The Intelligent Management of Crowd-Powered Machine Learning

Christopher H. Lin

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

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Daniel S. Weld, Chair

Mausam, Chair

Eric Horvitz

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

The Intelligent Management of Crowd-Powered Machine Learning

Christopher H. Lin

Co-Chairs of the Supervisory Committee:
Professor Daniel S. Weld
Computer Science & Engineering

Associate Professor Mausam
Computer Science & Engineering

Artificial intelligence and machine learning power many technologies today, from spam filters to self-driving cars to medical decision assistants. While this revolution has hugely benefited from algorithmic developments, it also could not have occurred without data, which nowadays is frequently procured at massive scale from crowds. Because data is so crucial, a key next step towards truly autonomous agents is the design of better methods for intelligently managing now-ubiquitous crowd-powered data-gathering processes. This dissertation takes this key next step by developing algorithms for the online and dynamic control of these processes. We consider how to gather data for its two primary purposes: training and evaluation.

In the first part of the dissertation, we develop algorithms for obtaining data for testing. The most important requirement of testing data is that it must be extremely clean. Thus to deal with noisy human annotations, machine learning practitioners typically rely on careful workflow design and advanced statistical techniques for label aggregation. A common process involves designing and testing multiple crowdsourcing workflows for their tasks, identifying the single best-performing workflow, and then aggregating worker responses from redundant runs of that single workflow. We improve upon this process by building two control models:

one that allows for switching between many workflows depending on how well a particular workflow is performing for a given example and worker; and one that can aggregate labels from tasks that do not have a finite predefined set of multiple choice answers (*e.g.* counting tasks). We then implement agents that use our new models to dynamically choose whether to acquire more labels from the crowd or stop, and show that they can produce higher quality labels at a cheaper cost than state-of-the-art baselines.

In the second part of the dissertation, we shift to tackle the second purpose of data: training. Because learning algorithms are often robust to noise, training sets do not necessarily have to be clean and have more complex requirements. We first investigate a tradeoff between size and noise. We survey how inductive bias, worker accuracy, and budget affect whether a larger and noisier training set or a smaller and cleaner one will train better classifiers. We then set up a formal framework for dynamically choosing the next example to label or relabel by generalizing active learning to allow for relabeling, which we call re-active learning, and we design new algorithms for re-active learning that outperform active learning baselines. Finally, we leave the noisy setting and investigate how to collect balanced training sets in domains of varying skew, by considering a setting in which workers can not only label examples, but also generate examples with various distributions. We design algorithms that can intelligently switch between deploying these various worker tasks depending on the skew in the dataset, and show that our algorithms can result in significantly better performance than state-of-the-art baselines.

TABLE OF CONTENTS

Page

List of Figures iv

List of Tables viii

Chapter 1: Introduction 1
1.1 Intelligently Gathering Data for Evaluation 4
1.2 Intelligently Gathering Data for Training 6
1.3 Dissertation Outline 9

Part I: Intelligently Gathering Data from the Crowd for Testing 10

Chapter 2: Dynamically Switching between Synergistic Workflows 11
2.1 Introduction 11
2.2 Background 13
2.3 Probabilistic Model for Multiple Workflows 17
2.4 A Decision-Theoretic Agent 19
2.5 Learning the Model 21
2.6 Experiments 24
2.7 Related Work 32
2.8 Conclusion 33

Chapter 3: Moving Beyond Multiple Choice 35
3.1 Introduction 35
3.2 Background 36
3.3 Probabilistic Model 37
3.4 A Decision-Theoretic Agent 43
3.5 Experiments 52

3.6 Related Work . . . 57
3.7 Conclusion . . . 58

Part II: Intelligently Gathering Data from the Crowd for Training . . . 60

Chapter 4: To Re(label), or Not To Re(label) . . . 61
4.1 Introduction . . . 61
4.2 Problem Setting . . . 63
4.3 The Effect of Inductive Bias . . . 64
4.4 The Effect of Worker Accuracy . . . 71
4.5 The Effect of Budget . . . 74
4.6 Real Dataset Experiments . . . 76
4.7 Related Work . . . 77
4.8 Conclusion . . . 78

Chapter 5: Re-Active Learning: Active Learning with Relabeling . . . 80
5.1 Introduction . . . 80
5.2 Preliminaries . . . 81
5.3 Algorithms For Re-Active Learning . . . 82
5.4 Behavior of Impact Sampling . . . 92
5.5 Experiments . . . 100
5.6 Discussion . . . 106
5.7 Related Work . . . 107
5.8 Conclusion . . . 108

Chapter 6: Skew-Robust Learning Powered by the Crowd . . . 109
6.1 Introduction . . . 109
6.2 Example-Acquisition Primitives . . . 111
6.3 Algorithms . . . 112
6.4 Experiments . . . 114
6.5 Related Work . . . 131
6.6 Conclusion . . . 133

Chapter 7: Conclusions and Future Work 135
7.1 Conclusions 135
7.2 Future Work 137

Bibliography 141

LIST OF FIGURES

Figure Number — Page

1.1 In the first part of the dissertation, we develop methods for collecting labels for test sets as accurately and cheaply as possible. In the second part of the dissertation, we analyze the tradeoffs that exist when collecting training data and build agents that can intelligently make these tradeoffs in their effort to maximize machine learning performance. 3

2.1 a) A Bayesian network with 2 random variables. b) A Bayesian network with $2g$ random variables. c) A Bayesian network in which Z_2 and Z_3 are conditionally independent given Z_1. 15

2.2 A worker's answer b depends on the difficulty of the question d, the worker's error parameter γ and the question's true answer v. There are W workers, who complete Ψ tasks. b is the only observed variable. 17

2.3 A worker's answer b depends on the difficulty of the question d (generated by δ), the worker's error parameter γ and the question's true answer v. There are W workers, who complete Ψ tasks, all of which can be solved using a set of K different workflows. b is the only observed variable. 18

2.4 AgentHunt's decisions when executing a task. 20

2.5 In this NER task, the TagFlow is considerably harder than the WikiFlow since the tags are very similar. The correct tag set for this task is {location} since Washington State is neither a county nor a citytown. 25

2.6 In simulation, as the importance of answer correctness increases, AgentHunt outperforms TurKontrol by an ever-increasing margin. 28

2.7 In simulation, as the importance of answer correctness increases, both agents converge to 100 percent accuracy, but AgentHunt does so more quickly. . . 29

3.1 Whether or not worker i gets it right, x_i, depends on the worker's error parameter γ_i and the difficulty of the task d. Worker i's answer b_i depends on x_i, the question's true answer v, and all the previous workers' answers $b_1, \ldots, b_{i-1}$. R_i is a Chinese Restaurant Process defined by $\mathbf{b_i}$. This figure shows 4 workers. The b_i are the only observed variables. 39

3.2 In simulation, our lookahead search does well at all depths. 53

3.3 An example of an SAT Math Question task posed to workers for live experiments on Mechanical Turk. 56

3.4 An example of a "triangle" task posed to workers for live experiments on Mechanical Turk. The area of this triangle, rounded down, is 38. 56

4.1 Relabeling training data improved learned-classifier accuracy on 5 out of 12 real-world domains (see Table 4.1) when workers were highly noisy (55% accurate). But why these domains and not the others? 62

4.2 An "O" represents an example labeled as a "senior citizen" and an "X" designates an example labeled as "not a senior citizen." The target concept classifies all people older than 65 as senior citizens. 65

4.3 Given a fixed budget of 1000, we see in various settings of worker accuracy that as the VC dimension increases, more relabeling of a fewer number of examples achieves lower upper bounds on classification error. 66

4.4 As the number of features (VC dimension) increases, relabeling becomes more and more effective at training an accurate logistic regression classifier. 69

4.5 Classifiers with weaker inductive bias tend to benefit more from relabeling. . 69

4.6 Decision trees with lower regularization tend to benefit more from relabeling. 70

4.7 The increase in the aggregate accuracy of the training set data when using relabeling instead of unilabeling for various worker accuracies. 71

4.8 When the workers are moderately accurate ($p = 0.75$), the decrease in the upper bound on error from additional relabeling is greatest. 72

4.9 For simulated Gaussian datasets, relabeling strategies based on majority-vote are most powerful at moderate values of worker accuracy. 73

4.10 When $p = 0.75$, relabeling strategies initially achieve higher accuracies than unilabeling, but are eventually defeated. 75

4.11 When $p = 0.55$, relabeling strategies initially achieve higher accuracies than unilabeling, but are defeated earlier than when $p = 0.75$. 75

4.12 Unilabeling obtains better classifiers in some datasets, even when the workers are moderately accurate (p=0.75). 77

5.1 An example domain in which a naïve extension of uncertainty sampling to re-active learning, $\texttt{US}_{\mathcal{X}}$, can fall into a trap. Given that x_1 and x_2 are the only labeled points in this distribution of diamonds and circles and h is the current hypothesis, uncertainty sampling is likely to converge to behavior in which it will always pick these same examples to relabel and cause an infinite loop in which no learning takes place. The true hypothesis, h^*, will never be learned. 83

5.2 Suppose x_1, x_2, x_3, and x_4 have been labeled multiple times and have converged to the correct label. Suppose x_5 is the only unlabeled example. If we are learning a max-margin classifier, then h would be the current hypothesis. If x_1, x_2, x_3, and x_4 are far enough away from h so that $P_{\mathcal{A}}(h^*(x_1) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_2) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_3) = \text{circle})$, and $P_{\mathcal{A}}(h^*(x_4) = \text{circle})$ are equal to 1, and x_5 is close enough to h so that $P_{\mathcal{A}}(h^*(x_5) = \text{diamond}) = P_{\mathcal{A}}(h^*(x_5) = \text{circle}) = 0.5$, then the error of h is 0.5. Labeling one of x_1, x_2, x_3, or x_4 will not change the hypothesis, and so the expected error reduction is 0. However, labeling x_5 will result in a hypothesis that *increases* the expected error from 0.5 to 0.75! As a result, EER is likely to converge to behavior in which it will always relabel x_1, x_2, x_3, and x_4 forever. 88

5.3 Average generalization accuracy of logistic regression over randomly generated Gaussian datasets with 90 features and label accuracy 0.75, when trained using various strategies. a) compares various settings of $\texttt{US}^{\alpha}_{\mathcal{X}}$ and shows $\alpha = 0.1$ is best. b) compares various settings of $\texttt{US}^{j/k}_{\mathcal{X}_U}$ and shows that $j/k = 3/5$ is best. c) compares various impact sampling strategies and shows `impactPLOPT(7)` is best. d) compares impact sampling, uncertainty sampling, and EER, and shows that impact sampling produces the best results. 101

5.4 Comparison of impact sampling versus uncertainty sampling on real-world datasets, Internet Ads and Arrhythmia, using simulated labels modeling annotators whose accuracy is 0.75. Impact sampling shows significant gains. . . 103

5.5 Impact sampling performs well against uncertainty sampling on the real-world datasets with real human-generated labels. 104

6.1 As skew decreases, `RandomNegs` become less effective than `ModifyNegs`. However, in such cases, `Label-Only(Random)` is the most effective strategy, so Crowd-Modify+To- is likely not a good EAP to use in any skew setting. . . . 120

