case that may benefit more from learning methods that can properly trade-off learning from both the general population and individuals.

Chapter 3

Learning User's Preferred Household Organization via Collaborative Filtering Methods

3.1 Introduction

As robots become common household items, people will be required to train their robot on details specific to their household and lifestyle. In the context of household automation, smart devices like the Nest thermostat can learn a user's preferred heating and cooling schedules from tracked manual inputs [46]. Compared to smart devices, robots will need to learn significantly larger amounts of household information, putting a substantial burden on the user for training. As this *burden of personalization* becomes more commonplace in home automation and robotics, users may find themselves investing more and more time into training their smart devices and robots. We are interested in reducing the burden of personalization by leveraging learned information from other households.

The challenge lies in the inability to directly transfer learned information from one household to another, due to uniqueness of household designs, schedule, demographics and composition of the household members, and the combined preferences of all users interacting with the system. Therefore, it is important to identify certain contextual facets of the learning problem to work on. We consider one such personalization problem in the area of household robotics: the problem of a robot tasked with learning the organization of items in a user's kitchen to support tasks like putting away groceries or unloading the dishwasher. Asking for the location of every item would be time consuming—possibly more so than having the person put away the items themselves. Therefore, we were motivated to design an algorithm that leverages the organization of other users' kitchens to help predict the user's organization of items.

In the context of videos, products, and news articles, collaborative filtering techniques used for recommender systems have been developed to transfer the experience of a population of users to help identify items a target user would find desirable. Typically, these techniques predict ratings of user–item pairs, that is, how would a given user rate an item based on their past ratings [58]. Recently, *context* about users and items has been incorporated into recommender systems to improve their performance—a crucial element for our application.

Context-aware collaborative filtering, for example as presented by Panniello et al. [49] and Rendle et al. [57], assumes that more information exists that predicts or informs a user's choice of ratings than just the item itself. In the case of household organization, the context can include the set of possible locations, the identities of the items known to be in those locations, or categorical information related to the items and locations. For this application, we envision a robot with sufficient perceptual abilities to identify these pieces of context.

One of our proposed methods employs a context-based recommender system, factorization machines (FMs) [55], to predict a user's item–location ratings—the degree of acceptability of the placement of a given item at a given location in a kitchen. In contrast, we also present an alternative collaborative filtering method that predicts locations based on the item's predicted item–item ratings

to items already placed in that location, which builds directly on prior work [12]. The important difference is, while the previous collaborative filtering technique required explicit item–item ratings, the FM method enables direct predictions of item–location ratings, which can even be boosted by learning on the contextual features of these variables.

We present data collected from participants on Amazon Mechanical Turk who annotated kitchens with locations of items. We describe several methods for enabling a learning robot to predict item–location ratings for novel users from this data.

We also discuss how these ratings can be used in an interactive system that attempts to find a trade-off between asking the user for the correct input and making a wrong choice. We demonstrate results on a collection of user-annotated, simulated kitchen examples.

3.2 Related Work

Researchers have long argued that user interfaces can benefit from learning from their users [42]. However, devices in the home like the Nest thermostat [46] require a significant amount of learning effort. Yang and Newman [67] reported that early users struggled with understanding what the Nest thermostat learned and that these users found it hard to override its learned behavior.

Home organization is a good example of a task that both elicits strong user preferences and is highly desirable to automate. Several researchers found that cleaning and organization are the top two tasks users most want robots to do [61, 22]. Pantofaru et al. [50] argue that, unlike cleaning tasks, organization is "nuanced and emotional." They also found that, even if they could afford the help, many people will not hire human assistants to organize their belongings because it is too personal of a problem. Therefore, a robot learning this task must tread lightly by weighing the costs of requiring too many user interactions and of misplacing items.

Several researchers in the robotics community have begun to look at placing and organizing items. Cha et al. [23] examined methods for predicting locations of items in a user's kitchen from

other items in the kitchen. Using item-related features like item type and use, they found that a Support Vector Machine classifier (SVM) performed the best in their domain.

However, in collaborative filtering, SVM methods are often disregarded due to their poor performance on sparse datasets. We present as a baseline a Support Vector Regression method that struggles on the full data set when considering individual locations.

Fisher et al. [28] presented a probabilistic model that can generate plausible scenes of items from user-provided examples. Leveraging a larger scene database, they can create arrangements of novel items suitable for the user. Collaborative filtering methods extend beyond this type of generative model by incorporating preferences of other users to predict a given user's preferences. Jiang et al. [32] discuss a method for placing items optimizing for stable and semantically relevant locations, but do not capture a user's preferences for locations among semantically identical locations. Schuster et al. [60] learn organizational principles to place items into meaningful locations, but do not make use of user preferences to select locations. However, a robot employing our algorithms could also easily incorporate these capabilities of identifying free space and stable placement poses within the specific location suggested by our prediction system.

The work by Abdo et al. [13] is closest to the contributions of our paper. They use collaborative filtering techniques on item–item pair ratings and then approximately solve a minimum k-cut problem to best group the items from their predicted ratings. Using these groupings, they then place the items into semantically identical bins or shelves. We build on this work to address the problem of predicting concrete locations for items in a kitchen instead of only general groupings. We present several methods to solve this more central problem, including a method for adapting their technique for using collocation data to predict suitable locations. We further present a context-aware collaborative filtering technique that leverages item–item collocation information without its explicit representation in the training data, and an active learning method that further capitalizes on the interactions with the user.

3.3 Method

Our proposed solutions predict a user's preferred locations for items they want organized via two components: a rating-prediction collaborative filter and an algorithm for producing the optimal location given a set of location ratings produced from the collaborative filter.

There are three solutions we present here. The first builds on previous work by using a collaborative filter to predict a user's item–item ratings—whether the two items should be placed together or not. Predicting the correct location is a matter of choosing the location that contains the item with highest item–item rating to the item being placed for the evaluated user. The second uses a context-aware collaborative filter to directly predict a user's item–location ratings. The predicted location is just the location with the minimum item–location rating for the user.

The third solution capitalizes on the robot's back-and-forth interactions with its user. It adopts an active learning perspective, choosing the set of items with the weakest predicted ratings by the FM model and queries the user. It then retrains its model to better incorporate the demonstrations.

3.4 Recommender Systems

Recommender systems are concerned with the problem of predicting a user's item ratings. That is, given user u, what rating $r \in \mathbb{R}$ would they assign to the item i? For the problem of household organization, we adopt a variant of the classical recommender problem in which users provide ratings on either item–item pairs or item–location pairs and the system predicts item–location ratings.

Collaborative filtering is a category of methods for recommender systems that seek to predict ratings for items novel to a user from ratings of items from other users. The typical input is a rating matrix $R \in \mathbb{R}^{M \times N}$, where rows are associated with the items and the columns are associated with users. Factorization methods attempt to generate a lower dimensional representation to improve

generalization. We follow Abdo et al. [13] and decompose this matrix into:

$$R = B + \bar{R}, \tag{3.1}$$

where B is a bias matrix that is computed to encode the global bias, item bias and user bias, and $\bar{R}$ is the residuals matrix that the collaborative filtering algorithm then attempts to learn. We use the L-BFGS minimization algorithm [41] to find the factorization of the residual matrix and bias matrix that minimizes the regularized squared error.

3.4.1 Context-aware recommender systems

Compared to classical recommender systems, context-aware recommender systems assume that some additional information exists that relates to the user's choices of ratings. For example, a user may choose a location based on the other items already placed there or based on the location's salient features. We include information about the locations into the context-based recommender, but our chosen context-aware recommender also learns automatically information related to item–item collocation to inform its predictions.

3.4.2 Factorization Machines

Rendle et al. [55] developed factorization machines to make predictions in a model with all multi-degree interactions among the context variables with sparse data in linear time. For a system that models only interactions between two variables at a time, the model equation is:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\mathbf{v_i} \cdot \mathbf{v_j})\, x_i x_j, \tag{3.2}$$

where $\hat{y}$ is the predicted rating, $\mathbf{x}$ is the input variable vector, w_0 is the global bias, and $\mathbf{w}$ is the weights over $\mathbf{x}$. The quantity $(\mathbf{v_i} \cdot \mathbf{v_j})$ is the dot product of $\mathbf{v_i}$ and $\mathbf{v_j}$ and models the interaction of

the i-th and j-th variable, where $\mathbf{v_{i/j}} \in \mathbf{V}$ is a factorized representation of the weighting of $x_{i/j}$.

Instead of estimating an individual weight parameter ($w_{i,j} \in \mathbb{R}$) for higher-order interactions ($d \geq 2$) between variables x_i and x_j, the factorization of $\mathbf{v}$ enables modeling even under sparsity. Here, w_0, $\mathbf{w}$ and $\mathbf{V} \in \mathbb{R}^{\mathbf{n} \times \mathbf{k}}$ are the parameters to be estimated.

Though we show that item–item collocations have explanatory power through the context-unaware method, we do not need to represent that information in the context variables. Instead, we include users, items, locations and features, and the optimization of the model will seek to explain these ratings appropriately.

Though the model presented in Equation 3.2 suggests a solution would require a run time of $\mathcal{O}(kn^2)$ where k is the number of dimensions and n is the number of context variables, Rendle et al. show this equation can be reformulated into a linear solution with runtime complexity $\mathcal{O}(kn)$. Indeed, for models with higher degrees of interactions, it can still be refactored into a linear time solution.

3.5 Location Selection

One difficulty of choosing a correct location for an item is that many locations in a kitchen are interchangeable and are justifiable placement locations *a priori*. A user's preferred organization therefore requires the robot to identify the most likely location from existing item placements and physical suitability of the location for the item.

For our work, we code ratings as 1.0 for a positive example and 0.0 for a negative example. Thus, for item–location pairs, a rating $r(i, l) = 1.0$ indicates the item i is in the preferred location l, whereas $r(i, l) = 0.0$ indicates it is not. Item–item pairs are coded analogously. Rating predictions produced by the collaborative filtering methods are then real values approximately in the range of $[0, 1]$.

3.5.1 Choosing a Location from Item–Item Pairs

In our first approach, item–item ratings $r(i, i')$ are learned from training input via a context-unaware collaborative filtering technique [13]. Finding a suitable location for an item in the kitchen then requires finding the location that results in the best match to the item–item ratings. If a location is empty, the method takes the mean pair rating as its default.

Formally, for items $i, i', j, j' \in I = \{i_1, i_2, \cdots, i_m\}$, and location $l \in L = \{l_1, l_2, \cdots, l_n\}$ where $i \neq i'$ and $j \neq j'$, n is the number of locations and m is the number of items, we want to find the item–location rating $R_{\text{item–item}}$ summarized from item–item ratings $r(i, i')$:

$$R_{\text{item–item}}(i, l) = \begin{cases} \max_{i' \in l} r(i, i') & \text{if } \exists i' \in l, \\ \frac{1}{m^2} \sum_{j \in I} \sum_{j' \in I} r(j, j') & \text{otherwise.} \end{cases} \tag{3.3}$$

We denote the event of item i being placed in location l as $i \in l$. We then select the location with the highest item–item rating and write the selected location as

$$\hat{l} = \underset{l}{\operatorname{argmax}}\, R_{\text{item–item}}(i, l). \tag{3.4}$$

3.5.2 Predicting Location Ratings through FMs

We use the factorization machine's model to predict item–location ratings. Each row in the input matrix includes context variables for the user, item, location, and a location category. For each provided example of an item–location pairing in the input data, we assign a rating of 1.0. We produce $n - 1$ other ratings of value 0.0 for this same item in all the other locations in the kitchen. Each location variable is encoded uniquely for the user, even if—in experiments—different users annotated the same kitchens.

In Equation 3.5, we show the context variable vector we used in our dataset. Each context variable is assigned a value of 1.0 when active and 0.0 otherwise. In our notation, u is a user, i_i is

an item, l_j is location, and $\Phi(l)$ are the feature values over the active location variable l:

$$\mathbf{x} = \{u_1, \cdots, u_U, i_1, \cdots, i_m, l_1, \cdots, l_n, \Phi(l)\}. \tag{3.5}$$

Because the model directly learns item–location ratings $R_{\text{item–location}}(i, l) = r(i, l)$, we can find the location with the highest pair rating:

$$\hat{l} = \underset{l}{\operatorname{argmax}}\, R_{\text{item–location}}(i, l). \tag{3.6}$$

3.5.3 Active Learning with FMs

For a robot engaging in an interactive scenario with a human, it could feasibly query the human for specific probes. The Factorization Machines method presented above enables the robot to identify items that fail to achieve a satisfactory threshold cutoff in terms of the predicted rating for a location. By choosing the items with the lowest predicted location rating, the robot could ask the user for the locations of these items and add their results to the training data and re-run the model training. Via this manner of active learning, the robot can significantly improve over randomly placed probes.