6.2 `MB-CB(Active)` outperforms `MB-CB(Pos)` in all skew settings when training classifiers to identify "Health" headlines, and shows similar performance for the other domains. 124

6.3 When the skew is 999, `MB-CB(Active)` executes Labeling Primitives more often and Generation Primitives less often than `MB-CB(Pos)`. 125

6.4 When the skew is 999, `MB-CB(Active)` surprisingly obtains more positive examples from Labeling Primitives than does `MB-CB(Pos)`. 125

6.5 Comparison of `MB-T(Active)`, `MB-CB(Active)`, `Round-Robin`, `Label-Only(Active)`, `GL`, and `GL-Hybrid` on 4 domains with synthetic Generation Sets. `MB-T(Active)` and `MB-CB(Active)` both train better classifiers than the state-of-the-art baselines, `GL` and `GL-Hybrid`, across many domains and skew settings. 128

6.6 Comparison of `MB-T(Active)`, `MB-CB(Active)`, `Round-Robin`, `Label-Only(Active)`, `GL`, and `GL-Hybrid` on 4 domains using Generation Sets with real data. `MB-CB(Active)` and `MB-T(Active)` both train better classifiers than the state-of-the-art baselines, `GL` and `GL-Hybrid`, across many domains and skew settings. 129

LIST OF TABLES

Table Number Page

1.1 We strengthen assumptions as we move from chapter to chapter in this dissertation in order to tackle more complex problems. While we begin with models that consider workers of varying skill and tasks of varying difficulty, we move to settings in which we assume crowd responses are independent and identically distributed regardless of the worker and task, and then finally to a setting in which we assume workers are perfect. We also make a number of assumptions common to all chapters. In particular, we assume that tasks have a single objective true answer; workers are not adversarial and do not collaborate; and no mislabeled example, whether for training or testing, is more important than any other mislabeled example. 3

2.1 Comparisons of accuracies, costs, and net utilities of various agents when run on Mechanical Turk. 31

3.1 Summary of notation used in this chapter 44

3.2 Comparison of average accuracies, costs, and net utilities of LazySusan and MV when workers either have $\gamma \in (0, 1)$ or $\gamma \in (0, 2)$. 55

3.3 Comparisons of average accuracies, costs, and net utilities of LazySusan and MV when run on Mechanical Turk. 57

4.1 The 12 datasets we use, with the total number of examples and number of features in each. 76

ACKNOWLEDGMENTS

My advisors, Dan Weld and Mausam, have selflessly dedicated years of their lives to making me a better scientist. They have tried to bestow upon me all of their vast wisdom, and provided me with all the opportunities a PhD student could ever wish for, and they have done all this with infinite patience and understanding. I could not have asked for better advisors. Thank you Dan, and thank you Mausam.

Of course my graduate school experience was also shaped immensely by my many labmates, hallwaymates, and buildingmates, including Jonathan Bragg, Xiao Ling, Gagan Bansal, Shih-Wen Huang, Angli Liu, Eunsol Choi, Danielle Bragg, Jim Chen, Congle Zhang, Ryan Drapeau, Daniel Gorrie, Eunice Jun, Aileen Granstrom, Colin Lockard, Lydia Chilton, James Ferguson, Peng Dai, Sherry Wu, Mandar Joshi, Raphael Hoffmann, and Mitchell Koch. Many of these past and current members of the extended CrowdLab have not only been wonderful colleagues, but also amazing friends. I'm thankful for their abundant assistance with my research, and I've thrived on our many trips to the fitness torture chamber and our fun discussions about life, liberty, and the pursuit of happiness.

While most of my time was spent in Seattle, I also had the opportunity to go on several side adventures on the other side of Lake Washington through fruitful internships at Microsoft Research. While there, my mentors, Ece Kamar, Andrey Kolobov, and Eric Horvitz, were perfect supplements to my advisors at UW, graciously dedicating time and energy towards my growth and providing me with more diverse perspectives on science in general. Dan Bohus, Zhou Yu, and Rich Caruana also contributed to my very enjoyable experience, from providing entertaining lunch conversations to taking me on freezing cold sailboat rides.

I have also benefited from numerous comments, discussions, and technical help from

members of the academic community outside my immediate circle of colleagues and collaborators, including Adish Singla, Walter Lasecki, Andrew Mao, Chien-Ju Ho, Siddharth Suri, Stephen Soderland, Rob Miller, Michael Bernstein, Eytan Adar, Haoqi Zhang, Carlos Guestrin, Thomas Richardson, David Alan Grier, Matt Lease, Yiling Chen, Pedro Domingos, Luke Zettlemoyer, and Josh Attenberg.

Finally, I would like to thank those people outside of my academic world who have made my 6 years in grad school more survivable, more colorful, and more fulfilling. Many wonderful new friends from the Seattle Philharmonic Orchestra and the Serendipity Quartet have not only kept me sane and helped me grow as a musician, but also taught me a surprisingly huge amount about life in general, including Michael Shen, Jonathan Icasas, Janice Lee, Sooyoung Hong, Jae-In Shin, Allion Salvador, Amy Werner-Allen, Jon Epstein, Sheila Oh, Matt Sayre, Adam Stern, Shion Yamakawa, and Nick Pozoulakis. In addition, I am lucky to have equally-wonderful old friends Yiding Jia, Richard Shin, Darren Kuo, Charles Chen, Alvin Chen, Long Wei, and Calvin Wang, who have kept in touch despite long distances and generously given me much computer science and personal support. I treasure our late-night gaming sessions and our many amusing chats.

And without a doubt, none of my accomplishments could exist without my parents, Tsung-Hua Lin and Nai-Huei Chen, who tirelessly built the foundation of who I am today. They and my brother, Alex Lin, have always supported me through good and bad. I am very grateful for their unconditional love.

DEDICATION

To my parents

Chapter 1

INTRODUCTION

Research in artificial intelligence and machine learning has exploded in the last decade, bringing humanity to the cusp of self-driving cars, digital personal assistants, and unbeatable game-playing robots. While the exponential improvement in ability can be attributed to many factors, including the advancements of core machine learning algorithms, perhaps the most important driver is the unprecedented availability of massive amounts of data. More data is always better, and the rise of crowdsourcing has contributed to making data very cheap and very abundant, with large technology companies like Google and Microsoft heavily relying on internal crowdsourcing platforms to power their machine learning systems.

Crowdsourcing is powering artificial intelligence, and artificial intelligence is revolutionizing the world. This dissertation envisions the natural next step: artificial intelligence that powers crowdsourcing, completing the loop that will bring us closer to truly autonomous thinking machines. Imagine the following agent: while reading a newspaper to keep up with current events, it sees a verb it doesn't understand. Instead of doing nothing, it launches into learning mode. It asks the crowd to provide sentences that use this verb and similar verbs to help it train its natural language understanding algorithm. It might also ask the crowd to label examples from the web that it finds confusing or ambiguous. Later on, it tests itself, by attempting to form sentences with that verb and asking the crowd whether its usage is correct, perhaps dynamically increasing the number of people it asks if the initial ones disagree. Then, it fine-tunes its hyper-parameters, rinses, and repeats.

Instead of passively waiting to be given a manually curated dataset by its machine learning practitioner overlord, this agent is able to actively seek out data from the crowd, intelligently gathering the examples and labels that will help it best improve. Unfortunately,

such an agent is still far-off today. Despite progressing to super-human intelligence in so many facets of life, machines still need to be told by humans how to learn what they wish to learn. At a higher level, while machines have become very good at the execution of machine learning, they are not very good at the management of machine learning. A key next step towards creating agents like the one we envision above and thereby significantly transform artificial intelligence is to develop algorithms for the intelligent management of crowd-powered machine learning. In this dissertation, we take that key next step through a wide-ranging examination of the various ways in which an agent can intelligently solicit help from humans during machine learning.

Our examination is built upon and organized around two perhaps obvious but important observations. First, machine learning uses data for primarily two purposes: training and testing, and second, the requirements for data for these two purposes are very different. On the one hand, data collected for testing must be as close to noise-free as possible, because measuring performance accurately with a noisy gold standard is a lost cause. On the other hand, data collected for training does not necessarily need to be extremely clean, because many machine learning algorithms are robust to noise. Instead, in order to maximize performance of the classifier, training sets must 1) find a happy medium between noise and size, and 2) be balanced in class.

Because cost-effective data is so extraordinarily critical to machine learning and the requirements for differently-purposed data are so different, we develop models and algorithms for the intelligent control of crowd-powered data-gathering processes specifically with these differences in mind. In the first part of the dissertation, we develop methods for collecting labels for test sets as accurately and cheaply as possible. In the second part of the dissertation, we analyze the many requirements that exist when collecting training data and build agents that can intelligently manage these requirements in their effort to maximize machine learning performance.

As we tackle more complex problems from chapter to chapter, we note that we rely on increasingly stronger assumptions in order to make progress, and readers of this dissertation

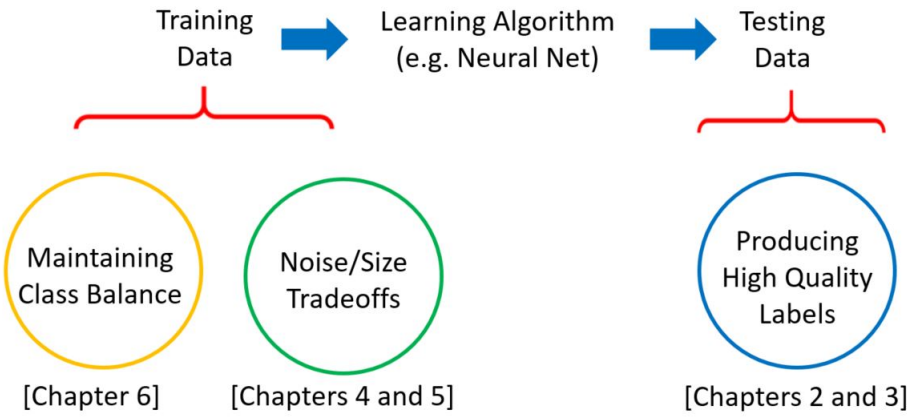

Figure 1.1: In the first part of the dissertation, we develop methods for collecting labels for test sets as accurately and cheaply as possible. In the second part of the dissertation, we analyze the tradeoffs that exist when collecting training data and build agents that can intelligently make these tradeoffs in their effort to maximize machine learning performance.

	Worker skill	Task difficulty	Task cost	# Possible Answers
Chapter 2	Varying skill	Varying difficulty	Same cost	2
Chapter 3	Varying skill	Varying difficulty	Same cost	∞
Chapter 4	Same skill	Same difficulty	Same cost	2
Chapter 5	Same skill	Same difficulty	Same cost	2
Chapter 6	Perfect	Not applicable	Varying costs	2

Table 1.1: We strengthen assumptions as we move from chapter to chapter in this dissertation in order to tackle more complex problems. While we begin with models that consider workers of varying skill and tasks of varying difficulty, we move to settings in which we assume crowd responses are independent and identically distributed regardless of the worker and task, and then finally to a setting in which we assume workers are perfect. We also make a number of assumptions common to all chapters. In particular, we assume that tasks have a single objective true answer; workers are not adversarial and do not collaborate; and no mislabeled example, whether for training or testing, is more important than any other mislabeled example.

should keep this slow evolution in mind. While we make a number of assumptions common to all chapters (tasks have a single objective true answer; workers are not adversarial and do not collaborate; and no mislabeled example, whether for training or testing, is more important than any other mislabeled example), in the first part of the dissertation we model tasks of varying difficulty and workers of varying skill level, and in the second part of the dissertation, we first assume that all tasks and workers are the same, and then eventually look at a setting in which workers are perfect. Table 1.1 summarizes these changes in assumptions. We now provide a high-level overview of each of the specific contributions that we make within this larger story.