Specifically, given a user, a set of m probes, and a set of n items to place in a kitchen, we chose k items whose predicted locations through this active learning method for $m - k$ probes received the lowest ratings, and moved them to the training data $(m + k)$. We then re-ran the model training, and tested on the remaining items $(n - k)$. The 0 probe data was the same data used for the 0 probes evaluation as the FM model. There exists a trade-off in choosing k between the predictive success and the number of times to rerun the training, $\lceil \frac{m}{k} \rceil$.

3.6 Data Collection

To create a data set with maximum opportunities for generalizing between users, we were interested in finding a collection of items that many people would be familiar with and would have in their households. As such, we ran a study to identify the most common kitchen items.

3.6.1 Finding common kitchen items

To create our item set, we began by pulling a list of recommended cooking items from 'How to Cook Everything' by Bittman [18]. Their recommended items consisted of cooking tools and basic foods useful for cooking a variety of recipes. We also examined the list of items identified by Cha et al. [23] in their CMU kitchen dataset. They surveyed several households and annotated all the items found in their participants' kitchens. We merged the items across these sources and removed

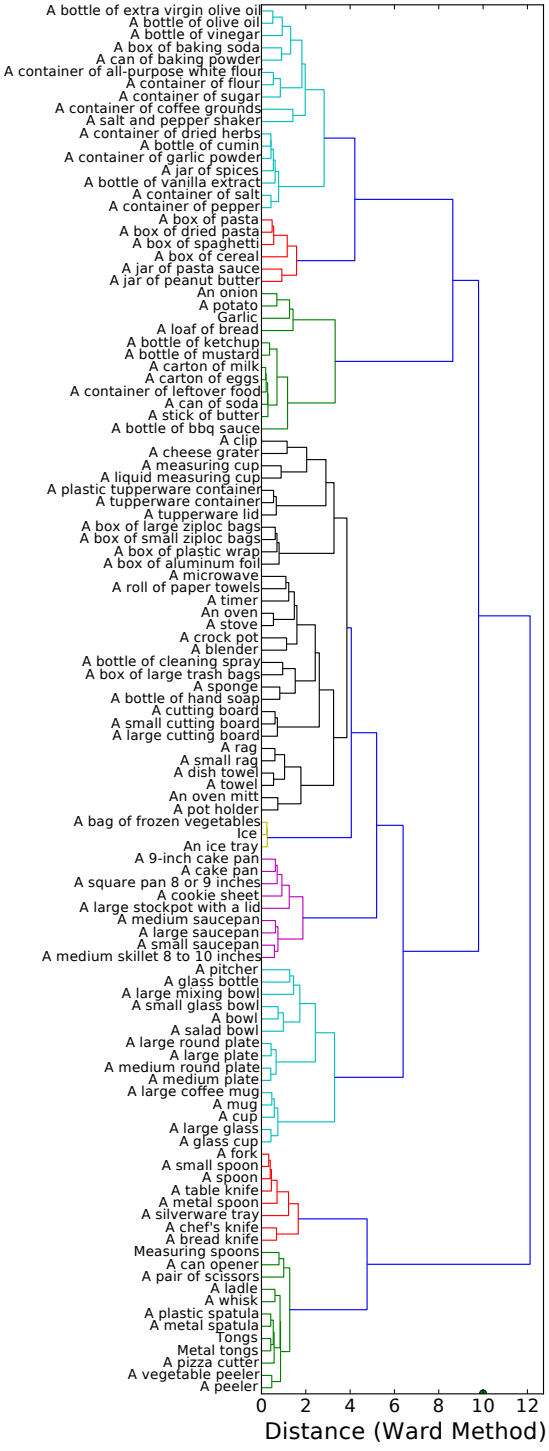


Figure 3.1: A hierarchical clustering of 111 kitchen items and their mean pair ratings. Groups were colored for distances less than or equal to 4

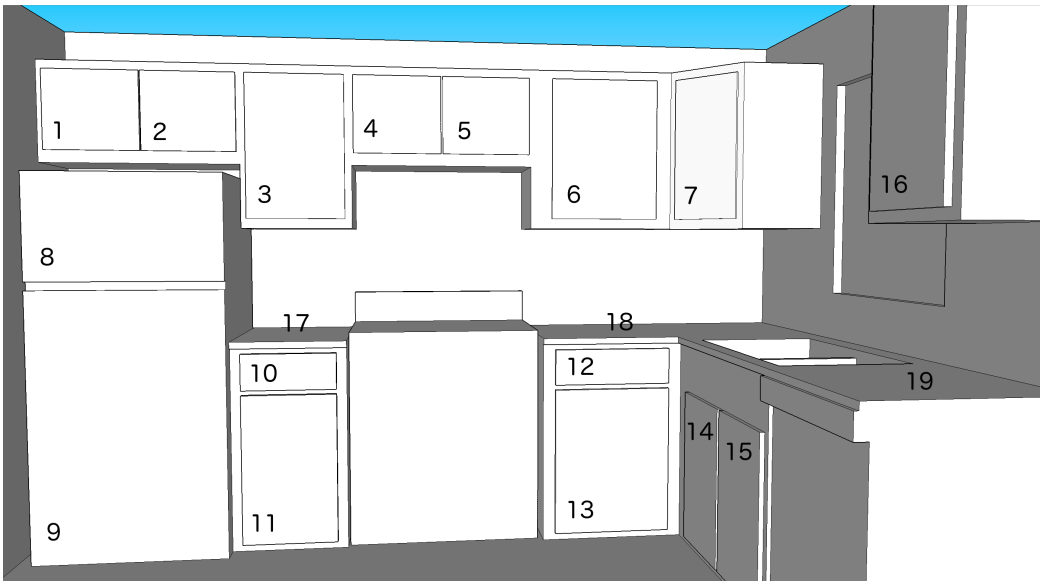


Figure 3.2: Diagram of a kitchen with numeric labels.

duplicates—items that we judged to be the same as an item already in the list—and were left with 398 items. We replaced items commonly found in a container, like seasonings, oils or other liquids, with the appropriate container of that item ('olive oil' became 'a bottle of olive oil').

We randomly split the selected items into surveys of 8 to 10 items, and asked participants on Amazon Mechanical Turk if they had any of the items in their household kitchen. For each survey, we collected 20 responses. Participants were allowed to answer as many different surveys as they chose (min: 1, max: 30, mean: 10.5). No one participant completed all the surveys.

From the full set, we selected the items at least 85% of respondents indicated were in their households, producing a list of 111 items to be used for location annotations.

3.6.2 Kitchen annotations

We were provided four kitchen layouts from the authors of the CMU Kitchen Dataset. We labeled each drawer, cabinet, refrigerator, freezer and counter with a unique numerical identifier. Figure 3.2 shows one of the labeled kitchen images we provided for a survey. We created four different surveys, each with a different labeled kitchen. We asked participants to assign locations in their labeled

kitchen for all 111 items. Participants were asked not to respond to more than one survey. For the participants who took multiple surveys, we only kept responses to their first. We received 25 responses to each survey. We removed three responses from two groups due to duplication of users. Ten participants from the previous task also completed this one.

To summarize patterns in how objects were placed closely to one another, we built the hierarchical clustering of item–item pair distances shown in Figure 3.1. Over all users, and for each item–item pair, we computed the fraction of times the items were not placed together. Groups of items with distances less than or equal to 4 are given unique colors in the diagram. The figure illustrates that items fall into clear categories in their placement around the kitchen. For example, 'a glass cup', 'a large glass', 'a cup', 'a mug', and 'a large coffee mug' were commonly placed together.

3.7 Experiments

From the data presented in Section 3.6.2, we created four sets of training and testing data by assigning data collected for three kitchens to training and the fourth to testing. Even though many people annotated the same kitchen, we used a variable for a location unique to each user. For each item and location in a kitchen, we created a collocation dataset by indicating for each item–location pair in the original data set whether it was found with one of the other 111 items. That is, we changed the data lines from rating–item–location to rating–item–item–location. We set the associated rating to 1.0 if the two items were found together in the location and 0.0 otherwise.

For the item–location dataset, probes were drawn from the testing data. For each user in the testing data, we selected a random set of items and moved the ratings of that user for those items to the training dataset. We used the same sets of items for the collocation dataset and created a set of probes for all item pairings in this set of items.

3.7.1 Model configuration

We trained the factorization-machine model (FM) using libFM [56] on the training data to produce predictions for each user–item–location in the testing dataset. We used adaptive stochastic gradient descent with a learning rate of 0.03, a sampling standard deviation of 0.1 and 30 for the number of dimensions of the model. The item–location tuple with the highest predicted rating was chosen as the predicted location of the item.

For the active learning method, we utilized the base FM method above. We retrained the model every $k = 5$ probes.

For the data-driven factorization method (DD), we trained the collaborative filter on the collocation training dataset without the additional probes. The probes were then used to update the model through the new-user update method presented by Abdo et al. [13]. As done in their work, we also built our model with three dimensions for the factorization. The limited memory BFGS minimization algorithm from SciPy [33] was used to find the optimal variables, with a stopping criteria 'factr' set to 10.

We included a support vector regression (SVR) model to predict ratings from the dataset. The SVR model was trained on the item–location data set. We used the implementation provided in scikit-learn [51]. The user columns of the item–location dataset were removed to reduce sparseness. We used a radial basis function kernel, with the default gamma of $\frac{1}{N}$ where N is the number of features. The stopping tolerance was set to 0.0001, and no shrinkage was used.

We trained both SVR models and one FM model with features. For both, we included features about the locations describing the location type: drawer, low cabinet, high cabinet, counter, refrigerator, freezer. Additionally, for the SVR models, we also included item features related to the use of the item: edible, drinkable, food storage, etc.

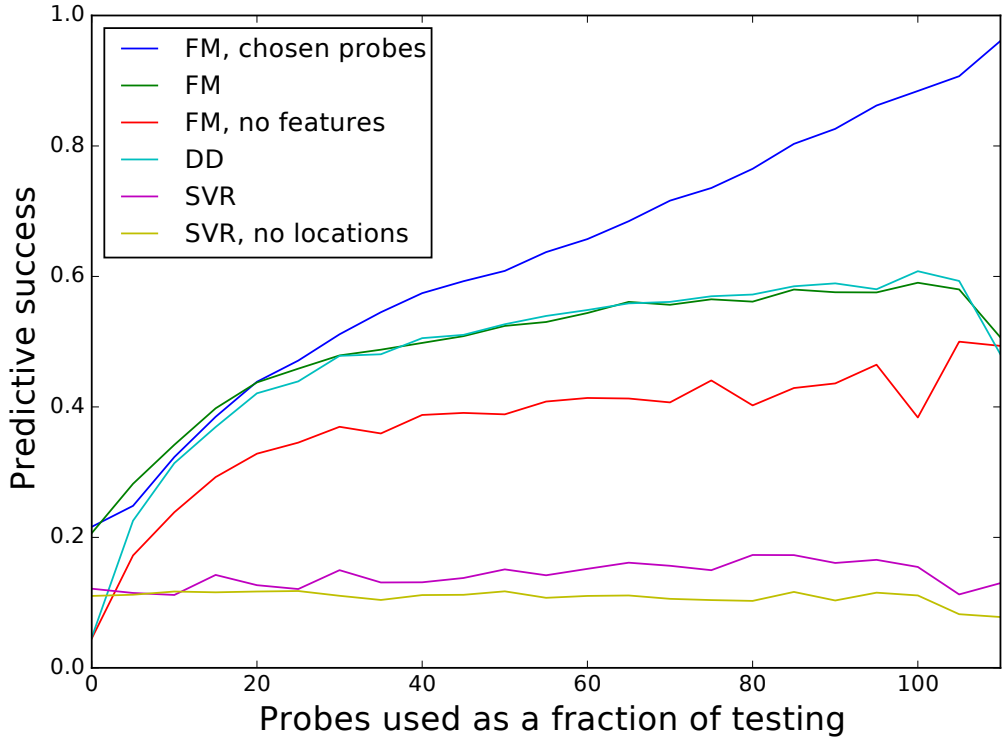


Figure 3.3: Prediction accuracy versus the number of probes for several different learning methods.