1.1 Intelligently Gathering Data for Evaluation

In the first part of the dissertation, we improve the intelligent gathering of data for testing purposes by extending two common ideas for more accurately and cheaply acquiring labels: workflow design and label aggregation.

1.1.1 Workflow Switching

To ensure quality results from unreliable crowdsourced workers, task designers often utilize a process in which they construct complex workflows and aggregate worker responses from redundant runs. In this process, they might experiment with several alternative workflows to accomplish the task, and eventually deploy the one that achieves the best performance during early trials. For example, if the task is to translate text from English to German, one workflow might ask workers to directly write a translation, and another workflow might ask workers to correct machine translations [192]. Then, to mitigate the risk of poor data, task designers may dynamically adjust the number of workers they put through that workflow for each task they have. For instance, if 2 workers have high disagreement on a particular labeling of an example, then they may ask another, more trusted, worker.

In Chapter 2, we show that surprisingly, this seemingly natural design paradigm does not achieve the full potential of crowdsourcing. In particular, using a *single* workflow (even the

best) to accomplish a task is suboptimal, because switching between alternative workflows can yield much higher quality output. For example, if one workflow is particularly difficult for a specific task, then switching to an easier one should help performance. We formalize the insight with a novel probabilistic graphical model that considers the difficulty of tasks and the skill of workers, and based on this model, we design and implement a controller based on a Partially Observable Markov Decision Process that dynamically switches between these workflows to achieve higher returns on investment.

Additionally, we design offline and online methods for learning model parameters. Finally, we present live experiments on Amazon Mechanical Turk to demonstrate the superiority of our controller for the task of generating data for NLP purposes, yielding up to 50% error reduction and greater net utility compared to previous state-of-the-art methods.

1.1.2 Label Aggregation for Free-Response Tasks

In our exploration of workflow design, we build a model that is able to aggregate labels from multiple workflows by assuming prior knowledge of all possible outcomes of the task. Many other advanced models for label aggregation also make this assumption [58, 242, 177]. While not an unreasonable assumption for tasks that can be posited as multiple-choice questions (e.g. n-ary classification), many tasks do not naturally fit this paradigm, but instead demand a free-response formulation where the outcome space is of infinite size (e.g. audio transcription).

In Chapter 3, in order to address such tasks and make label aggregation more practical and realistic, we develop a novel probabilistic graphical model that uses the Chinese Restaurant Process [3] to model the probability that a worker will provide a new, unseen label. As in our model for multiple workflows, we also consider difficulty of each task and the skill of each worker. Then, we design and implement a decision-theoretic controller based on this model that dynamically requests responses as necessary in order to infer answers to these tasks. We also show how to jointly learn the parameters of our model while inferring the correct answers to multiple tasks at a time. Live experiments on Amazon Mechanical Turk demonstrate our

controller's superiority at solving SAT Math questions, eliminating 83.2% of the error and achieving greater net utility compared to the state-of-the-art strategy, majority-voting. We also show in live experiments that our model outperforms majority-voting on a visualization task that we design.

1.2 Intelligently Gathering Data for Training

In the second part of the dissertation, we shift our focus from addressing the clean labeling that test sets require to methods for constructing the best possible training sets for machine learning. As we have mentioned, unlike test data, training data does not necessarily need to be perfect. We first address the tradeoff between quantity and noise in training sets, and then shift to methods for maintaining balanced training sets.

1.2.1 Unilabeling Versus Relabeling

Chapter 4 investigates the value of relabeling training examples, because while relabeling examples is a common method for reducing noise, it is unclear whether this strategy is the best use of budget. For example, instead of gathering three labels for every example, one could use the same amount of money to gather three times as many examples. In order to make progress on this tradeoff, we strengthen the assumptions we make in the first part of the dissertation. In particular, we assume that all tasks are of the same difficulty and all workers are independent and identically Bernoulli-distributed. In this setting, while prior research has shown that relabeling can indeed be helpful [198], we first show using datasets from the UCI Machine Learning repository [13] that surprisingly, given a fixed budget, oftentimes classifiers that are trained used large sets of noisy singly-labeled or "unilabeled" data are better than classifiers trained using smaller sets of cleaner relabeled data.

To understand this result, we then make a series of intuitive and theoretical arguments and perform a wide-ranging set of empirical studies on multiple synthetic and real datasets to demonstrate how three different properties of learning problems affect whether relabeling examples is an effective strategy: classifier expressiveness (VC dimension), worker accuracy,

and budget. We show that as classifier expressiveness increases, relabeling strategies become more effective because of the increased possibility of overfitting. Then we show that relabeling strategies are most effective not when workers are extremely accurate or inaccurate, but when they are moderately noisy. Finally, we show that as overall budget increases, unilabeling strategies become more effective because increasing the number of examples can have a noise-canceling effect despite large amounts of noise in the labels.

1.2.2 Re-Active Learning

While the results of Chapter 4 identify factors that influence the utility of relabeling, they do not result in any actionable prescription for actually choosing the amount of relabeling to do during the collection of training data. But also as important as understanding the characteristics of learning problems that make them amenable to relabeling is devising strategies for dynamically controlling the relabeling. An ideal agent should be able to choose which example to newly label or relabel next at every timestep during training.

In Chapter 5, we address the control problem by building upon the long-studied AI subfield of *active learning* [190], which seeks to train the best classifier at the lowest annotation cost by answering variations of the broad question: What is the next best training example to label in order to maximize the long-term performance of the learner? While the ongoing modernization of active learning is quickly updating many of traditional active learning's outdated core assumptions to more closely reflect contemporary practice (*e.g.*, [198][106][249][104]), previous work does not consider the possibility of switching between gathering new examples and relabeling existing ones. We present a generalization of active learning, which we call *re-active learning*, that allows for the possibility of requesting additional independent annotations for a previously labeled example.

Then, we show that while traditional active learning methods can be easily extended to the re-active learning setting, they oftentimes perform extremely poorly at re-active learning, because they often starve examples of labels. To rectify these problems, we present new algorithms designed specifically for re-active learning, and formally characterize their

behavior. We then show that one of our algorithms can be considered a generalization of a popular active learning algorithm to the re-active learning setting. Finally, we empirically show using experiments on real and synthetic datasets that our methods effectively make the re-active learning tradeoff.

1.2.3 Obtaining Balanced Training Sets

In Chapter 6, we move away from the noisy setting by assuming all workers are perfect, and consider how to maintain class balance in training sets, which is also essential for training effective classifiers [237].

All the work that we have outlined so far assumes that the unlabeled datasets we use are balanced in class. Unfortunately, in practice, this assumption is simply not true. Machine learning in high-skew domains, domains with extreme class imbalance, is often extremely difficult, because traditional strategies for obtaining labeled training examples (e.g., active learning) are ineffective at locating the scarce minority-class examples. Imagine that the NSA has an extremely large corpus consisting of the entirety of all Twitter and Facebook posts, and they want to find all sentences that talk about bombings. If they currently don't have a classifier that can identify bombings, and given that the probability that any given sentence in the corpus is positive for "bombing" is vanishingly small, how can they solve this problem?

Researchers have proposed tasking the crowd with finding or generating minority-class examples, but such expensive strategies cannot be broadly applied without thought. In well-balanced domains, crowd generation of examples is far more costly than necessary, because even tasking crowd workers with labeling random examples will result in a balanced training set. And in all domains, relying on such expensive crowd-work in order to achieve a satisfactory class balance may result in a much smaller, and perhaps less effective training set than one created using cheap and traditional crowd-labeling strategies. Thus, any agent that has the power to utilize these multiple strategies for obtaining training examples must carefully weigh their benefits, and switch between them as necessary. In order to make

progress towards such an agent, we tackle the following general problem: Given a budget, an unlabeled corpus, a classifier (possibly untrained), a labeled training set (possibly empty), and a set of strategies for obtaining labeled training examples (*e.g.*, generate examples, label examples selected using active learning), which strategy should one use next in order to obtain another labeled example? We first survey several possible strategies and consider their advantages and disadvantages. Then, we develop bandit-based algorithms that can intelligently use cheaper labeling strategies in low-skew settings while also utilizing more expensive generation strategies in high-skew settings. Finally, through experiments with real and synthetic data, we investigate the value of various strategies and also show that our algorithms outperform state-of-the-art baselines across various domains and skew settings.

1.3 Dissertation Outline

Figure 1.1 shows the organization of the rest of this thesis. In Part I, we discuss the intelligent gathering of testing data. Chapter 2 discusses how to dynamically switch between workflows and Chapter 3 discusses models for aggregating labels from free-response questions. In Part II, we switch to discussing the intelligent gathering of training data. Chapter 4 discusses how various properties of learning problems affect quantity and noise tradeoffs, Chapter 5 investigates re-active learning, and Chapter 6 considers how to cost-effectively construct training sets that are balanced in class. Code and data to reproduce many of the experiments can be found at https://github.com/polarcoconut.

Part I

INTELLIGENTLY GATHERING DATA FROM THE CROWD FOR TESTING

Chapter 2

DYNAMICALLY SWITCHING BETWEEN SYNERGISTIC WORKFLOWS

2.1 Introduction

In this first part of the dissertation, we develop models and algorithms for obtaining test data for machine learning. Because test data must be extremely clean, methods for more cheaply and accurately obtaining labels are extremely important. One such method that is commonly employed is careful workflow design. In this chapter, we focus on improving this method.

Careful workflow design frequently entails a process in which machine learning practitioners first experiment with several alternative workflows to accomplish a task, and then choose a single one for the production runs (*e.g.* the workflow that achieves the best performance during early testing). In the simplest case, alternative workflows may differ only in their user interfaces or instructions. For example, one set of instructions ("Select all correct statements") might be the negation of another ("Select all incorrect statements") [19]. For a more involved task like text-improvement, one workflow may present workers with several different improvements of a text and ask them to select the best one. A second workflow might instead present workers with one improvement and ask them to rate it on a scale from 1 to 10. Other examples include different workflows for text translation [192], alternative interfaces for a task, different ways of wording instructions, and various ways of collecting Natural Language Processing (NLP) tagging data (our use case in this chapter).

After choosing a single workflow, in order to further ensure quality results, practitioners often aggregate worker responses on redundant runs of the workflows [206, 242, 52]. For instance, to determine a "gold label" from the results of the second workflow for text-

improvement, the task designer might select the one with the highest average rating. Unfortunately, this seemingly natural design paradigm does not achieve the full potential of crowdsourcing. Selecting a *single* best workflow is suboptimal, because alternative workflows can compose *synergistically* to attain higher quality results.

Suppose after gathering some answers for a task, one wishes to further increase one's confidence in the results; which workflow should be invoked? Due to the very fact that it is different, an alternative workflow may offer independent evidence and significantly bolster one's confidence in the answer. If the "best" workflow is giving mixed results for a task, then an alternative workflow is often the best way to disambiguate. For instance, in our example above, if workers are having trouble distinguishing between two improvements, one might prefer future workers to provide absolute ratings.

Instead of selecting one a priori best workflow, a better solution should reason about this potential synergy and *dynamically* switch between different workflows. This chapter explains how to do exactly that, making the following contributions:

- We formalize the intuitions underlying workflow-switching with a novel, probabilistic model relating worker responses to the accuracy of workers and the difficulty of alternative workflows for a given task.

- We specify the problem of switching between workflows as a Partially-Observable Markov Decision Process (POMDP), and implement the POMDP policy in our task controller, AgentHunt.

- Optimal control requires estimating the parameter values for the POMDP. We describe two unsupervised methods that use an expectation-maximization (EM) algorithm for learning these latent variables: 1) an offline approach, and 2) an online approach that uses reinforcement learning (RL) to couple learning with control [213].

- We evaluate the benefits of our approach first in a simulated environment and then with live experiments on Amazon Mechanical Turk. We show that AgentHunt outper-

forms the state-of-the-art single-workflow task controller, TurKontrol, [54], achieving up to 50% error reduction and greater net utility for the task of generating data for NLP. Surprisingly, we also show that our adaptive RL method yields almost as high a utility as the approach requiring an explicit training phase.

By using our system, task designers can combine simple training-free deployment with powerful optimized control.

2.2 Background

We first provide a short overview of the AI machinery that we use in this chapter. The material we present here is not meant to be comprehensive, and we provide pointers to appropriate references that provide a more in-depth introduction to these topics.

2.2.1 Partially-Observable Markov Decision Processes

A partially-observable Markov decision process (POMDP) is a model for single-agent decision-making under uncertainty. Formally, a POMDP is a seven-tuple $< \mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, \mathcal{O}, \mathcal{R}, \phi >$, where

- $\mathcal{S}$ is a finite set of unobservable states,
- $\mathcal{A}$ is a finite set of actions,
- Ω is a finite set of observations,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function defining the probability that an agent taking a given action in a given state will transition to another state,
- $\mathcal{O} : \mathcal{S} \times \Omega \rightarrow [0, 1]$ is an observation function defining the probability that an agent receives an observation in a given state,

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function defining the reward that an agent receives from taking an action in a state.

- $\phi \in [0, 1]$ is a discount factor.

At every timestep in the decision process, the agent being modeled exists in a single state, and must choose an action to execute. After executing the action, the agent transitions to another state (which could be the same), and receives an observation, which the agent uses to figure out which state it is currently in, and a reward.

Because the state is unobservable to the agent, the agent maintains a belief state, Λ, a distribution over the state space $\mathcal{S}$, that describes the probability that the agent is in each state. Every time the agent receives an observation $o \in \Omega$ after taking an action $\triangle \in \mathcal{A}$, it computes a new belief state Λ' by computing for every $s' \in \mathcal{S}$, $\Lambda'(s')_{o,\triangle}$, the probability it is currently in state s', using its previous beliefs, Λ, via simple application of Bayes' rule:

$$\Lambda'_{o,\triangle}(s') \propto \mathcal{O}(s', o) \sum_{s \in \mathcal{S}} \mathcal{T}(s, \triangle, s')\Lambda(s).$$

Solving a POMDP entails defining a policy π that specifies an action for every belief state. The optimal policy maximizes the agent's expected future discounted reward: $E[\sum_{\hat{t}=0}^{\infty} \phi^{\hat{t}} r_{\hat{t}}]$ where $\hat{t}$ is the timestep and $r_{\hat{t}}$ is the random variable describing the reward the agent receive at time $\hat{t}$.

Unfortunately, solving a POMDP exactly by finding the optimal policy is a difficult problem. Finding an approximate solution in a scalable manner is the subject of much current research. Many methods work by attempting to estimate the optimal *value function* of each belief state, $V^*(\Lambda)$, which satisfies the following system of equations:

$$V^*(\Lambda) = \max_{\triangle \in \mathcal{A}} \sum_{s \in \mathcal{S}} \Lambda(s)\mathcal{R}(s, \triangle) + \phi \sum_{o \in \Omega} [\sum_{s \in \mathcal{S}} \Lambda(s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \triangle, s')\mathcal{O}(s', o)]V^*(\Lambda'_{o,\triangle}).$$

One of the most popular solution algorithms are the point-based value iteration methods [169, 170, 197], which work by only using a subset of all possible belief states to estimate the value function and then inferring the value function of the rest of the belief states. These

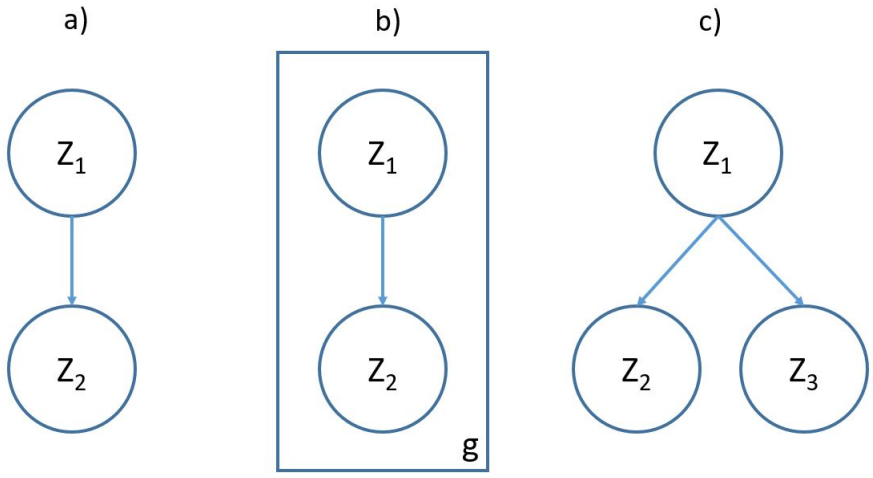

Figure 2.1: a) A Bayesian network with 2 random variables. b) A Bayesian network with $2g$ random variables. c) A Bayesian network in which Z_2 and Z_3 are conditionally independent given Z_1.

methods differ in the way they select subsets of belief states and the way they use those belief states to compute the value function.

2.2.2 Bayesian Networks

A Bayesian network, or Bayes net, is a directed acyclic graph that encodes a probability distribution. Nodes in the graph represent random variables, and directed edges between nodes represent dependencies between those random variables. The structure of a Bayes net limits the expressiveness of the joint probability distribution of the random variables. If $Z_1, \ldots, Z_n$ are random variables, and $\Upsilon(Z)$ is the set of parents of Z, then the joint probability distribution of the Bayes Net is defined as:

$$P(Z_1, \ldots, Z_n) = \prod_z P(Z_z | \Upsilon(Z_z))$$

Thus to fully define a joint probability distribution, one only needs to define the conditional distributions of each random variable given its parents.

Figure 2.1a shows an example of a Bayes net with 2 random variables, Z_1 and Z_2. Because of the directed edge from Z_1 to Z_2, the joint probability distribution of these two variables is restricted to be of the form $P(Z_1, Z_2) = P(Z_1)P(Z_2|Z_1)$. Figure 2.1b shows an example of *plate notation*, in which a rectangle drawn around a model represents duplication of all nodes and edges inside that rectangle. In this case, the plate notation shows g independent and identically distributed copies of the model in Figure 2.1a.

Bayes nets can also make quickly seeing or defining conditional independence assumptions easy [76]. For example, the structure of the Bayes net in Figure 2.1c implies that Z_2 and Z_3 are conditionally independent given Z_1, or more formally, $P(Z_2, Z_3|Z_1) = P(Z_2|Z_1)P(Z_3|Z_1)$.

2.2.3 Dai et al.'s Model for Crowd Work

Many of the commonly employed jobs in crowdsourcing may be modeled abstractly as the task of selecting a single correct answer from a set of alternatives. Labeling classification data or choosing the best of two artifacts are two examples. For this scenario, several previous researchers have designed probabilistic models that combine multiple answers from noisy workers [198, 242, 52].

We follow and extend Dai *et al.*'s probabilistic generative model for crowd worker responses to tasks with 2 answer choices [52, 54]. The model defines the accuracy of a worker's answer to be: $\xi(d, \gamma_w) = \frac{1}{2}[1 + (1 - d)^{\gamma_w}]$, where d is the *inherent difficulty* of the task and γ_w is the *error parameter* of worker w. As d and γ increase, the probability that the worker produces the correct answer approaches 0.5, suggesting that the worker is guessing randomly. On the other hand, as d and γ decrease, ξ approaches 1, when the worker will deterministically produce the correct answer.

Figure 2.2 defines a Bayes net for the model with W workers and Ψ tasks. The worker's answer b depends on the the difficulty of the task d, the worker's error parameter γ, and the hidden true answer v. The probability that the worker's answer b is the same as v for a given task is $P(b_w = v) = \xi(d, \gamma_w)$, and the probability that b is not the same as v is $P(b_w \neq v) = 1 - \xi(d, \gamma_w)$. The Bayes net encodes the assumption that for a particular task,

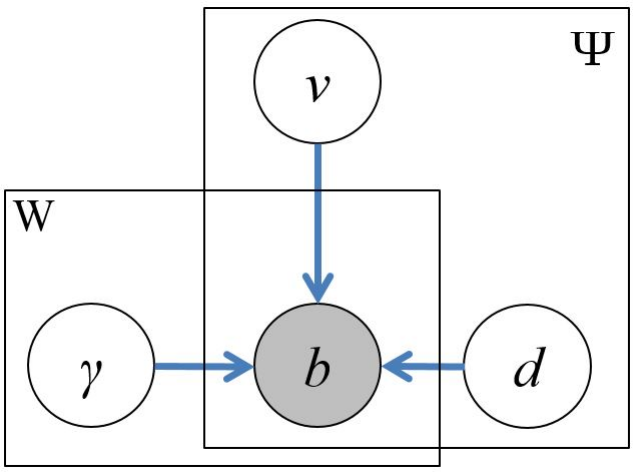

Figure 2.2: A worker's answer b depends on the difficulty of the question d, the worker's error parameter γ and the question's true answer v. There are W workers, who complete Ψ tasks. b is the only observed variable.

given the difficulty of that task d, each worker's correctness is independent of each other worker's correctness.

2.3 Probabilistic Model for Multiple Workflows

We now present a probabilistic model for multiple workflows. One of our key contributions in this chapter (Figure 2.3) is to extend Dai *et al.*'s [52, 54] probabilistic generative model to allow for the existence of multiple workflows for completing the same task. It includes multiple, workflow-specific error parameters for each worker and workflow-specific difficulties.

For our model, there are K alternative workflows that a worker could use to arrive at an answer. Let $d^k \in [0, 1]$ denote the inherent difficulty of completing a task using workflow k, and let $\gamma_w^k \in [0, \infty)$ be worker w's error parameter for workflow k. Notice that every worker has K error parameters. Having several parameters per worker incorporates the insight that some workers may perform well when asked the question in one way (*e.g.*, visually) but not so well when asked in a different way (*e.g.*, when asked in English, since their command of that language may not be great).