3.7.2 Results

Figure 3.3 presents the predictive success of the non-probe items in the testing dataset for two FM models, a data-driven factorization model over item collocation data (DD), an active-learning method with FMs (FM chosen probes), and two SVR models. We measured predictive success as the fraction of times the model predicted the correct location a user chose to place an item.

For the two FM models, we present a model trained as we described in previous sections (FM), and also a model trained on these variables but without location features (FM, no features). For the two SVR methods, we show the results of one model trained using location variables (SVR) and one without (SVR, no location). We show results probes ranging from 0 to 110 out of 111 items.

As seen in this graph, the proposed methods perform quite well for this difficult task. Even with no information about a current user (0 probes), the FM method and the 'FM chosen probes'

can correctly predict a significant fraction of locations for that user. The 'FM, no features' and the DD methods successfully leverage probes when available, but degrade to random guessing when no probes are used. This result is due to the limitations of collaborative filtering without additional contextual information. Without any information from the user, the base performance is generally quite poor.

The FM models exhibited strong performance over a wide range of probes. The success of the DD method to predict locations when probes were available illustrates the importance of the inherent item–item affinities presented earlier.

The active learning method significantly outperformed the FM and DD models above 20 probes. This method achieves 96% accuracy when allowed to choose 110 probes and retrain the model every 5 probes. Though it is highly unrealistic that a robot would be able to query for 110 out of 111 items, the results strongly support the use of an active learning methodology.

Both SVR methods performed worse than the proposed solutions regardless of the number of probes presented. The SVR method trained with locations performed slightly better than without locations. Regardless, the collaborative filtering methods solidly outperformed the SVR methods.

3.7.3 Placement-versus-asking trade-off

An important design consideration for an interactive system is the trade-off between placing an item incorrectly and asking the user for the correct placement. We observed that each algorithm places higher confident answers closer to the extrema of the ranges of their output. Therefore, we can enable systems to ask questions for uncertain items by requiring a threshold for ratings.

Figure 3.4 plots the success of each algorithm against the fraction of items in the testing dataset that do not meet the threshold for placement with 20 probes already placed.

Data points at a fraction of 1.0 correspond to data presented in Figure 3.3 for 20 probes. Data for fractions below 0.15 are particularly noisy due to the low number of items that were available for evaluation.

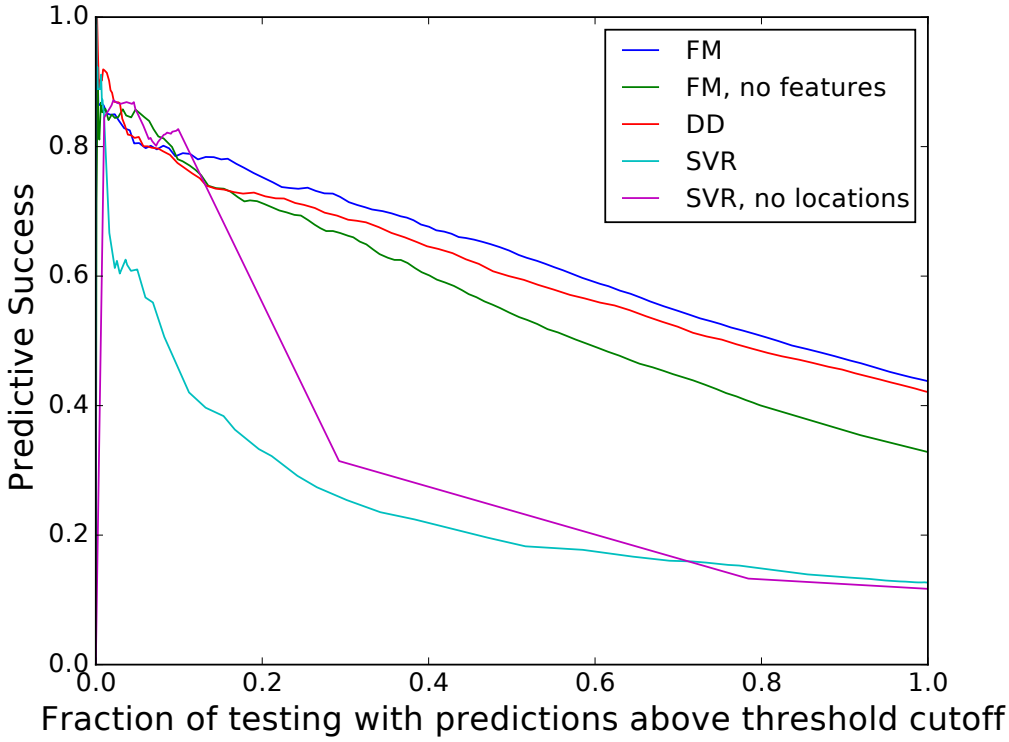


Figure 3.4: Predictive success versus the fraction of testing above the minimum required threshold for placement.

These results illustrate why the active-learning method performs so strongly above. By choosing items with weaker predictions to query the user, the systems can improve its overall predictive success. This active learning method was not included in this graph because it already queries the user to retrain its model, and therefore this process is incompatible with that method.

3.8 Conclusions

In this chapter, we presented several methods for reducing the burden of personalization that comes from users supplying a learning system household organization preferences. We collected data about the existence, pairing and location of items in participants' kitchens. A system that recognizes a location suitable for an item could easily still fail to match the user's preference for its location. In response, we presented a solution that incorporates a user's preferences but enables an autonomous

system to make use of perceivable features of these locations when learning.

We showed that a context-based collaborative filtering model called factorization machines is well suited to predict item–location ratings. Choosing the appropriate location is a simple task of finding the location with the best rating. For interactive systems, we showed there is significant improvement when preferentially querying about items with poorer predictions.

Though household robotics entails a sizable burden of personalization, this issue also plagues many other modern smart-household devices like thermostats and lighting. Machine-learning methods that use data from other households in learning a user's preferences will require smart recontextualizations that can map the preferences drawn from a collection of different users onto the user's own household space. We envision these topics as bright areas of future research.

Chapter 4

Trigger-Action Programming Recipe Prediction and Personalization

4.1 Introduction

With the proliferation of smart devices and robots in households, a growing number of end-users will desire to control their devices in their home through interaction and programs. End-User programming has been a long explored area of research to help users with this problem. Trigger-action programming (TAP) is one specific paradigm in the area of smart devices and internet services. A trigger-action program contains two program components, an event trigger and an action. When the event trigger fires, the action is signaled to occur.

On the web, IFTTT (If-This-Then-That) carries on this style of simple trigger-action programs and contains support for multitudes of smart devices and internet services [64]. At the time of writing, IFTTT supports 370 channels. 302 channels support a total of 1316 triggers and 255

channels support 774 actions, which means there are over 1 million different recipes that can be created. This overwhelmingly large number of possibilities provides the user with great automation power, but produces another problem. It becomes overwhelmingly impossible to explore this space of possibilities. The limited programming space requires a complete understanding of all the channels available at IFTTT, and new users would find difficulty understanding the components available.

In fact, in Ur et al. [65], we found that this space is largely unvisited and people prefer to program with only a subset of the available choices. From a dataset collected in 2015, we found that only 6% of the possible combinations were actually created. We also found that there was a high skew among the popularity of channels. Some channels like Instagram, Feed, Weather, and Email were used in tens of thousands of recipes, while numerous others were used only a handlful of times or not at all.

Meanwhile, numerous IFTTT channels reproduce functionality of others but integrate with different services or devices like emails, texts and switching devices. Therefore, when programming one's own house, it is highly likely that only a specific set of channels, triggers and actions are even usable.

This work seeks to produce a recommender system capable of providing TAP authors with personalized recipe component recommendations. We have already begun this investigation and have developed several algorithms for predicting user's more likely recipe components. The proposed work seeks to investigate how to improve the incorporation of recipe's context into the predicted ratings.

4.2 Related work

The problem of predicting end-users programming components is similar in concept to the idea of code completion in integrated development environments (IDE). Code completion is the problem of suggesting code snippets to finish out a line that the programmer is currently working on. Some

versions perform automatic suggestions, others require manually requesting code completion proposals. Code completion has recently garnered significant research attention [21],[48], [47]. Murphy et al. [45] found *Content Assist* in the Eclipse IDE was used by 100% of the programmers they surveyed.

Robbes and Lanza [59] found that utilizing the program's recent code history significantly improved code completion proposals beyond suggestions made solely by type. That is, methods and classes that were recently created or used should receive priority in the proposal rankings. However, it is difficult to transfer these methods completely to trigger-action programming suggestions since there does not exist a notion of type in trigger-action programming beyond simply triggers and actions.

4.3 Collaborative Filtering and Recipe Component Recommendation

As presented in Chapter 3, collaborative filtering methods provide a mechanism to recommend user–item ratings to users to explore new content. For the problem of predicting likely trigger-action programming components, we again turn to these methods to provide a solution.

In addition to simple user–item ratings, we are interested in making recommendations to the user from their previous recipes, the current recipe they are constructing and the similarity to other recipes already created on IFTTT. As one possible solution, we employ contextual recommender Factorization Machines for this problem.

Each row in the ratings matrix, contains a variable for the user, a variable for the component under recommendation and we include a subset of the four components to a recipe, the trigger channel, the trigger selected from the channel, the action channel and the action.

Similarly to the work presented in Chapter 3, we turn each example of a user's recipe into a set of rows iterating through the available items for recommendation and set a row's rating to 1 if the

R	User	Trigger Channel	Trigger
1	Stephen	GMail	Email in inbox
0	Stephen	GMail	Email with attachment
0	Stephen	GMail	Email from
1	Stephen	GMail	Email with label
...	Stephen	GMail	...
0	Michael	SMS	Send IFTTT SMS
1	Michael	SMS	Send IFTTT SMS with Hashtag
...	...	...	...

(a) Indicator Function

R	User	Trigger Channel	Trigger
4	Stephen	GMail	Email in inbox
0	Stephen	GMail	Email with attachment
0	Stephen	GMail	Email from
2	Stephen	GMail	Email with label
...	Stephen	GMail	...
0	Michael	SMS	Send IFTTT SMS
3	Michael	SMS	Send IFTTT SMS with Hashtag
...	...	...	...

(b) Counts

Table 4.1: Examples of the two types of rating matrix data: indicator function of whether user and variables were observed in training and testing datasets, and counts of the times they were observed in the datasets.

user has utilized that component before or 0 otherwise. Alternatively, we can set the row's rating to the number of times that user has created a recipe with that component. The second approach can increase the weight of the most commonly used recipe components for that user. Examples of these types of data are shown in figure 4.1.

4.4 Clustering Model for Component Recommendation

We developed a clustering model for contextual variables to predict the component under recommendation. We use an Expectation Maximization (EM) algorithm to cluster over independent variables to predict the dependent variable. In many cases, the independent variable would be the user, and we would want to predict the most likely recipe components. However, we can also cluster over trigger channels to predict the likely triggers. This sort of component model allows for a probabilistic interpretation of strict channel and function categories.

From the individual models, we can then combine the distributions over the independent models to produce a better informed distribution of the components we are predicting.