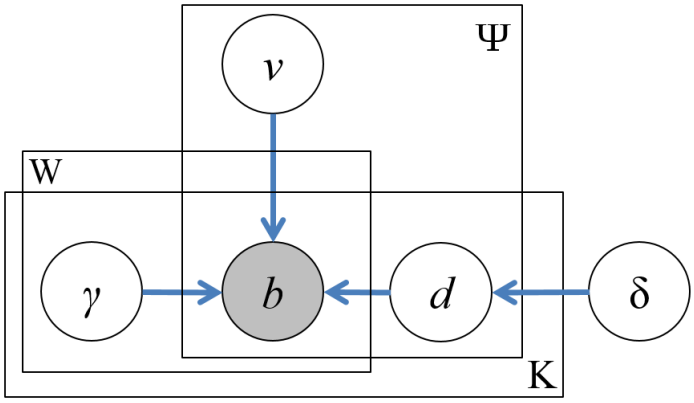

Figure 2.3: A worker's answer b depends on the difficulty of the question d (generated by δ), the worker's error parameter γ and the question's true answer v. There are W workers, who complete Ψ tasks, all of which can be solved using a set of K different workflows. b is the only observed variable.

The accuracy of a worker w, $\xi(d^k, \gamma_w^k)$, is the probability that she produces the correct answer using workflow k. We rewrite Dai *et al.*'s definition of worker accuracy accordingly:

$$\xi(d^k, \gamma_w^k) \quad = \quad \frac{1}{2}\left(1 + (1 - d^k)^{\gamma_w^k}\right)$$

As a worker's error parameter and/or the workflow's difficulty increases, ξ approaches 1/2, suggesting that worker is randomly guessing. On the other hand, as the stated parameters decrease, ξ approaches 1, when the worker always produces the correct answer.

Figure 2.3 illustrates the plate notation for our generative model, which encodes a Bayes Net for responses made by W workers on Ψ tasks, all of which can be solved using a set of K alternative workflows. The correct answer, v, the difficulty parameter, d, and the error parameter, γ, influence the final answer, b, that a worker provides, which is the only observed variable. d is generated by δ, a K-dimensional random variable describing a joint distribution on workflow difficulties. The answer b_w^k that worker w with error parameter γ_w^k provides for a task using workflow k is governed by the following equations:

$$
\begin{aligned}
P(b_w^k = v | d^k) &= \xi(d^k, \gamma_w^k) \\
P(b_w^k \neq v | d^k) &= 1 - \xi(d^k, \gamma_w^k)
\end{aligned}
$$

An underlying assumption is that given the workflow difficulty, d^k, the b_w^k's are independent of each other. This is in line with Dai *et al.*'s assumptions for the single workflow case. δ encodes the assumption that workflows may not be independent of each other. The fact that one workflow is easy might imply that a related workflow is easy. Finally, we assume that the workers do not collaborate with each other and that they are not adversarial, *i.e.*, they do not purposely submit incorrect answers.

The availability of multiple workflows with independent difficulties introduces the possibility of dynamically switching between them to obtain the highest accuracy for a given task. We now discuss our approach for taking advantage of this possibility.

2.4 A Decision-Theoretic Agent

In this section, we answer the following question: given a specific task that can be accomplished using alternative workflows, how do we design an agent that can leverage the availability of these alternatives by dynamically switching between them, in order to achieve a high quality solution? We design an automated agent, named AgentHunt, that uses a POMDP [207, 183] to capture all the assumptions of our problem.

For AgentHunt, a state $s \in \mathcal{S}$ in the POMDP is a $K+1$ tuple $(d^1, d^2, \ldots, d^K, v)$, where d^k is the difficulty of the k^{th} workflow and v is the true answer of the task. Notice that AgentHunt can not observe any component of its state. At each time step, AgentHunt has a choice of $K+2$ actions ($|\mathcal{A}| = K+2$). It can submit one of two possible answers or create a new job with any of the K workflows. The process terminates when AgentHunt submits any answer. When AgentHunt creates a new job using workflow k, it will receive an observation b_w^k containing one of the 2 answers chosen by some worker w with observation probability $\mathcal{O}$ dictated by our model. This information allows AgentHunt to update its

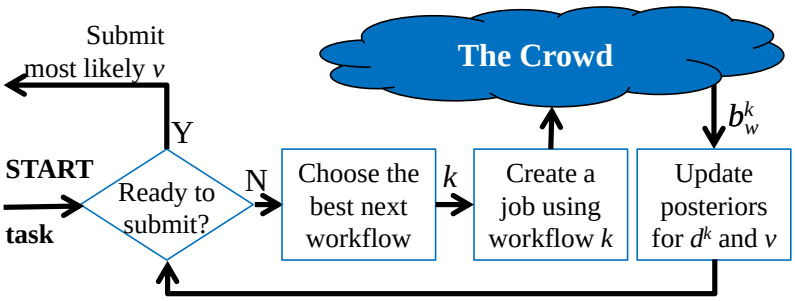

Figure 2.4: AGENTHUNT's decisions when executing a task.

belief about the state using b_w^k and its knowledge of γ_w^k. Figure 2.4 is a flow-chart of decisions that AGENTHUNT has to take.

None of the actions in $\mathcal{A}$ changes what world-state AGENTHUNT is in; this POMDP is a purely sensing POMDP, so the transition function $\mathcal{T}$ maps every state to itself, regardless of the action. The reward function $\mathcal{R}$ maintains the value of submitting a correct answer and the penalty for submitting an incorrect answer. Additionally, it maintains a cost that AGENTHUNT incurs when it creates a job. We can modify the reward function to match our desired budgets and accuracies.

In many crowdsourcing platforms, such as Mechanical Turk, we cannot preselect the workers to answer a job. However, in order to specify our observation probabilities, which are defined by our generative model, we need access to future workers' parameters. To simplify the computation, our POMDP assumes that every future worker is an average worker. In other words, for a given workflow k, every future worker has an error parameter equal to $\overline{\gamma}^k = \frac{1}{W}\sum_w \gamma_w^k$ where W is the number of workers.

After submitting an answer, AGENTHUNT can update its records about all the workers who participated in the task using what it believes to be the correct answer. We follow the approach of Dai *et al.* [52], using the following update rules: For a worker w who submitted an answer using workflow k, $\gamma_w^k \leftarrow \gamma_w^k - d^k\sigma$, should the worker answer correctly, and $\gamma_w^k \leftarrow \gamma_w^k + (1-d^k)\sigma$, should the worker answer incorrectly, where σ is a learning rate. Any

worker that AgentHunt has not seen previously begins with the average $\overline{\gamma}^k$.

2.5 Learning the Model

In order to behave optimally, AgentHunt needs to learn all $\boldsymbol{\gamma}$ values, average worker error parameters $\overline{\boldsymbol{\gamma}}$, and the joint workflow difficulty prior δ, which is a part of its initial belief. We consider two unsupervised approaches to learning, offline batch learning and online RL.

2.5.1 Offline Learning

In this approach we first collect training data by having a set of workers complete a set of tasks using a set of workflows. This generates a set of worker responses, $\boldsymbol{b}$. Since the true answer values $\boldsymbol{v}$ are unknown, an option is supervised learning, where experts label true answers and difficulties. This approach was used by Dai *et al.* [54]. But such an approach is not a scalable option, since it requires significant expert time upfront.

In contrast, we use an EM algorithm, similar to that proposed by Whitehill *et al.* [242] to learn all parameters jointly. For EM purposes, we simplify the model by removing the joint prior δ, and treat the variables $\mathbf{d}$ and $\boldsymbol{\gamma}$ as parameters. We alternate between the following E-step and M-step until the parameters converge. Let old parameters be denoted with a dot (*e.g.* $\dot{\mathbf{d}}$). In the E-step, we keep old parameters fixed to compute the posterior probabilities of the hidden true answers $P(v_t|\mathbf{b}\dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}})$ for each task t:

$$\begin{aligned} P(v_t|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}) &= P(v_t|\mathbf{b_t}, \dot{\mathbf{d}}_\mathbf{t}, \dot{\boldsymbol{\gamma}}_\mathbf{t}) \\ &\propto P(\mathbf{b_t}|v_t, \dot{\mathbf{d}}_\mathbf{t}, \dot{\boldsymbol{\gamma}}_\mathbf{t}) P(v_t|\dot{\mathbf{d}}_\mathbf{t}, \dot{\boldsymbol{\gamma}}_\mathbf{t}) \\ &= P(\mathbf{b_t}|v_t, \dot{\mathbf{d}}_\mathbf{t}, \dot{\boldsymbol{\gamma}}_\mathbf{t}) P(v_t) \\ &= P(v_t) \prod_{w,k} P(b^k_{t,w}|v_t, \dot{d^k_t}, \dot{\gamma^k_w}) \end{aligned}$$

Next, the M-step uses these posteriors to find new parameters that maximize the standard

expected complete log-likelihood L over $\mathbf{d}$ and $\boldsymbol{\gamma}$:

$$L(\mathbf{d}, \boldsymbol{\gamma}) = E[\ln p(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma})],$$

where the expectation is taken over $P(\mathbf{v}|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}})$:

$$\begin{aligned}
& E[\ln P(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma}) \\
= \;& E[\ln P(\mathbf{v}|\mathbf{d}, \boldsymbol{\gamma}) P(\mathbf{b}|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma})] \\
= \;& E[\ln P(\mathbf{v}) P(\mathbf{b}|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma})]) \\
= \;& E[\ln \prod_t P(v_t) P(\mathbf{b}_t|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma})] \\
= \;& E[\ln \prod_t P(v_t) \prod_{wk} P(b^k_{t,w}|v_t, d^k_t, \gamma^k_w,)] \\
= \;& E[\sum_t \ln P(v_t) + \sum_{wk} \ln P(b^k_{t,w}|v_t, d^k_t, \gamma^k_w)] \\
= \;& \sum_t E[\ln P(v_t)] + \sum_{wk} E[\ln P(b^k_{t,w}|v_t, d^k_t, \gamma^k_w) \\
= \;& \sum_t \sum_{a=0}^{1} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}) \ln P(v_t = a) + \\
& \sum_{wk} \sum_{a=0}^{1} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}) \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma^k_w)
\end{aligned}$$

In order to find the parameters that maximize L, we can differentiate with respect to the parameters and apply iterative minimization algorithms like conjugate gradient methods (*e.g.* [74]). Differentiating with respect to $\boldsymbol{d}$, we have that

$$\frac{\partial L}{\partial d^k_t} = \sum_w \sum_{a=0}^{1} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}) \frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma^k_w)}{\partial d^k_t},$$

where if $b^k_{t,w} = v_t$

$$\begin{aligned}
\frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma_w)}{\partial d^k_t} &= \frac{\partial \ln \frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)}{\partial d^k_t} \\
&= \frac{\frac{-1}{2}\gamma^k_w (1 - d^k_t)^{\gamma^k_w - 1}}{\frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)},
\end{aligned}$$

and otherwise,

$$\begin{aligned}\frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma_w)}{\partial d^k_t} &= \frac{\partial \ln 1 - \frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)}{\partial d^k_t} \\ &= \frac{\frac{1}{2}\gamma^k_w (1 - d^k_t)^{\gamma^k_w - 1}}{1 - \frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)}.\end{aligned}$$

And differentiating with respect to $\boldsymbol{\gamma}$, we have that

$$\frac{\partial L}{\partial \gamma^k_w} = \sum_t \sum_{a=0}^{1} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}) \frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma^k_w)}{\partial \gamma^k_w},$$

where if $b^k_{t,w} = v_t$

$$\frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma^k_w)}{\partial \gamma^k_w} = \frac{\frac{1}{2}(1 - d^k_t)^{\gamma^k_w} \ln(1 - d^k_t)}{\frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)},$$

and otherwise,

$$\frac{\partial \ln P(b^k_{t,w}|v_t = a, d^k_t, \gamma^k_w)}{\partial \gamma^k_w} = \frac{\frac{-1}{2}(1 - d^k_t)^{\gamma^k_w} \ln(1 - d^k_t)}{1 - \frac{1}{2}\left(1 + (1 - d^k_t)^{\gamma^k_w}\right)}.$$

After estimating all the hidden parameters, AgentHunt can compute $\overline{\gamma}^k$ for each workflow k by taking the average of all the learned $\boldsymbol{\gamma}^k$ parameters. Then, to learn δ, we can fit a Truncated Multivariate Normal distribution to the learned $\boldsymbol{d}$. This difficulty prior determines a part of the initial belief state of AgentHunt. We complete the initial belief state by assuming the correct answer is distributed uniformly among the 2 alternatives.