4.4.1 Single Component Clustering Model

The Expectation Maximation algorithm is used in this section to derive a clustering model to predict a component recommendation. Our goal is to compute the expected complete data log likelihood. Doing so through an iterative E-step, M-step process, we can find the parameters that maximize the likelihood of our data. This expected complete data log likelihood is given below, where x_i the observed data, which in this case is the item being predicted and the independent variable, for example users. z are the latent clusters we are trying to group the independent variables. θ are the parameters of the model to be computed at the current time-step,

$$Q\left(\theta, \theta^{t-1}\right) = \mathbb{E}\left[\sum_{i}^{N} \log \Pr(x_i, z_i \mid \theta)\right] \tag{4.1}$$

$$= \sum_{i}^{N} \mathbb{E}\left[\log \Pr(x_i, z_i \mid \theta)\right] \tag{4.2}$$

To derive the parameters we need to compute, we can substitute the following definition into the above equation. Here $\pi_k = \Pr(z_k)$.

$$\Pr(x_i, z_i \mid \theta) = \prod_{k}^{K} (\pi_k * \Pr(x_i \mid \theta_k)) * \mathbb{I}(z_i = k) \tag{4.3}$$

What follows are a series of algebraic steps to arrive at a form which helps understand the parameters necessary.

$$Q\left(\theta, \theta^{t-1}\right) = \sum_{i}^{N} \mathbb{E}\left[\log\left(\prod_{k}^{K} (\pi_k * \Pr(x_i \mid \theta_k))^{\mathbb{I}(z_i=k)}\right)\right] \tag{4.4}$$

$$= \sum_{i}^{N} \mathbb{E}\left[\sum_{k}^{K} \mathbb{I}(z_i = k) \log\left(\pi_k * \Pr(x_i \mid \theta_k)\right)\right] \tag{4.5}$$

$$= \sum_{i}^{N} \sum_{k}^{K} \mathbb{E}\left[\mathbb{I}(z_i = k) \log\left(\pi_k * \Pr(x_i \mid \theta_k)\right)\right] \tag{4.6}$$

$$= \sum_{i}^{N} \sum_{k}^{K} \mathbb{E}\left(\mathbb{I}(z_i = k)\right) \log\left(\pi_k * \Pr(x_i \mid \theta_k)\right) \tag{4.7}$$

$$= \sum_{i}^{N} \sum_{k}^{K} \Pr\left(z_i = k \mid x_i, \theta^{t-1}\right) \log\left(\pi_k * \Pr(x_i \mid \theta_k)\right) \tag{4.8}$$

$$= \sum_{i}^{N} \sum_{k}^{K} \Pr\left(z_i = k \mid x_i, \theta^{t-1}\right) \log(\pi_k) + \sum_{i}^{N} \sum_{k}^{K} \Pr\left(z_i = k \mid x_i, \theta^{t-1}\right) \log\left(\Pr(x_i \mid \theta_k)\right) \tag{4.9}$$

We now need to estimate these individual terms of this equation. Each x_i is a tuple of an independent variable v_{ind} and the dependent variable we are looking to predict v_{dep}. We need to break up the pieces above into parts to reflect the parts of x_i.

$$\Pr(z_k \mid x_i) = \Pr(z_k \mid v_{ind}, v_{dep}) \tag{4.10}$$

$$= \frac{\Pr(v_{ind}, v_{dep} \mid z_k) * \Pr(z_k)}{\Pr(v_{ind}, v_{dep})} \tag{4.11}$$

$$= \frac{\Pr(z_k \mid v_{ind}) * \Pr(v_{ind}) * \Pr(v_{dep} \mid z_k)}{\Pr(v_{ind}, v_{dep})} \tag{4.12}$$

$$\Pr(x_i \mid z_k) = \Pr(v_{ind}, v_{dep} \mid z_k) = \frac{\Pr(z_k \mid v_{ind}) * \Pr(v_{ind}) * \Pr(v_{dep} \mid z_k)}{\Pr(z_k)} \tag{4.13}$$

Because, every datapoint x_i contains two discrete variables, we need to keep track of the distributions over clusters given the independent variable $\Pr(z_i \mid v_{ind})$ and the distribution over dependent

variables given the clusters $\Pr(v_{dep} \mid z_i)$.

$$\Pr(z_k \mid v_{ind}) = \frac{\sum_{x_i=(v_{ind},*)}^{X} \Pr(z_k \mid x_i)}{\sum_{z'} \sum_{x_j=(v_{ind},*)}^{X} \Pr(z' \mid x_j)} \tag{4.14}$$

$$\Pr(v_{dep} \mid z_k) = \frac{\sum_{x_i=(*,v_{dep})}^{X} \Pr(z_k \mid x_i)}{\sum_{z'} \sum_{x_j=(*,v_{dep})}^{X} \Pr(z' \mid x_j)} \tag{4.15}$$

To predict the dependent variable given a single independent variable with our clustering model is then

$$\Pr(v_{dep} \mid v_{ind}) = \sum_{z} \Pr(z \mid v_{ind}) * \Pr(v_{dep} \mid z) \tag{4.16}$$

4.4.2 Combining Models

Given multiple models which predict a dependent variable from different independent variables, how do we combine their predictions to create a more informed model? This section focuses on combining the predictions through multiplication.

Our goal is to derive a combination model from a Naive Bayes assumption. Given a set of independent variables $v_1, \cdots, v_n$ we want to find the probability of dependent variable v_{dep}. The Naive Bayes model assumes that each independent variable is also independent of the other variables.

$$\Pr(v_{dep} \mid v_1, \cdots, v_n) \propto \Pr(v_{dep}) * \prod_{i}^{n} \Pr(v_i \mid v_{dep}) \tag{4.17}$$

Each model computes $\Pr(v_{dep} \mid v_i)$, so we would like an equivalent expression but with these terms incorporated. From the chain rule, we know:

$$\Pr(v_{dep}, v_1, \cdots, v_n) = \Pr(v_1, \cdots, v_n, v_{dep}) \tag{4.18}$$

$$= \Pr(v_1 \mid v_2, \cdots, v_n, v_{dep}) * \Pr(v_2, \cdots, v_n, v_{dep}) \tag{4.19}$$

$$= \Pr(v_{dep}) * \prod_{i}^{n} \Pr(v_i \mid v_{i+1}, \cdots, v_n, v_{dep}) \tag{4.20}$$

By assuming conditional independence following the Naive Bayes model:

$$\Pr(v_i \mid v_{i+1}, \cdots, v_n, v_{dep}) = \Pr(v_i \mid v_{dep}) \tag{4.21}$$

Then we are left with chain rule simplification

$$\Pr(v_{dep}, v_1, \cdots, v_n) = \Pr(v_{dep}) * \prod_{i}^{n} \Pr(v_i \mid v_{dep}) \tag{4.22}$$

$$= \Pr(v_{dep} \mid v_1, \cdots, v_n) * \Pr(v_1, \cdots, v_n) \tag{4.23}$$

Then we make this a proportionality by dropping the joint probability of $\Pr(v_1, \cdots, v_n)$

$$\Pr(v_{dep} \mid v_1, \cdots, v_n) \propto \Pr(v_{dep}) * \prod_{i}^{n} \Pr(v_i \mid v_{dep}) \tag{4.24}$$

Now if we look at each individual component $\Pr(v_i \mid v_{dep})$ we can find its relation to our individual model $\Pr(v_{dep} \mid v_i)$ again by Bayes Rule

$$\Pr(v_i \mid v_{dep}) = \frac{\Pr(v_{dep} \mid v_i) * \Pr(v_i)}{\Pr(v_{dep})} \tag{4.25}$$

Substituting this in to equation 4.24 we have:

$$\Pr(v_{dep} \mid v_1, \cdots, v_n) \propto \Pr(v_{dep}) * \prod_i^n \frac{\Pr(v_{dep} \mid v_i) * \Pr(v_i)}{\Pr(v_{dep})} \tag{4.26}$$

$$\propto \Pr(v_{dep}) * \left[\prod_i^n \frac{\Pr(v_i)}{\Pr(v_{dep})}\right] * \left[\prod_i^n \Pr(v_{dep} \mid v_i)\right] \tag{4.27}$$

$$\tag{4.28}$$

Since we're just interested in a ranking of probabilities for the dependent variable v_{dep} we can drop the $\prod_i^n \Pr(v_i)$ from the equation because it will be equivalent over all v_{dep}.

$$\Pr(v_{dep} \mid v_1, \cdots, v_n) \propto \Pr(v_{dep})^{-(n-1)} * \prod_i^n \Pr(v_{dep} \mid v_i) \tag{4.29}$$

4.5 Initial Dataset

We were provided with the collection of over $200,000$ IFTTT trigger-action programming recipes from Ur et al. [65]. Each recipe included the user who shared the recipe, a description, title, number of recipe adoptions by other users on the site and the components of the recipe: trigger channel, trigger, action channel, and action.

As we were only interested in the recipe components and user, we stripped out all other information and created unique variable identification numbers for each possible variable. This data was split into a training set (70%), development set (10%) and a testing set (20%).

4.6 Results

We ran four different evaluations to predict each component of a recipe. For each evaluation, we only used a subset of components. For the different channels, we predicted the best channel given the other required channel. That is, given a trigger channel, we predicted the action channel and

vice versa. For the channel functions, we predicted them from their associated channel. We broke each evaluation into two categories, predicting with the user information and without.

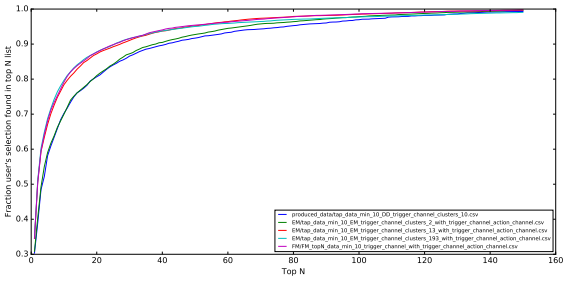
For each evaluation we compared five different models. A collaborative filtering model based solely on the users, the EM model with 2, 13, and 193 clusters used for each model, and also a factorization machines model.

In each evaluation the FM model performs as well as or better than the other models. For some evaluations the EM model with 193 clusters does just as well as the FM model, so it's quite likely that EM models with more clusters might perform similarly as the FM model. The straight collaborative model does poorly regardless, since it does not make use of any contextual information.

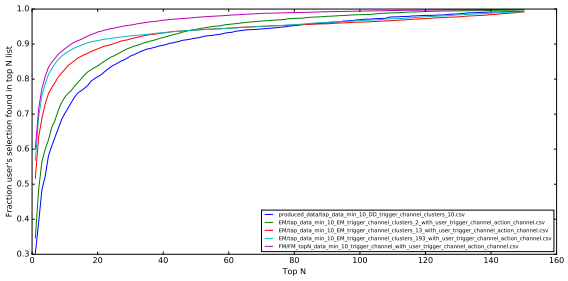
Across each evaluation, adding users to the models significantly helped the predictive performance. There was more noticeable improvement for predicting the channels with user information than just the functions. However, they also had significantly more room for improvement.

Figure 4.1 plots the result of these different evaluations.

Without Users

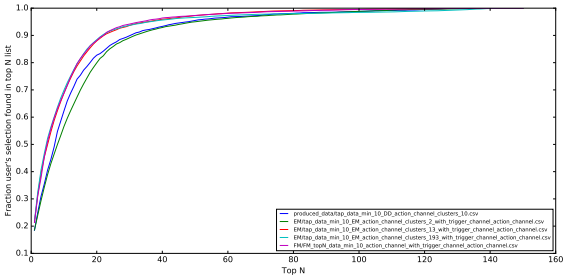


With Users

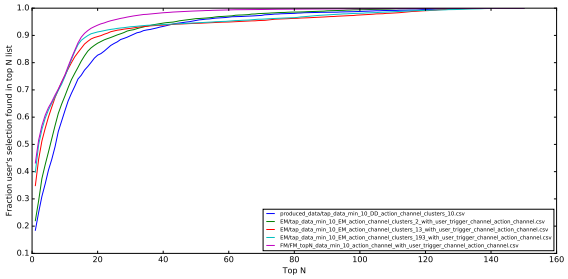


(a) Predicting Trigger Channel from Action Channel

Without Users

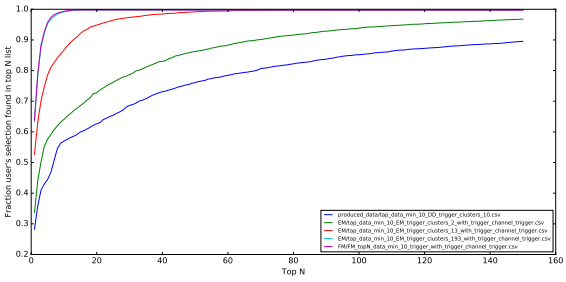


With Users

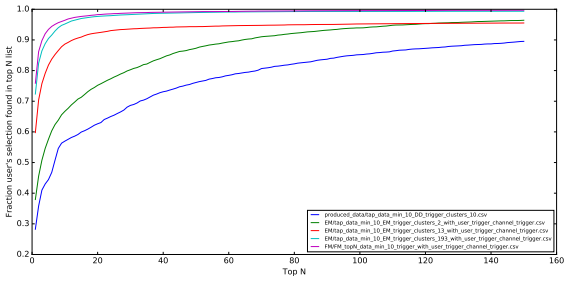


(b) Predicting Action Channel from Trigger Channel

Without Users

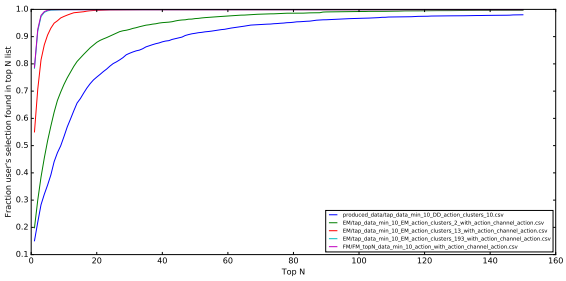


With Users

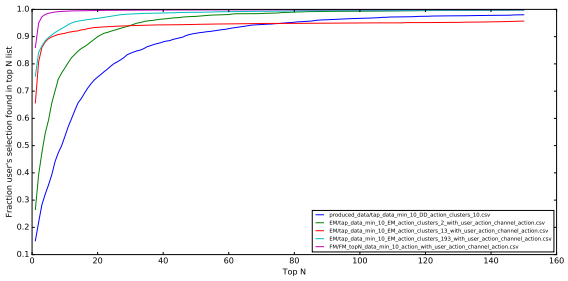


(c) Predicting Trigger from Trigger Channel

Without Users



With Users



(d) Predicting Action from Action Channel

Figure 4.1: Plots showing success of predicting the user's chosen recipe component from different prior given independent variables. Models incorporating users as an independent variable are shown side-by-side with an equivalent model that does not.

Chapter 5

Trigger-Action Program Synthesis from Natural Language Commands

5.1 Introduction

The problem of translating natural language to computer programming remains a largely unsolved problem. Researchers have targeted specific domains with either limited grammars or domains with strict formulaic syntax.

There have been several researchers that have specifically looked into the problem of translating natural language text into If-This-Then-That recipes. This previous work has modeled the problem as one of machine translation.

These solutions have not, however, taken into account user preferences. We are aware from results presented in Chapter 4 that users limit themselves to smaller combinations of items, and we can predict these patterns through contextual recommendation methods.

This work investigated how we can best combine the recommendations from the recommender system with the predictions from a machine-translation system. For example, a translation system

that is unaware of a user's preferences might struggle to identify the specific email channel a user is referring to when they say, "If it is raining tomorrow, send me an email". However, if we have previous examples that show they typically prefer a single email channel, like 'Gmail', then we can better predict their translation.

5.2 Related Work

There has been significant interest in translating natural language to a logical form. Branavan et al. [19] learn natural language groundings from Windows troubleshooting guides. There also exists significant work in understanding database queries from natural text as first proposed by Harris [30].

In the realm of embodied smart devices, there exists much more research in the robotics domain, ranging from coaching robots in RoboCup [36] to providing commands for navigation and mobile manipulation [63]. Recently, there has been interest in composing trigger-action programs from natural language descriptions, which we describe next.

Quirk et al. [54] first introduced the problem of translating natural language descriptions to If-this-then-that (IFTTT) recipes. They built the original dataset composed of nearly 115,000 recipes and introduced several baseline methods for translating the recipe titles into parse trees of the IFTTT recipes, including the trigger/action channels, functions and recipe parameters. They leveraged a method similar to that introduced by Kate and Mooney [35].

Dong and Lapata [25] followed up on that work by introducing a recurrent neural network model with neural attention. Besides improving on the success on the previously introduced IFTTT dataset, their method successfully generalized to a number of tasks besides IFTTT recipe translation.

Beltagy and Quirk [16] followed up on the work introduced in [54], and bumped the state of the art a few percentage points.

Liu et al. [40] has presented the most recent state of the art for this dataset by introducing

latent attention, which computes an individual weight for each input token, which determines its importance in predicting the output trigger or action. They also introduce a one-shot version of the problem.

Though these researchers have made significant progress, they have not addressed the problem of personalizing translations to each individual user. We have shown in Chapter 4, that there is significant prediction potential by utilizing user's preferences when predicting IFTTT recipe components, and we hypothesize this information can be used to improve the translation of recipes.

5.3 Trigger-Action Programs

A trigger-action program is composed of a triggering event and an action that is signaled to occur on the triggering event. For IFTTT, triggers and actions belong to channels, which helpfully group the triggers and actions according to their specific categories. Many triggers and actions in addition have specific parameters. For example, the Weather Channel's 'Tomorrow's forecast calls for' trigger, requires an additional parameter to specify 'rain', 'cloudy', 'clear', or 'snow'. Though an IFTTT recipe is not complete without these parameters, the dataset provided by Quirk et al. [54] only includes links to the recipes and IFTTT does not currently show the original parameters chosen by the recipe's author. Therefore, current work focuses solely on predicting the trigger channel, trigger, action channel and action.

5.4 Grounding TAP As Machine Translation

Grounding trigger-action programs from natural language descriptions is naturally formatted as a machine translation problem. In machine translation, tokens or words in the first language are translated to tokens or program components in the second language. Though the trigger-action programs have a strict ordering and limited vocabulary, the order of words and vocabulary size is somewhat unbounded for the input language. Algorithms in machine translation have to solve both

problems of word alignment and translation.

In machine translation, the IBM models first introduced by Brown et al. [20] laid a strong foundation in statistical methods that were developed upon for roughly 20 years until the resurgence of deep learning. These methods draw from a Bayesian understanding to find a translation that maximizes the probability of the translation and the probability of the target language phrase. For example, though the French phrase 'Coûter les yeux de la tête' literally translates 'to cost the eyes from the head' English speakers might understand the translation as 'to cost an arm and a leg'. These methods find a translation that balances the translation of the phrase with the likelihood of that phrase in the target language.

As neural networks became popular in computer vision, NLP researchers began to study their applicability to language translation, a novel idea given the structural differences of the two fields. Kalchbrenner and Blunsom [34] introduced a recurrent neural network approach that allowed for end-to-end learning. They broke the popular Markovian n-gram assumption about word dependencies in the target language. Sutskever et al. [62] introduced the Sequence to Sequence translation model, which incorporated the use of Long short-term memory units in neural networks. Though it did not at the time match the performance of the state of art of Durrani et al. [26], a phrasal based statistical method, it performed significantly well with a relatively simple, unoptimized model. Because of the model simplicity and generalizability, their techniques have enjoyed wide-spread applicability in NLP.

Regarding this specific problem of translating natural language to trigger-action programs, Liu et al. [40] built on the Sequence to Sequence model by incorporating a Latent-Attention module that emphasizes the terms and phrases that have higher importance in the translation.

5.5 Incorporating A Learned Model to Improve Translation

We have devised three separate methods to improve the grounding of natural language utterances to TAP programs. The first method combines the predicted likelihoods from a collaborative filtering recommender and the pre-existing translation models from Liu et al. [40]. The second model consists of several different translation models that are incrementally improved over clusters of users through an Expectation-Maximization method. The third model also incorporates several separate translation models, but fully learns user assignments and cluster weights in a single large deep learning model.

5.5.1 Hybrid Recommender System

The collaborative filtering recommender, Factorization Machines, is a model with an input matrix where each row represents a single TAP program, and each column represents the context variables, which includes the program author and all the possible components. By training over this matrix, it finds a factorized modeling of these context variables that reduces the recommendation error. On prediction, we find the combinations of context variables that are most recommended for a given author.

We can build these programs by finding the highest rated trigger given the author, then the highest rated action given the author and preceding trigger. Each prediction is provided with a rating value, generally in the range of [0,1], though not strictly.

As most Sequence to Sequence based models, our chosen translation model presented in [40] predicts a distribution over choices for each component of the program. The most likely components are selected as the components of the program.

To combine the predictions from both these models, the FM recommender (R_{FM}) and the Sequence to Sequence neural network translation model ($R_{seq2seq}$), we learn a linear function to combine them:

$$R = (A_{seq2seq} + B_{seq2seq} * R_{seq2seq}(x_i)) * (A_{FM} + B_{FM} * R_{FM}(x_i)) \,. \tag{5.1}$$

The loss function we minimize is then l^2-norm of the recommendation error between the recommended value and actual recommender indicator value for the true program component $\hat{x}$

$$e = \|R_i - \mathbb{1}(x_i = \hat{x})\|^2 \,. \tag{5.2}$$

5.5.2 Expectation Maximization Model

A recommendation system provides a recommended selection for a given user and user-item ratings. Another form of providing recommendations for our translation model is to recommend a translator for a provided author. Instead of learning a program component recommender and a translation model separately, we can use an expectation-maximization model that learns translation models for different populations of users.

For this model, we assume that an author's true intent can be modeled by a weighted combination of different translation models. For example, the translation of the phrase, 'Tell me if the weather is bad tomorrow' into the program {Trigger: *If tomorrow's weather predicts rain*, Action: *Send me an SMS* } might combine a translator that puts more weight on translating 'bad weather' onto a trigger specific to rain and a translator that emphasizes translations for 'Tell me' to an SMS action, as opposed to an email or app notification. In this manner, this method recommends a learned translator for each component rather than separate recommendations for the program and language translation.

The Expectation-Maximization algorithm is a reliable workhorse for finding reasonable solutions to latent-clustering models like that proposed above. It is an iterative algorithm that seeks to maximize the likelihood of latent parameter estimates in a model. In our model, the latent parameters represent the weights assigned to each translation model for a given user. The likelihood is then the

probability of an author's actual translation as estimated by the combined translation model.

This algorithm divides each iteration into two steps: calculating the expected log-likelihood of the data given the current model parameters (the expectation or E step), and calculating parameter values that maximize the likelihood found in the previous step (the maximization or M step).

As with most statistical methods, calculations of the likelihood are performed in log-space so as to eliminate computational issues due to arithmetic underflow. Arithmetic underflow is the issue whereby products of many small numbers lose precision and become 0 due to the limitations imposed by the bounds on exponents in floating point numbers.

At the commencement of our method, we choose a number of translation models, and for all our users randomly assigned latent weights of each model to all users. Unfortunately, to randomly sample weights, a uniform distribution over $[0, 1]$ does not suffice because the weights do not remain uniformly distributed after normalization. To solve this issue, we take the negative natural log of a normal distribution and normalize those values to compute normalized weight values evenly distributed in $[0, 1]$. Here, $\Pr(A_i \mid \Theta, z_k)$ represents the probability of translation model k for author i:

$$w_{i,k} = -\log \mathbf{U}(0, 1). \tag{5.3}$$

$$\Pr(A_i \mid \Theta_k, z_k) = \frac{w_{i,k}}{\sum_j w_{j,k}}. \tag{5.4}$$

The log-likelihood is computed from a summation over the log-likelihoods of each author's natural language description and translation pair. For author A, model k and sentence s, the log-likelihood is:

$$ll_{A,k} = \sum_{s,(TC,T,AC,A)} [\log(\Pr(T \mid \Theta_k, s)) \tag{5.5}$$

$$+ \log(\Pr(TC \mid \Theta_k, s)) \tag{5.6}$$

$$+ \log(\Pr(A \mid \Theta_k, s)) \tag{5.7}$$

$$+ \log(\Pr(AC \mid \Theta_k, s))]. \tag{5.8}$$

The total log-likelihood is then:

$$ll = \sum_A \sum_k \Pr(A \mid \Theta_k) * ll_{A,k} \tag{5.9}$$

During the maximization step, the weights are calculated to maximize this log-likelihood. That is, it assign model weights such that the actual programs are the most likely predicted programs from the descriptions:

$$\Pr(A \mid \Theta_k) = \exp(ll_{A,k} + \log \Pr(\Theta_k)). \tag{5.10}$$

The prior weights of each model are then computed as:

$$\Pr(\Theta) = \frac{\sum_A \Pr(A \mid \Theta_k)}{\sum_A \sum_k \Pr(A \mid \Theta_k)} \tag{5.11}$$

This algorithm runs until convergence. Upon completion, we predict new translated programs by finding the most likely components from weighting distributions computed by each translation model.