2.5.2 Online Reinforcement Learning

Offline learning of our model can be very expensive, both temporally and monetarily. Moreover, we can not be sure how much training data is necessary before the agents are ready to act in the real-world. An ideal AI agent can learn while acting in the real world, tune its parameters as it acquires more knowledge, while still producing meaningful results. We modify AgentHunt to build its RL twin, $\text{AgentHunt}_{\text{RL}}$, which is able to accomplish tasks right out of the box.

AgentHunt$_{\mathrm{RL}}$ starts with uniform priors on difficulties of all workflows. When it begins a new task, it uses the existing parameters to recompute the best policy and uses that policy to guide the next set of decisions. After completing the task, AgentHunt$_{\mathrm{RL}}$ recalculates the maximum-likelihood estimates of the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{d}$ using EM as above. The updated parameters define a new POMDP for which our agent computes a new policy for the future tasks. This relearning and POMDP-solving can be time-consuming, but we do not have to relearn and resolve after completing every task. We can easily speed the process by solving a few tasks before launching a relearning phase.

As in all of RL, AgentHunt$_{\mathrm{RL}}$ must also make a tradeoff between taking possibly suboptimal actions in order to learn more about its model of the world (exploration), or taking actions that it believes to be optimal (exploitation). AgentHunt$_{\mathrm{RL}}$ uses a modification of the standard ϵ-greedy approach [213]. With probability ϵ, AgentHunt$_{\mathrm{RL}}$ will uniformly choose between suboptimal actions. The exception is that it will never submit an answer that it believes to be incorrect, since doing so would not help it learn anything about the world.

2.6 Experiments

This section addresses the following three questions.

1. In practice, how much value can be gained from switching between different workflows for a task?

2. What is the tradeoff between cost and accuracy?

3. Previous decision-theoretic crowdsourcing systems have required an initial training phase; can reinforcement learning provide similar benefits without such training?

We choose an NLP labeling task, for which we create $K = 2$ alternative workflows (described below). To answer the first two questions, we compare two agents: TurKontrol, a state-of-the-art controller for optimizing the execution of a single (best) workflow [52], and our

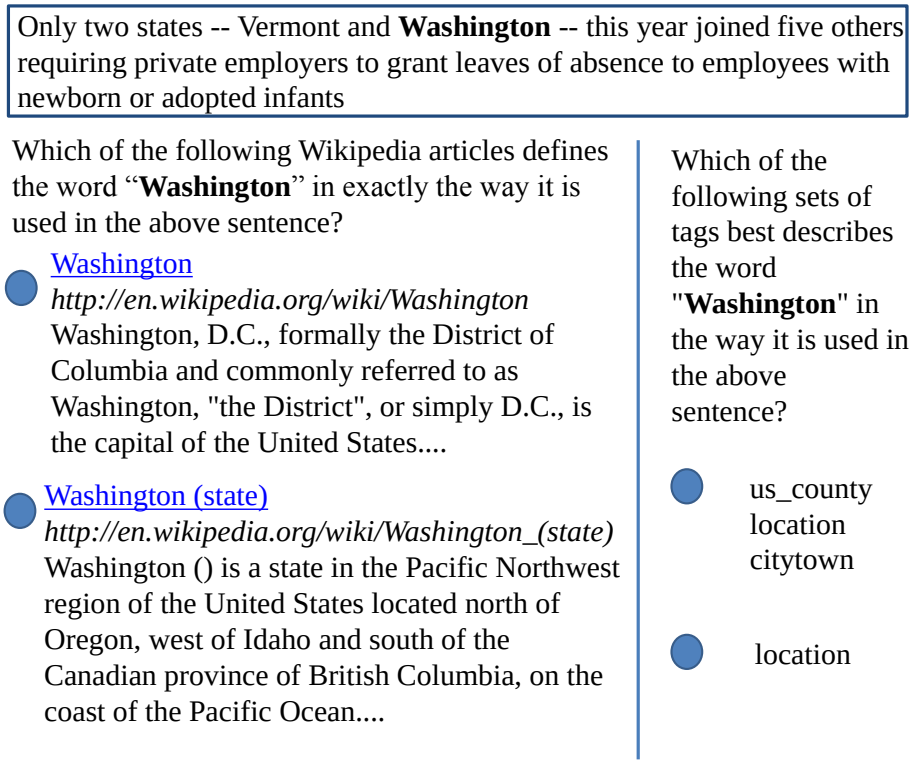

Figure 2.5: In this NER task, the TagFlow is considerably harder than the WikiFlow since the tags are very similar. The correct tag set for this task is {location} since Washington State is neither a county nor a citytown.

AgentHunt, which can switch between the two workflows dynamically. We first compare them in simulation; then we allow the agents to control live workers on Amazon Mechanical Turk. We answer the third question by comparing AgentHunt with $\text{AgentHunt}_{\text{RL}}$.

2.6.1 Implementation

The POMDP must manage a belief state over the cross product of the Boolean answer and the two continuous difficulties of the two workflows. Since solving a POMDP with a continuous state space is challenging, we discretize difficulty into eleven possible values, leading to a (world) state space of size $2 \times 11 \times 11 = 242$. To solve POMDPs, we run the ZMDP package

[203] for 300 seconds using the default Focused Real-Time Dynamic Programming search strategy [204]. Since we can cache the complete POMDP policy in advance, AgentHunt can control workflows in real time.

Since we have discretized difficulty, we also modify the offline learning process slightly. After we learn all values of $\mathbf{d}$, we round the values to the nearest discretizations and construct a histogram to count the number of times every state appears in the training data. Then, before we use the implicit joint distribution as the agent's starting belief state, we smooth it by adding 1 to every bin (Laplace smoothing).

2.6.2 Evaluation Task: NER Tagging

In order to test our agents, we select a task that is needed by several colleagues: named-entity recognition (NER) [187]. NER tagging is a common problem in NLP and information extraction: given a body of text (*e.g.*, "Barack Obama thinks this research is not bad.") and a subsequence of that text (*e.g.*, "Barack Obama") that specifies an entity, output a set of tags that classify the type of the entity (*e.g.*, *person*, *politician*).

The Two Workflows

In consultation with NER domain experts we develop two workflows for the task (Figure 2.5). Both workflows begin by providing users with a body of text and an entity, like "**Nixon** concluded five days of private talks with Chinese leaders in Beijing." The first workflow, called "WikiFlow," first uses *Wikification* [155, 176] to find a set of possible Wikipedia articles describing the entity, such as "Nixon (film)" and "Richard Nixon." It displays these articles (including the first sentence of each article) and asks workers to choose the one that best describes the entity. Finally, it returns the Freebase[1] tags associated with the Wikipedia article selected by the worker.

The second workflow, "TagFlow," asks users to choose the best set of Freebase tags

[1] *www.freebase.com*

directly. For example, the single Freebase tag associated with "Nixon (film)" is `/film/film`, while the tags associated with "Richard Nixon" are `/people/person`, `/base/crime/lawyer`, and `/government/us-congressperson`, TagFlow displays the sets of tags corresponding to the different options and asks the worker to choose the set that best describes the entity mentioned in the sentence.

2.6.3 Experimental Setup

First, we gather data using Mechanical Turk. We generate 50 NER tasks. For each task, we submit 40 identical WikiFlow jobs and 40 identical TagFlow jobs to Mechanical Turk. At \$0.01 per job, the total cost is \$60.00 including Amazon commission. Using our EM technique, we then calculate average worker accuracies, $\overline{\gamma}^{WF}$ and $\overline{\gamma}^{TF}$, corresponding to WikiFlow and TagFlow respectively. Somewhat to our surprise, we find that $\overline{\gamma}^{TF} = 0.538 < \overline{\gamma}^{WF} = 0.547$. On average, workers found TagFlow to be very slightly easier than WikiFlow. Note that this result implies that AgentHunt will always create a TagFlow job to begin a task. We also note that the difference between $\overline{\gamma}^{TF}$ and $\overline{\gamma}^{WF}$ controls the switching behavior of AgentHunt. Intuitively, if AgentHunt were given two workflows whose average difficulties were further apart, AgentHunt would become more reluctant to switch to a harder workflow. Because we find TagFlow jobs to be slightly easier, for all experiments, we set TurKontrol so it creates TagFlow jobs. We also use this data to construct both agents' initial beliefs.

2.6.4 Experiments using Simulation

We first run AgentHunt and TurKontrol in a simulated environment. On each run, the simulator draws states from the agents' initial belief distributions. We fix the reward of returning the correct answer to 0, and vary the reward (penalty) of returning an incorrect answer between the following values: -10, -100, -1,000, and -10,000. We set the cost of creating a job for a (simulated) worker to -1. We use a discount factor of 0.9999 in the

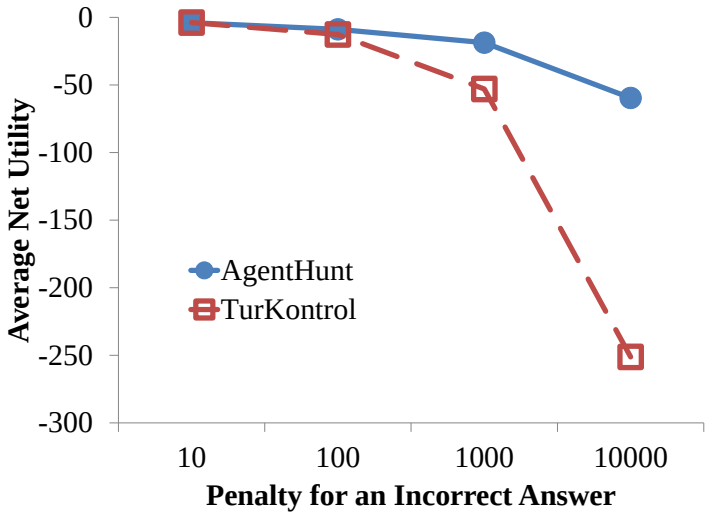

Figure 2.6: In simulation, as the importance of answer correctness increases, AgentHunt outperforms TurKontrol by an ever-increasing margin.