For example, to find the trigger for author A given input sentence s, we find:

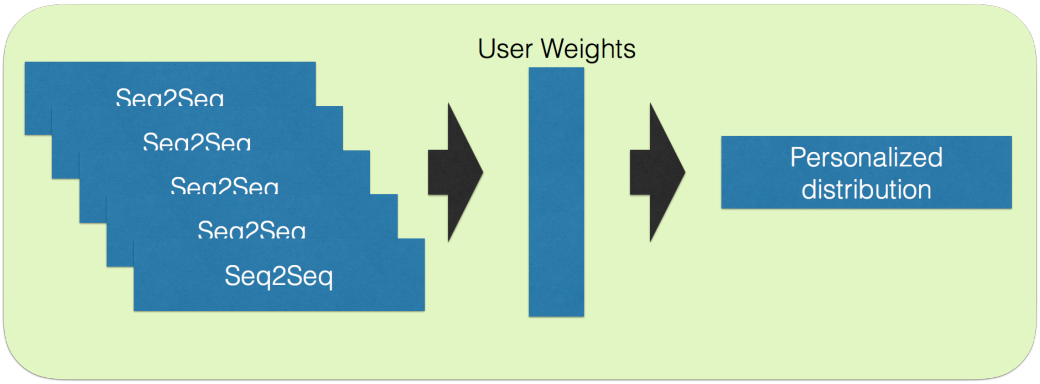


Figure 5.1: Diagram of personalized model trained through EM. Each author's translation prediction is a weighted combination of predictions from the personalized models.

$$\Pr(Trigger \mid s) = \sum_{k} \Pr(A \mid \Theta_k) * \Pr(Trigger \mid \Theta_k) \tag{5.12}$$

$$Trigger = \underset{Trigger}{\operatorname{argmax}} \Pr(Trigger \mid s). \tag{5.13}$$

5.5.3 Integrated Deep Learning Model

Deep learning strategies ofter are trained end-to-end, and this particular solution seeks to encapsulate the clustering method previously presented as an end-to-end network. Instead of learning the parameter weights of translation separately, we built a single large network composed of multiple translation networks with a set of output weights to produce a single output prediction.

N inner translation models are each fed the input sentence tokens. Though the exact translation model can be replaced with the machine translation model of one's choosing, we decided to incorporate again the model presented in Liu et al. [40]. The output distributions estimated by each model are then combined in a weighted-sum with weights unique to each author.

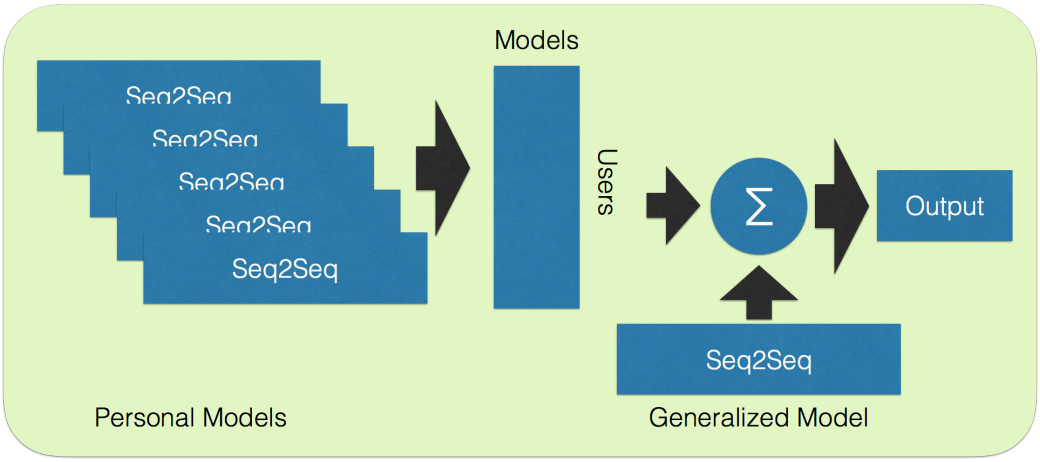


Figure 5.2: Diagram of the components of the deep personalized model. There are N personal models that combine with a matrix of weights to produce a summarized personal distribution of translations that is further combined with a general model.

5.6 Quirk et al. Dataset

The results presented in the works described in this chapter's related work all utilize the same dataset, first published in Quirk et al. [54]. It has been broken down into four categories of understandability: all programs, programs with only English words in the description, programs with intelligible English-only descriptions, and descriptions that were clear enough that a crowd-sourced workforce could recreate the recipe.

However, datasets drawn from IFTTT itself have an issue where most programs were written by authors with very minimal number of recipes. Ur et al. [65] had shown that, at least by 2016, 68% of all authors on the site had shared publicly only one program, representing nearly half of all programs on the website. Only about 7% of authors had shared at least 5 programs.

There remains another question regarding this dataset. Are authors easily summarized by a global statistical summary of all authors, or are the programs produced by each author perplexing in their own right? Is there even the possibility that a model could learn to personalize over the authors?

Figure 5.3 shows the distribution of trigger functions and action functions according to their

usage in the dataset provided by Quirk et al. [54]. The top three triggers include two specific to new RSS posts, and the third is a trigger that is activated at a specific time each day. The top three actions are posting a tweet, sending an email and sending a text SMS. Only 23 of the 494 triggers and 13 of the 200 actions are used in only a single program.

To measure how far each author's own program distribution differs from the global distribution, we used the Jensen–Shannon divergence metric (JS divergence). Like the Kullback–Leibler divergence (KL divergence), the JS divergence is a measurement of how two probability distributions differ. KL divergence has an inherent drawback with sample distributions that is problematic to our dataset. It expects each distribution to have all non-zero probabilities. Given the distributions Q and P, the KL divergence from Q to P is defined as:

$$D_{KL}(P,Q) = -\sum P(i) \log \frac{Q(i)}{P(i)}. \tag{5.14}$$

As one can see, if either $P(i)$ or $Q(i)$ is 0, the result is undefined. Because we desire to compare a global distribution of program components to a single author's, there will necessarily be zero-valued for the author's sample distribution. There are numerous tricks one could employ such that this value is defined over the entire distribution, like adding pseudocounts. However, JS divergence does not have this drawback, and does not lack symmetry like KL divergence ($D_{KL}(P,Q) \neq D_{KL}(Q,P)$).

The JS divergence is the mean of the KL divergence of the two distributions P and Q to the mean distribution $M = \frac{1}{2}(P+Q)$.

$$D_{JS} = \frac{1}{2} D_{KL}(P,M) + \frac{1}{2} D_{KL}(Q,M). \tag{5.15}$$

We calculated the JS divergence from the global population for each author. Also, we randomly generated 10,000 authors for each quantity of recipes represented by our population authors. These randomly generated authors' programs were drawn from the global distribution. We computed the minimum, maximum, mean JS divergence as well as the JS divergence values for which 2.5% and

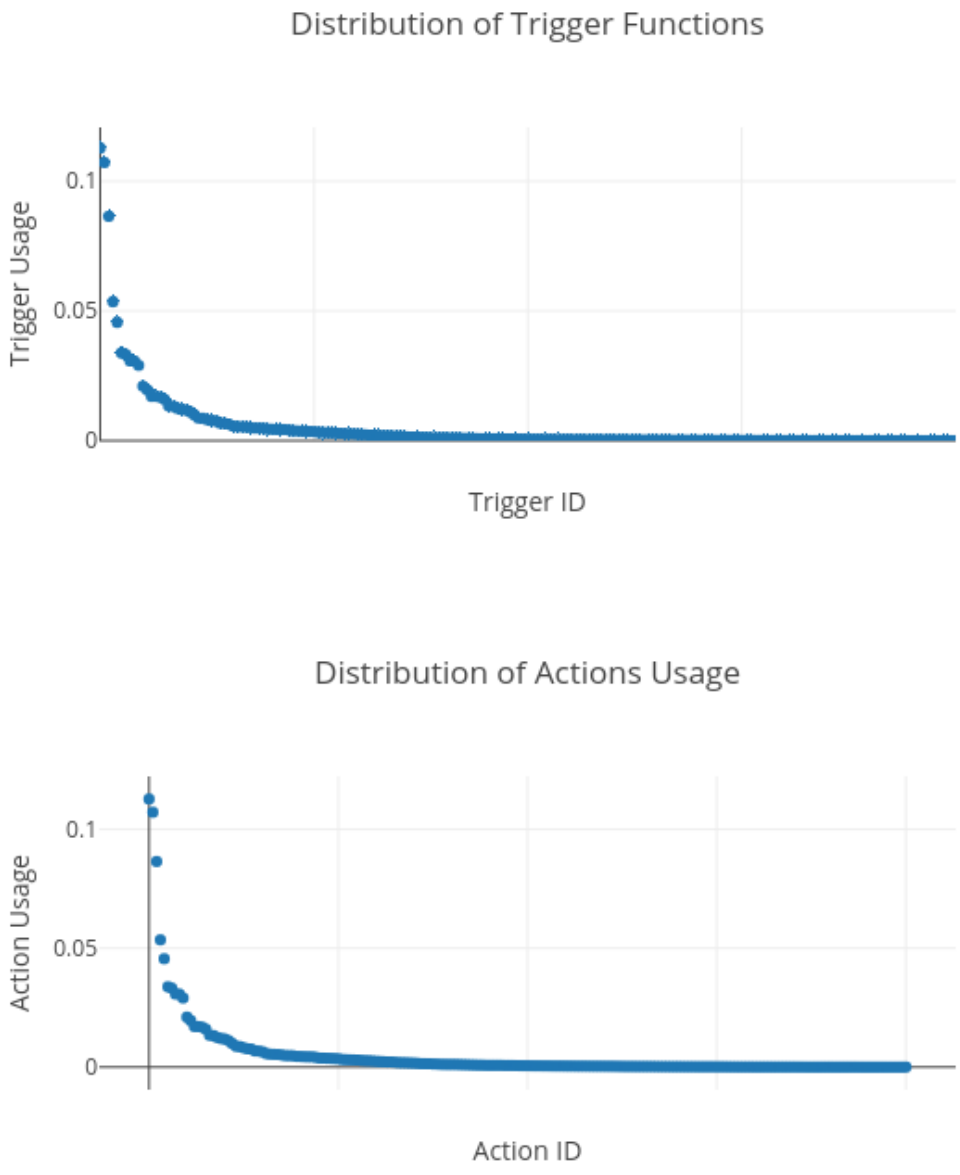


Figure 5.3: Distributions of usage of the Triggers and Action components in the Quick et al. dataset. A small fraction of triggers and actions are used quite abundantly.

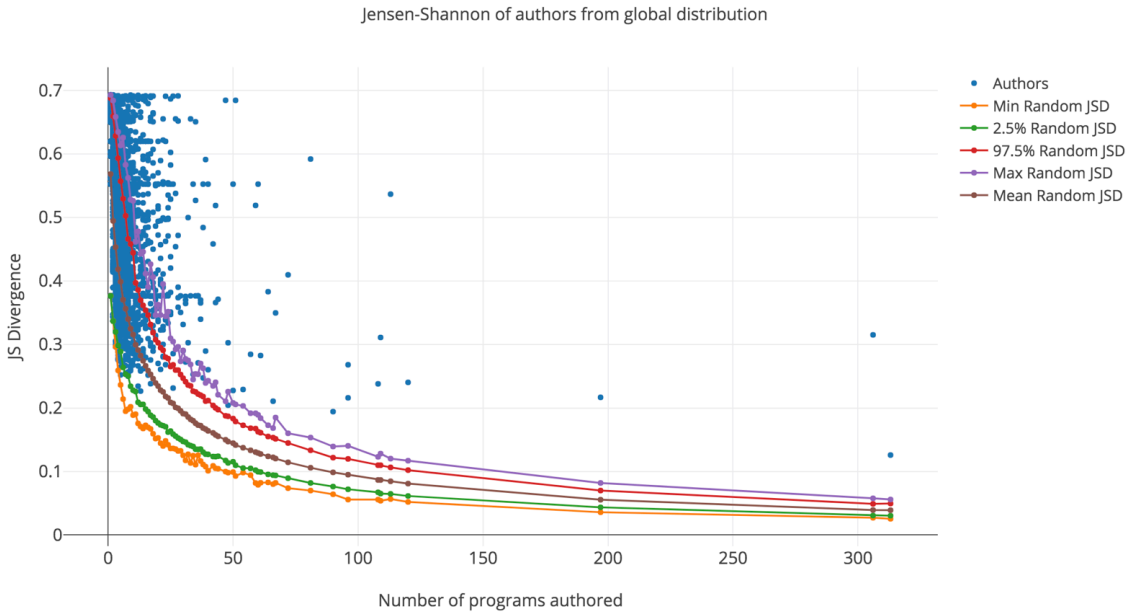


Figure 5.4: Plot of the Jensen–Shannon divergence of each author's trigger component distribution to the global trigger component distribution. Statistical summary lines of the randomly generated authors are also plotted.

97.5% of random authors were below these values.

Figure 5.4 shows a plot with the JS divergence of each author against the number of programs they shared on IFTTT. Also plotted are the lines summarizing the values of the randomly generated authors. This graph shows that as the number of recipes increases for a given author, their pattern of program-creation gets further away from the JS divergence of random authors generated by the global distribution. In fact, for authors with more than 50 programs, they exhibited a JS divergence to the global distribution greater than all randomly generated authors.