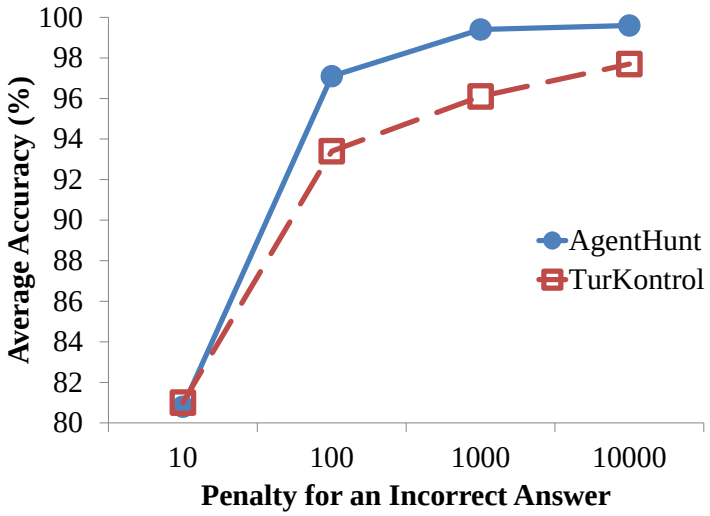

Figure 2.7: In simulation, as the importance of answer correctness increases, both agents converge to 100 percent accuracy, but AgentHunt does so more quickly.

POMDP so that the POMDP solver converges quickly. For each setting of reward values we run 1,000 simulations and report mean net utilities (Figure 2.6).

We find that when the stakes are low, the two agents behave almost identically. However, as the penalty for an incorrect answer increases, AgentHunt's ability to switch between workflows allows it to capture much more utility than TurKontrol. As expected, both agents submit an increasing number of jobs as the importance of answer correctness rises and in the process, both of their accuracies rise too (Figure 2.7). However, while both agents become more accurate, TurKontrol does not increase its accuracy enough to compensate for the exponentially growing penalties. Instead, as AgentHunt experiences an almost sublinear decline in net utility, TurKontrol sees an exponential drop (Figure 2.6). A Student's t-test shows that for all settings of penalty except -10, the differences between the two systems' average net utilities are statistically significant. When the penalty is -10, $p < 0.4$, and at all other reward settings, $p < 0.0001$. Thus we find that at least in simulation, AgentHunt outperforms TurKontrol on all our metrics.

We also analyze the systems' behaviors qualitatively. As expected, AgentHunt always starts by creating a TagFlow job, since $\overline{\gamma}^{TF} < \overline{\gamma}^{WF}$ implies that TagFlows lead to higher worker accuracy on average. Interestingly, although AgentHunt has more available workflows, it creates fewer actual jobs than TurKontrol, even as correct answers become increasingly important. We also split our problems into three categories to better understand the agents' behaviors: 1) TagFlow is easy, 2) TagFlow is hard, but WikiFlow is easy, and 3) both workflows are difficult.

In the first case, both agents terminate quickly, though AgentHunt spends a little more money since it also requests WikiFlow jobs to double-check what it learns from TagFlow jobs. In the second case, TurKontrol creates an enormous number of jobs before it decides to submit an answer, while AgentHunt terminates much faster, since it quickly deduces that TagFlow is hard and switches to creating easy WikiFlow jobs. In the third case, AgentHunt expectedly creates more jobs than TurKontrol before terminating, but AgentHunt does not do too much worse than TurKontrol, since it correctly deduces that gathering more

	AgentHunt	TurKontrol	TurKontrol$_{300}$	AgentHunt$_{RL}$
Avg Accuracy (%)	92.45	85.85	84.91	93.40
Avg Cost	5.81	4.21	6.26	7.25
Avg Net Utility	-13.36	-18.35	-21.35	-13.85

Table 2.1: Comparisons of accuracies, costs, and net utilities of various agents when run on Mechanical Turk.

information is unlikely to help.

2.6.5 Experiments using Mechanical Turk

We next run the agents on real data gathered from Mechanical Turk. We generate 106 new NER tasks for this experiment, and use gold labels supplied by a single expert. Since in our simulations we found that the agents spend on average about the same amount of money when the reward for an incorrect answer is -100, we use this reward value in our real-world experiments.

As Table 2.1 shows, AgentHunt fares remarkably better in the real-world than TurKontrol. A Student's t-test shows that the difference between the average net utilities of the two agents is statistically significant with $p < 0.03$. However, we see that TurKontrol spends less, leading one to naturally wonder whether the difference in utility can be accounted for by the cost discrepancy. Thus, we modify the reward for an incorrect answer (to -300) for TurKontrol to create TurKontrol$_{300}$, which spends about the same amount of money as AgentHunt.

But even after the modification, the accuracy of AgentHunt is still much higher. A Student's t-test shows that the difference between the average net utilities of AgentHunt and TurKontrol$_{300}$ is statistically significant at $p < 0.01$ showing that in the real-world, given similar budgets, AgentHunt produces significantly better results than TurKontrol. Indeed, AgentHunt reduces the error of TurKontrol by 45% and the error of

TurKontrol$_{300}$ by 50%. Surprisingly, the accuracy of TurKontrol$_{300}$ is lower than that of TurKontrol despite the additional jobs; we attribute this to statistical variance.

2.6.6 Adding Reinforcement Learning

Finally, we compare AgentHunt to AgentHunt$_{\mathrm{RL}}$. AgentHunt$_{\mathrm{RL}}$'s starting belief state is a uniform distribution over all world states and it assumes that $\overline{\gamma}^k = 1$ for all workflows. To encourage exploration, we set $\epsilon = 0.1$. We test it using the same 106 tasks described above. Table 2.1 shows that while AgentHunt$_{\mathrm{RL}}$ achieves a slightly higher accuracy than AgentHunt, the difference between their net utilities is not statistically significant ($p = 0.4$), which means AgentHunt$_{\mathrm{RL}}$ is comparable to AgentHunt, suggesting that AgentHunt can perform in an "out of the box" mode, without needing a training phase.

2.7 Related Work

The benefits from combining disparate workflows have been previously observed. Babbage's Law of Errors suggests that the accuracy of numerical calculations can be increased by comparing the outputs of two or more methods [81]. However, in previous work these workflows have been combined manually; AgentHunt embodies the first method for *automatically* evaluating potential synergy and dynamically switching between workflows.

Modeling repeated labeling in the face of noisy workers has received significant attention. Romney *et al.* [179] are one of the first to incorporate a worker accuracy model to improve label quality. Sheng *et al.* [198] explore when it is necessary to get another label for the purpose of machine learning. Raykar *et al.* [177] propose a model in which the parameters for worker accuracy depend on the true answer. Whitehill *et al.* [242] and Dai *et al.* [52] address the concern that worker labels should not be modeled as independent of each other unless given problem difficulty. Welinder *et al.* [240] design a multidimensional model for workers that takes into account competence, expertise, and annotator bias. Kamar *et al.* [104] extracts features from the task at hand and use Bayesian Structure Learning to learn

the worker response model. Parameswaran *et al.* [164] conduct a policy search to find an optimal dynamic control policy with respect to constraints like cost or accuracy. Karger *et al.* [109] develop an algorithm based on low-rank matrix approximation to assign tasks to workers and infer correct answers, and analytically prove the optimality of their algorithm at minimizing a budget given a reliability constraint.

Snow *et al.* [206] show that for labeling tasks, a small number of Mechanical Turk workers can achieve an accuracy comparable to that of an expert labeler. For more complex tasks, innovative workflows have been designed, for example, an iterative improvement workflow for creating complex artifacts [141], find-fix-verify for an intelligent editor [24], iterative dual pathways for speech-to-text transcription [130] and others for counting calories on a food plate [163]. Lasecki *et al.* [124] design a system that allows multiple users to control the same interface in real-time. Control can be switched between users depending on who is doing better. Kulkarni *et al.* [120] show the crowd itself can help with the design and execution of complex workflows.

An AI agent makes an efficient controller for these crowdsourced workflows. Dai *et al.* [52, 54] create a POMDP-based agent to control an iterative improvement workflow. Shahaf and Horvitz [192] develop a planning-based task allocator to assign subtasks to specific humans or computers with known abilities.

Weld *et al.* [239] discuss a broad vision for the use of AI techniques in crowdsourcing that includes workflow optimization, interface optimization, workflow selection and intelligent control for general crowdsourced workflows. Our work reifies their proposal for workflow selection.

2.8 Conclusion

We demonstrate that alternative workflows can compose synergistically to produce much higher quality results from crowdsourced workers. We design AgentHunt, a POMDP-based agent that dynamically switches between these workflows to obtain the best cost-quality tradeoffs. Live experiments on Mechanical Turk demonstrate the effectiveness of Agen-

tHunt. At comparable costs, it yields up to 50% error reduction compared to TurKontrol, a strong baseline agent that uses the best single workflow. Moreover, for a new task, AgentHunt can operate out of the box since it does not require any explicit learning phase to tune its parameters.

Chapter 3

MOVING BEYOND MULTIPLE CHOICE

3.1 Introduction

In the last chapter, we took workflow design, a common method for increasing the quality of test labels, and made it more impactful by designing a control model that allows for switching between alternative workflows in order to cut costs and increase accuracy.

In order to aggregate labels from multiple workers, our model, like many others [52, 242], assumed that tasks are posed to workers as multiple choice questions, where every alternative answer is known in advance and the worker has to simply select one. While many tasks can indeed be posed in a multiple-choice fashion (*e.g.* n-ary classification), a large number of tasks exist that can only be formulated with an unbounded number of possible answers. A common example is completing a database with workers' help, *e.g.*, asking questions such as "Find the mobile phone number of Acme Corporation's CEO." Since the space of possible number of answers is huge (possibly infinite), the task interface cannot explicitly enumerate them for the worker. We refer to these tasks as *open questions*.

Unfortunately, adapting multiple-choice label aggregation models for open questions is not straightforward, because of the difficulty with reasoning about unknown answers. Machine learning practitioners, therefore, must resort to using a majority-vote, a significant hindrance to achieving quality results from these more general open questions.

In this chapter, we tackle the challenging problem of modeling tasks where workers are free to give any answer. In particular, we make the following contributions:

- We propose a novel, probabilistic model relating the accuracy of workers and task difficulty to worker responses, which are generated from a countably infinite set.

- We design a decision-theoretic controller, LazySusan, to dynamically infer the correct answer to a task by only soliciting more worker responses when necessary. We also design an EM algorithm to jointly learn the parameters of our model while inferring the correct answers to multiple tasks at a time.

- We evaluate variations of our approach first in a simulated environment and then with live experiments on Amazon Mechanical Turk. We show that LazySusan outperforms state-of-the-art majority-voting, achieving 83.2% error reduction and greater net utility for the task of solving SAT Math questions. We also show in live experiments that our EM algorithm outperforms majority-voting on a visualization task that we design.