In Figure 5.5, the number of authors with a JS divergence greater than 97.5% of the generated authors is plotted in blue. As can be seen for authors with more than 20 programs, pretty much all patterns of behavior show a significant divergence from the global distribution. 46% of authors with 10 programs showed a JS divergence greater than 97.5% of the generated authors.

As can be seen with these metrics of programming behavior by the site's authors, authors with a

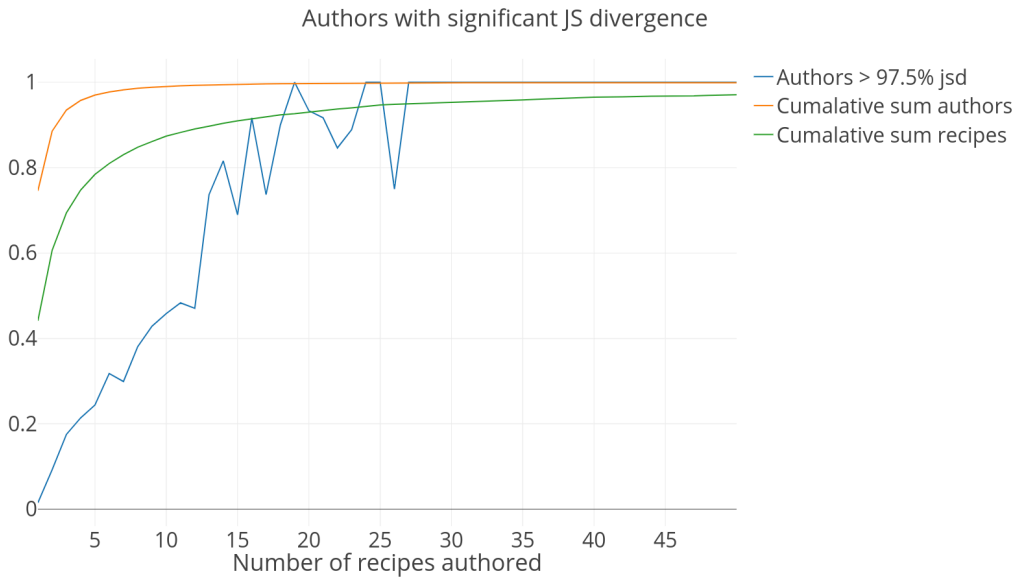


Figure 5.5: Plot of the fraction of authors with JS divergence greater than 97.5% of random authors. The cumulative fraction of recipes and fraction of authors versus the number of recipes authored is also plotted. Though prolific authors show program creation behavior different from the global distribution, they represent only a small subset of this dataset.

reasonable number of shared programs show patterns of behavior strikingly distinct from the norm. A model that learns from a global trend cannot possibly service all authors equally well.

5.7 New AMT Dataset

Due to the issues presented previously about the standard dataset used for this problem, we sought to build a new dataset with much more potential for personalization. With the issue that too few authors in the previous dataset shared more than a few programs, we sough to create a dataset that is perhaps more representative of typical users.

Creating an artificial dataset for this problem is no easy task. Though getting natural language descriptions from AMT is simple enough, these descriptions needed to be good English phrases that conveyed the intent of the trigger-action program, but were also sufficiently vague to require contextual information about the artificial user.

By analyzing the most common triggers and actions, we found several categories of trigger and action capabilities that were replicated among many different services. For triggers we selected the five most popular that were associated with the words 'email', 'post', 'temperature', 'location', 'time'. Similarly for actions, we selected those associated with the words 'email', 'sms', 'light', 'post' and 'backup'.

From these most popular items, we created six profiles of users. Each 'artificial' user used a different trigger and action for their desired program. The intent is that each program may be described similarly, but use different program components.

We then built 18 different programs for each artificial user and separated them into two different surveys. We asked participants to write an English description that would be adequate for a human programmer to understand their intent.

In total we collected 660 descriptions for 108 different programs. We did not remove any descriptions for being unclear or unintelligible. The descriptions provided by the participants for the most part were sufficiently clear for a reasonable person to create the appropriate program.

The benefit of this dataset is that many of the descriptions use general words and phrases like 'text me' or 'send an email' instead of something too precise like 'use the SMS service to send me an SMS' or 'use gmail to send me an email'. This imprecision provided an opportunity to evaluate the ability of a system to personalize translations to the context of the user under evaluation.

5.8 Evaluation

We evaluated the different translation models on the full IFTTT dataset from Quirk et al. [54]. Liu et al. [40] presented strong results for their model on the 'gold' data subset—programs that were replicated by at least three Amazon Mechanical Turk participants from the accompanying description – and our evaluation of their method shows similarly strong performance. They achieve 85.1% accuracy composing the entire program from a single description. Our solution actually

	Trigger Channel	Trigger	Action Channel	Action	TC+AC	TC+T+AC+A
English						
Seq2Seq LA	69.6	67.6	58.7	56.8	51.4	39.6
EM Clusters	71.7	59.7	**67.0**	57.6	51.8	31.4
Gen Personal * 3	**72.2**	**71.4**	61.7	**59.6**	**57.2**	**43.8**
Intelligible						
Seq2Seq LA	77.6	74.2	66.5	65.1	60.1	47.0
EM Clusters	**80.7**	68.0	**78.0**	**66.7**	62.7	37.3
Gen Personal * 3	78.8	**78.3**	67.8	**66.6**	**65.6**	**49.5**
Gold						
Seq2Seq LA	98.3	**98.3**	93.6	93.2	85.5	**85.1**
EM Clusters	**100.0**	86.2	**100.0**	**96.6**	**100.0**	82.7
Gen Personal * 3	98.3	95.3	92.3	91.1	81.7	79.1

Table 5.1: Results evaluated on dataset provided from Quirk et al. [54]. Numbers are the percent correct of selecting the appropriate program component or their combined accuracy.

performs worse on this gold standard, but outperforms their model on every other data subset.

On the 'intelligible' dataset—descriptions that were meaningful English sentences but not precise enough for humans to replicate the associated program—the multiple combined model increased each metric significantly. We improved the trigger and action functions selections from 71.4% to 78.3% and 59.6% to 66.6%. The full program recreation was improved from 47% to 49.5%.

We also evaluated the Latent-Attention model and our personalized model on the new dataset obtained from AMT. This dataset does not have the subdivisions of the Quirk et al. dataset, but these descriptions were all generated by people with the intent that other people would be able to reproduce the described programs.

Over all metrics, the General Personal model outperformed the Latent-Attention model. The personal model increased the combined channel accuracy from 78.4% to 83.8% and the full program accuracy from 70.3% to 81.1%.

5.9 Discussion

Natural language understanding is of growing importance in end-user applications. Many devices with embodied language understanding are being developed—Amazon's Echo, Apple's Siri, Microsoft's Cortana and Google's Google Home are embodied with a AI presence that understands a

	TC	T	AC	A	TC+AC	TC+T+AC+A
Single Model						
Seq2Seq LA	89.2	86.5	86.5	78.4	78.4	70.3
3 Models						
Seq2Seq LA Ensemble	93.9	84.8	**93.9**	87.9	**87.8**	78.8
EM Clusters	84.8	87.8	93.9	81.8	78.8	66.6
Gen Personal Model	**94.6**	**94.6**	89.1	**89.1**	83.8	**83.8**
6 Models						
Seq2Seq LA Ensemble	**93.9**	90.9	90.9	90.9	84.8	81.8
Gen Personal Model	91.9	**94.6**	86.5	89.2	**86.5**	**86.5**

Table 5.2: Results evaluated on the AMT dataset. Numbers are the percent correct of selecting the appropriate program component or their combined accuracy.

limited amount of natural language commands.

For users that desire to automate their home, using these interfaces may provide the most convenient solution to setting up their own automation. However, descriptions provided by one user may carry different meaning even if they are identical word for word to another user's description. 'Text me if the weather is bad tomorrow' has different groundings depending on the user's notion of bad weather and their desired texting service.

In this chapter, we have presented a solution to personalizing grounding natural language utterances to trigger-action programs. We improved the state-of-the-art results on the canonical dataset for this program for subsets of the data that are inherently vague and require understanding user context. We also showed significant improvement on a custom built dataset that required significant personal attention.

For smart devices that understand human language and can build smart programs, they will need to understand the context of their user. However, training machine translators from the ground up for each new user is impractical. Instead, our solution balances the benefits of learning from a general population and the target users themselves.

Chapter 6

Conclusions

The problem of improving how AI specializes to household applications is unfortunately difficult to solve in only a single dissertation. As this work has found, there is yet to be a single overarching solution. Instead, for each problem a unique solution was identified.

In the case of improving household organization, Factorization Machines, a context-aware recommending system, helped generalize the problem across multiple households. Utilizing context variables that a robot could easily perceive in a household, it was able to build generalizations about different objects and the types of locations in which they may be placed. A fork was unlikely to be placed in a refrigerator or freezer, so even if few objects had been placed in a kitchen the robot at least had something to work with. Making common sense judgements built from a large collection of households will reduce the stress and frustration for the robot user.

For programming a smart home, we found that users' patterns of behavior are highly unique. Additionally, their programs become more distinguishable as they create more programs. For a user who desires to write new programs in the home, they can especially utilize the benefits of recommender systems. Due to the modular nature of trigger-action programs, users have difficulty discovering all the possible combinations and are driven to recreate many of the same programs. Based on their history of usage, our application of Factorization Machines in this context enables

such a programming paradigm to aid users.

Collaborative filtering has been shown in this work to solve a large number of suitable problems requiring the incorporation of generalization and personalization. However, for more complex problems in AI that a robot is likely to face, we had to turn to deep neural networks.

We lastly investigated the problem of personalizing grounding natural language to trigger-action programming. Proper groundings required an understanding of the components installed in a user's home as well as identifier labels the user might give to different components in the home. For example, 'Turn on the living room lights when I arrive home', is an intelligible program that a user might desire to create. However, in the realm of trigger-action programming, the number of functions that provide the ability to turn on a light or denote when a user arrives home means that this description is actually quite vague. Beyond deciding if the user meant Philip Hue lights, or WeMo lights, or something else, the translator needs to understand what specific light bulbs 'living room' translates to.

Such a problem requires translations personalized for each user in a household. Live with someone long enough and the most vague instructions become coherent. However, current solutions with deep learning require significant quantity and diversity of data to adequately generalize to the problem. Our solution investigated how we might cluster household users to different model translations. We found for the original dataset that this solution improved significantly the system's ability to translate vague descriptions. The task descriptions that were intelligible English but not sufficiently precise for others to recreate their associated programs improved.

Furthermore, when investigating a new dataset with natural language descriptions provided by Amazon Mechanical Turkers, we found that a large deep neural network comprised of many single translation models improved translations quite a bit even compared to an ensemble method of the same number of translators.

The results shown in this work illustrate the promise of how AI solutions commonly used for robotics can be made to be personalized to the robot's household user.

6.1 Implications

Going back to the examples provided in Chapters 1 and 2, we found many uses where a technology designed to help people automate their home often backfired due to its difficulty to use. Even just examining the role of computers in our lives, a significant subpopulation find them difficult to use. Often this population is older and less inclined to learn new ways of doing things.

Introducing robots to these households, especially for the elderly or infirm, requires extremely high standards of usability and reliability. One might be frustrated when Microsoft Word reacts unexpectedly to a keystroke or mouse action, and that program just moves around words and images. After introducing a robot into the household, which can move around physical objects, these users will not tolerate robotic systems or automated households that manipulate their physical world in unexpected and undesirable ways.

Solutions to this problem will require a combination of strong software engineering, good user-interface design, and advanced AI solutions. Software engineering can identify reliability issues when encountering expected situations, and ensure the programmed behaviors react as expected to the unexpected. UI design will help the roboticists build interactive models that ensure the user's mental model matches how the robot actually interacts. For a robot with AI sufficiently advanced to learn from the household users, there exist a large number of possibilities that the designers cannot possibly test for.

If a smarthome or robot in a household is given the ability to learn from household users, there exists the possibility of unexpected behaviors learned from this interaction. Even assuming the household user is a benevolent trainer to the robot, machine learning is prone to overfitting. This is a common issue in machine learning where a model with too many parameters is fitted to insufficient data. The trained model, though showing outstanding success with the training data, performs especially poorly on even slightly different evaluation data.

In the household, if a robot overfits a task solution to the user's instructions, it could lead to

unforeseen results. For example, suppose a user, while attempting to show the robot how to clean up the kitchen, shows it that a glass belongs in the trashcan because it is chipped. The robot—not perceiving that this chipped glass is any different from the glasses on the shelf—may learn that all glasses of this type belong in the trashcan. Or, if the robot is insufficiently unable to distinguish this type of glass from any of the other glasses on the shelf, may learn that all glasses belong in the trash. Though single scenarios like this example may be imagined and tested by a robot designer, the fundamental issue remains that human users may ascribe too many perceptive abilities to the robot leading the robot to misunderstand a trained task.

The solutions presented in this work proceed from a point of view that personalization in the household needs to incorporate common sense. It not only helps the user teach the robot fewer examples, it may also lead the robot to make fewer mistakes. Utilizing a library of common sense built from robots or smarthomes installed across millions of homes is one goal of the generalized learning problem we try to incorporate here.

As argued in this thesis, the fundamental nature of training robots in the home will require a balance of generalization and specialization.

6.2 Future Work

This work has looked at combining generalized learning and personalized learning for household robotics. There are numerous problems for which this melding is crucial.

As alluded to before, the benefit of general learning is imparting a sense of common knowledge, or even common sense, on to the robot and smart home. They can learn to specialize with a single person, but training examples that do not fit within its common sense framework will require more examples to overcome.

The quantity of problems that fall within this framework is quite numerous. In the vein of language translation, robots that understand commands or task descriptions will always need to

have both common sense and personal context. The number of training examples required for grounding natural language utterances to tangible concepts will always be greater than a human user is willing to tolerate unless a larger, crowd-sourced dataset is available to serve as the training source for the common knowledge.

Though this thesis touched on the problem of describing to a robot how to organize a home, it assumed there existed an easy interface for connecting objects and their desired locations. Expanding this problem to allow for natural language descriptions requires understanding when terms like cup and glass may refer to the same object or when they may refer to different ones. It also requires understanding when silverware does not actually go in the silverware drawer – that fancy cutlery that is only used for fancy occasions and has its own drawer elsewhere, for example.

Though cleaning, unlike organization, can be better generalized across homes – households are much more likely to hire someone to clean their house than re-organize it – there is still a large number of instructions that people may give to a cleaning robot to accomplish the task to the person's preference. One will have to draw the robot's attention to parts of the house that require deeper cleaning, or special care to not damage anything. Every household is different, and a robot's perceived usefulness will always depend on how well it can match the expectations of its user.

Even non-manipulative tasks will require a degree of personalization for applications in the home.

At the time of this publication, Brown University's Human Centered Robotics Initiative is engaged in a partnership with the toy manufacturer Hasbro, Inc. to study the possibilities for embuing robotic care animals with artificial intelligence.

Their current care animal comes in the form of a robotic cat and comes with a limited, but physically realistic, cat-like actions. Adding a simple camera to this cat could enable a large host of computer vision approaches that may help understand how the robo-cat caretaker lives.

One such problem that is currently under investigation is to analyze a patient's speed and dexterity of movement around the household. By carrying this feline automaton around the house, it can image keypoints around the home and attach meaningful metrics to the person's quality of

movement. However, the challenge is learning these keypoints and their locations around the home without significant amounts of training by the robo-cat's user or their own caretaker. A generalized learning model could understand to a fairly good degree the type of room that the cat is currently within, but in many homes there may be rooms that defy strict definition. Can one train a computer vision model from a large dataset of rooms and households such that learning a user's home is significantly easier?

This dissertation has described the problem of combining generalized learning and personalized learning, but the larger framework pits generalized learning versus specialized learning without a user context. Though the argument about focusing on the needs of the user becomes moot without a user, there are many applications that still require specialized learning and can benefit from common knowledge developed from generalized learning.

Following this work, we are investigating the potential to improve renewable energy power plants with deep reinforcement learning. Though reinforcement learning has the fundamental problem of requiring long learning episodes, we hope that transferring knowledge learned in one location can be transferred to a target power plant.

6.3 Final Word

The main argument of this thesis was:

> *Learning algorithms for robots and smart homes perform better when they trade-off learning from both generalization of a large population and individualization through repeated user interactions than either strategy alone.*

For household robots and learning smart-home devices, we are currently at an interesting nexus in development. Will these manifest AI systems learn in the home and through user interaction, or will their intelligence be carefully constructed under their company's cultivation? By adopting and making use of solutions like those presented in this work, we may one day be able to own a curious, interactive learner rather than factory stock devices that are only periodically remotely updated for better general performance.

Bibliography

[1] Sony sl-7200 brochure. URL http://www.mrbetamax.com/BroSL7200Legend.htm.

[2] The blinking twelve problem: Closed loop marketing. URL http://mann.ly/the-blinking-1200-problem/.

[3] K-tronics, l.c. - cl9 - core remote control unit. URL http://www.ktronicslc.com/core.html.

[4] Netflix, deep sea horror movies. URL http://www.netflix.com/browse/genre/45028.

[5] Honeywell round thermostat. URL http://www.mnopedia.org/thing/honeywell-round-thermostat.

[6] URL http://www.ktronicslc.com/core.html.

[7] Netflix, movies for ages 0 to 2. URL http://www.netflix.com/browse/genre/6796.

[8] The minneapolis heat regulator, 1918. URL https://www.si.edu/object/nmah_1392735.

[9] URL https://tedium.co/2017/05/25/universal-remote-control-history/.

[10] URL http://www.touchsquid.com/main/products/touchsquid-gr-universal-remote/.

[11] Netflix, understated independent coming-of-age movies. URL https://www.netflix.com/browse/genre/562.

[12] Nichola Abdo, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard. Collaborative filtering for predicting user preferences for organizing objects. *arXiv preprint arXiv:1512.06362*, 2015.

[13] Nichola Abdo, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1557–1564. IEEE, 2015.

[14] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[15] Kim Baraka and Manuela Veloso. Adaptive interaction of persistent robots to user temporal preferences. In *International Conference on Social Robotics*, pages 61–71. Springer, 2015.

[16] I. Beltagy and Chris Quirk. Improved semantic parsers for if-then statements. ACL, 2016.

[17] Daniel Billsus and Michael J Pazzani. Learning collaborative information filters. In *ICML*, volume 98, pages 46–54, 1998.

[18] Mark Bittman. *How to Cook Everything: 2,000 Simple Recipes for Great Food*. John Wiley & Sons, 2011.

[19] Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.

[20] Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[21] Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 213–222. ACM, 2009.

[22] Maya Cakmak and Leila Takayama. Towards a comprehensive chore list for domestic robots. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 93–94. IEEE Press, 2013.

[23] Elizabeth Cha, Jodi Forlizzi, and Siddhartha S Srinivasa. Robots in the home: Qualitative and quantitative insights into kitchen organization. In *HRI*, pages 319–326, 2015.

[24] Kerstin Dautenhahn. Robots we like to live with?!-a developmental perspective on a personalized, life-long robot companion. In *Robot and human interactive communication, 2004. ROMAN 2004. 13th IEEE International Workshop on*, pages 17–22. IEEE, 2004.

[25] Li Dong and Mirella Lapata. Language to logical form with neural attention. *Association for Computational Linguistics (ACL)*, 2016.

[26] Nadir Durrani, Barry Haddow, Philipp Koehn, and Kenneth Heafield. Edinburgh's phrase-based machine translation systems for wmt-14. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 97–104, 2014.

[27] Damien Ernst, Guy-Bart Stan, Jorge Goncalves, and Louis Wehenkel. Clinical data based optimal sti strategies for hiv: a reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, pages 667–672. IEEE, 2006.

[28] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG)*, 31(6):135, 2012.

[29] Goren Gordon, Samuel Spaulding, Jacqueline Kory Westlund, Jin Joo Lee, Luke Plummer, Marayna Martinez, Madhurima Das, and Cynthia Breazeal. Affective personalization of a social robot tutor for children's second language skills. In *AAAI*, pages 3951–3957, 2016.

[30] Larry R Harris. User oriented data base query with the robot natural language query system. *International Journal of Man-Machine Studies*, 9(6):697–713, 1977.

[31] Irit Hochberg, Guy Feraru, Mark Kozdoba, Shie Mannor, Moshe Tennenholtz, and Elad Yom-Tov. A reinforcement learning system to encourage physical activity in diabetes patients. *arXiv preprint arXiv:1605.04070*, 2016.

[32] Yun Jiang, Changxi Zheng, Marcus Lim, and Ashutosh Saxena. Learning to place new objects. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3088–3095. IEEE, 2012.

[33] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed May 19, 2016].

[34] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[35] Rohit J Kate and Raymond J Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 913–920. Association for Computational Linguistics, 2006.

[36] Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1062. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[37] Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, pages 319–326. IEEE, 2012.

[38] Iolanda Leite, Carlos Martinho, and Ana Paiva. Social robots for long-term interaction: a survey. *International Journal of Social Robotics*, 5(2):291–308, 2013.

[39] Daniel Leyzberg, Samuel Spaulding, and Brian Scassellati. Personalizing robot tutors to individuals' learning differences. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 423–430. ACM, 2014.

[40] Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. Latent attention for if-then program synthesis. In *Advances In Neural Information Processing Systems*, pages 4574–4582, 2016.

[41] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[42] Pattie Maes et al. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[43] Vukosi Ntsakisi Marivate, Jessica Chemali, Emma Brunskill, and Michael L Littman. Quantifying uncertainty in batch personalized sequential decision making. In *AAAI Workshop: Modern Artificial Intelligence for Health Analytics*, 2014.

[44] Martin Mason and Manuel C Lopes. Robot self-initiative and personalization by learning through repeated interactions. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 433–440. ACM, 2011.

[45] Gail C Murphy, Mik Kersten, and Leah Findlater. How are java software developers using the elipse ide? *IEEE software*, 23(4):76–83, 2006.

[46] Nest. Home | nest, 2016. URL `https://nest.com/`. Accessed: 2016-4-1.

[47] Anh Tuan Nguyen and Tien N Nguyen. Graph-based statistical language model for code. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 858–868. IEEE Press, 2015.

[48] Cyrus Omar, YoungSeok Yoon, Thomas D LaToza, and Brad A Myers. Active code completion. In *Proceedings of the 34th International Conference on Software Engineering*, pages 859–869. IEEE Press, 2012.

[49] Umberto Panniello, Alexander Tuzhilin, Michele Gorgoglione, Cosimo Palmisano, and Anto Pedone. Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 265–268. ACM, 2009.

[50] Caroline Pantofaru, Leila Takayama, Tully Foote, and Bianca Soto. Exploring the role of robots in home organization. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 327–334. ACM, 2012.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[52] Scott Pigg and Monica Nevius. Energy and housing in wisconsin: A study of single-family owner-occupied homes. November 2000. URL `http://infohouse.p2ric.org/ref/40/39353.pdf`.

[53] Joelle Pineau, Marc G Bellemare, A John Rush, Adrian Ghizaru, and Susan A Murphy. Constructing evidence-based treatment strategies using methods from computer science. *Drug and alcohol dependence*, 88:S52–S60, 2007.

[54] Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, 2015.

[55] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[56] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3): 57:1–57:22, May 2012. ISSN 2157-6904.

[57] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

[58] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[59] Romain Robbes and Michele Lanza. Improving code completion with program history. *Automated Software Engineering*, 17(2):181–212, 2010.

[60] Martin J Schuster, Dominik Jain, Moritz Tenorth, and Michael Beetz. Learning organizational principles in human environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3867–3874. IEEE, 2012.

[61] Cory-Ann Smarr, Tracy L Mitzner, Jenay M Beer, Akanksha Prakash, Tiffany L Chen, Charles C Kemp, and Wendy A Rogers. Domestic robots for older adults: attitudes, preferences, and potential. *International journal of social robotics*, 6(2):229–247, 2014.

[62] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[63] Stefanie A Tellex, Thomas Fleming Kollar, Steven R Dickerson, Matthew R Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. 2011.

[64] Linden Tibbets, Jesse Tane, Scott Tong, and Alexander Tibbets. Discover ifttt and applets - ifttt, 2010–. URL `http://ifttt.com/`. [Online; accessed Jan 13, 2017].

[65] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. 2016.

[66] Manuela M Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, page 4423, 2015.

[67] Rayoung Yang and Mark W Newman. Learning from a learning thermostat: lessons for intelligent systems for the home. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 93–102. ACM, 2013.

[68] Yufan Zhao, Michael R Kosorok, and Donglin Zeng. Reinforcement learning design for cancer clinical trials. *Statistics in medicine*, 28(26):3294–3315, 2009.