3.2 Background

There exist many models that tackle the problem of inferring a correct answer for a task with a finite number of possible answers. Our work is based on the model of Dai *et al.* [52], which we use in the last chapter and briefly review again here. Their model assumes that there are exactly 2 possible answers (binary classification). Let $d \in [0, 1]$ denote the inherent *difficulty* of completing a given task, and let $\gamma_w \in [0, \infty)$ be worker w's innate proneness to *error*. The accuracy of a worker on a task, $\xi(d, \gamma_w)$, is defined to be the probability that she produces the correct answer using the following model: $\xi(d, \gamma_w) = \frac{1}{2}(1 + (1 - d)^{\gamma_w})$. As d and γ increase, the probability that the worker produces the correct answer approaches 0.5, suggesting that she is guessing randomly. On the other hand, as d and γ decrease, ξ approaches 1, when the worker will deterministically produce the correct answer. A significant limitation of this model is its inability to handle tasks with an infinite number of possible answers.

3.2.1 Chinese Restaurant Process

In order to address this limitation, we use the Chinese Restaurant Process [3], a discrete-time stochastic process that generates an infinite number of labels ("tables"). Intuitively, the process may be thought of as modeling sociable customers who, upon entering the restaurant,

decide between joining other diners at a table or starting a new table. The greater the number of customers sitting at a table, the more likely new customers will join that table.

Formally, a Chinese Restaurant $R = (\mathrm{T}, f, \theta)$ is a set of occupied tables $\mathrm{T} = \{\tau_1, \ldots, \tau_n | \tau_i \in \mathbb{N}\}$, a function $f : \mathrm{T} \to \mathbb{N}$ that denotes the number of customers at each table τ_i, and a parameter $\theta \in \mathbb{R}^+$. Imagine that a customer arrives at the restaurant. He can either choose to sit at one of the occupied tables, or at a new empty table. The probability that he chooses to sit at an occupied table $\tau \in \mathrm{T}$ is

$$C_R(\tau) = \frac{f(\tau)}{N + \theta}$$

where $N = \sum_{\tau \in \mathrm{T}} f(\tau)$ is the total number of customers in the restaurant. The probability that he chooses to begin a new table, or, equivalently, the probability that he chooses not to sit at an occupied table is

$$NT_R = \frac{\theta}{N + \theta}$$

Note that this probability is not equivalent to the probability that the new customer chooses to sit at a *specific* unoccupied table $\tau \in \mathbb{N} \setminus \mathrm{T}$. Since there are an infinite number of unoccupied tables, the total sum of these probabilities, should they be defined this way, would be unbounded.

θ is a parameter that defines how attractive unoccupied tables are at this restaurant. As θ grows, a new customer becomes more likely to sit by himself at a new empty table.

3.3 Probabilistic Model

We seek to develop a probabilistic model of tasks that have a countably infinite solution space. As a first step, we focus on tasks that have exactly one correct answer (*e.g.* the examples mentioned in the introduction). We also assume that given the correct answer and problem difficulty, the worker's capability to produce the correct answer is independent of all the previous workers' responses. However, even when conditioning on v and d, wrong answers *are* dependent on previous workers' responses (because common mistakes are often

repeated). Finally, we assume that workers are not adversarial and do not collaborate with each other.

Dai *et al.*'s model [53, 54] is unable to solve this problem. When one tries to extend their model to tasks with an infinite number of possible solutions, several issues arise from the difficulty of assigning an infinite number of probabilities. For instance, since their model assigns a probability to each possible solution, the naïve extension of attempting to place a uniform distribution over the space of solutions is impossible.

Additionally, a good model must consider correlated errors [81]. For instance, consider a task that asks a worker to find the mobile phone number of a company's CEO. We can reasonably guess that the worker might Google the company name, and if one examined a histogram of worker responses, it would likely be correlated with the search results. A common error might be to return the company's main number rather than the CEO's mobile. Not all possible answers are equally likely, and a good model must address this fact.

The Chinese Restaurant Process [3] meets our desiderata. Let tables correspond to possible incorrect solutions to the task (Chinese restaurant); a new worker (diner) is more likely to return a common solution (sit at a table with more people) than a less common solution. We now formally define our extension of Dai *et al.*'s model to the case of unbounded possible answers.

We redefine the accuracy of a worker for a given task, $\xi(d, \gamma_w)$, to be:

$$\xi(d, \gamma_w) \quad = \quad (1 - d)^{\gamma_w}$$

As a worker's error parameter and/or the task's difficulty increases, the probability the worker produces the correct answer approaches 0. On the other hand, as the stated parameters decrease, ξ approaches 1, meaning the worker always produces the correct answer.

In addition to the difficulty d and the worker error γ_w, let $\theta \in \mathbb{R}^+$ denote the task's *bandwagon coefficient*. The parameter θ encodes the concept of the "tendency towards a common wrong answer." If θ is high, then workers who answer incorrectly will tend to provide new, unseen, incorrect answers, suggesting that the task does not have "common"

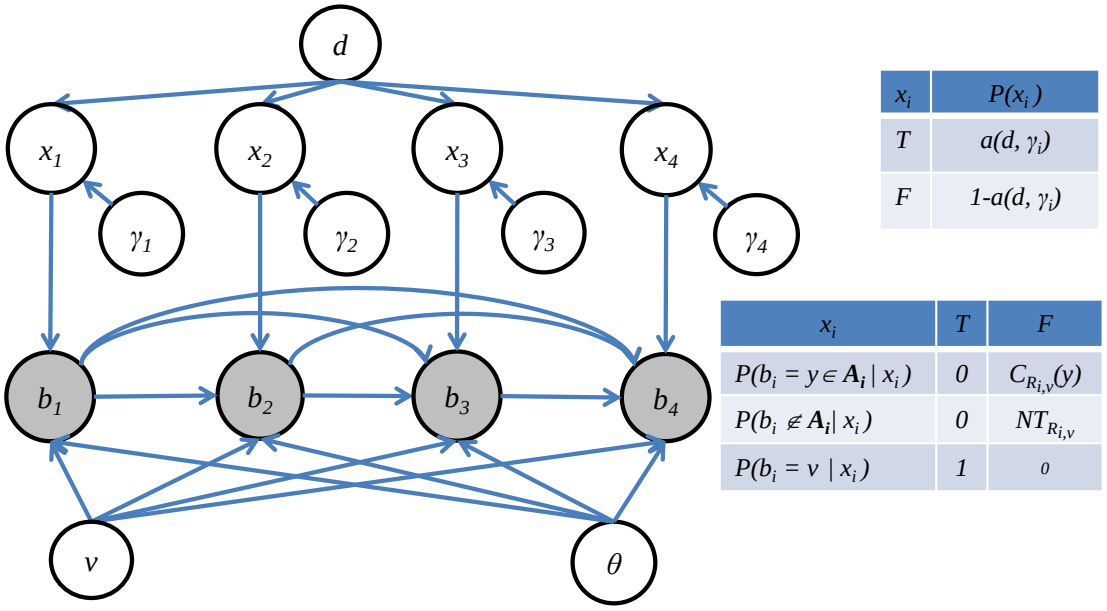

Figure 3.1: Whether or not worker i gets it right, x_i, depends on the worker's error parameter γ_i and the difficulty of the task d. Worker i's answer b_i depends on x_i, the question's true answer v, and all the previous workers' answers $b_1, \ldots, b_{i-1}$. R_i is a Chinese Restaurant Process defined by $\mathbf{b_i}$. This figure shows 4 workers. The b_i are the only observed variables.

wrong answers. Contrastingly, if θ is low, workers who answer incorrectly will tend toward the same incorrect answer, suggesting that the task lends itself to the same mistakes.

Figure 3.1 illustrates our generative model, which encodes a Bayes Net (see Section 2.2.2) for responses made by W workers on a given task. x_i is a binary random variable that indicates whether or not the i^{th} worker answers correctly. It is influenced by the correct answer v, the difficulty parameter d, and the error parameter γ_i. b_i, the answer that is provided by the i^{th} worker, is determined by x_i and all previous responses $b_1, \ldots, b_{i-1}$. Only the responses are observable variables.

Let $\mathbf{b_i} = \{b_1, \ldots, b_i\}$ be the multiset of answers that workers $w_1, \ldots, w_i$ provide. Let $\mathbf{a_i} = \{a_1, \ldots, a_k\}$ be the set of unique answers in $\mathbf{b_i}$. The probability that the $i + 1^{th}$ worker

produces the correct answer is simply the worker's accuracy for the given task:

$$\begin{aligned} P(x_{i+1} = T|d, v) &= \xi(d, \gamma_{i+1}), \\ P(x_{i+1} = F|d, v) &= 1 - \xi(d, \gamma_{i+1}). \end{aligned}$$

Then, the probability that the worker's response is correct is defined as

$$P(b_{i+1} = v|d, v, \mathbf{b_i}) = P(x_{i+1} = T|d, v).$$

To define the probability space of wrong answers we use the Chinese Restaurant Process. Let $f(a) = |\{b \in \mathbf{b_i}|b = a\}|$, and let $R_{i,v} = (\mathbf{a_i} \setminus \{v\}, f, \theta)$ be a Chinese Restaurant Process. Then, the probability that the worker returns a previously seen incorrect answer, $y \in \mathbf{a_i} \setminus \{v\}$ is

$$P(b_{i+1} = y|d, v, \mathbf{b_i}) = P(x_{i+1} = F|d, v) C_{R_{i,v}}(y).$$

Finally, the probability that the worker returns an unseen answer is

$$P(b_{i+1} = u|d, v, \mathbf{b_i}) = P(x_{i+1} = F|d, v) NT_{R_{i,v}},$$

where u represents whatever the worker returns as long as $u \notin \mathbf{a_i}$. We define the following notation to simplify and elucidate:

$$P(b_{i+1} \notin \mathbf{a_i}|d, v, \mathbf{b_i}) := P(b_{i+1} = u|d, v, \mathbf{b_i}).$$

The model cares only about whether it has seen a worker's answer before, not what it actually turns out to be.

3.3.1 Model Discussion

We now make several subtle and important observations. First, our model is dynamic in the following sense. As more workers provide answers, the probabilities that govern the generation of an incorrect answer change. In particular, the parameter θ becomes less and less significant as more and more workers provide answers. In other words, as i goes to

infinity, the probability that a new worker provides an unseen answer, $\theta/(\theta + i)$, goes to 0. As workers provide answers, the probability mass that used to dictate the generation of a new unseen answer is slowly shifted to that which determines the generation of seen answers. See Section 3.5.4 for a consequence of this behavior. Although we do not believe these model dynamics completely reflect the real-world accurately, we believe our model is a good first approximation with several desirable aspects. In fact, the model dynamics we just described are able to capture the intuition that as more and more answers arrive, we should expect to see fewer and fewer new answers.

Second, certain areas of parameter space cause our model to produce adversarial behavior. In other words, there are settings of d and θ for a task such that the probability a worker produces a particular incorrect answer is greater than the probability a worker provides the correct answer, on average. The following theorem makes this observation.

Theorem 3.1. *Suppose the difficulty, d, is fixed and all workers' γ are equal. Further suppose that we have seen an incorrect answer from a worker. Then, $\theta < \frac{1-2(1-d)^\gamma}{(1-d)^\gamma}$ if and only if the expected probability the next worker returns the first-seen incorrect answer is greater than the probability the next worker returns the correct answer. Stated more formally, if X_m is a random variable denoting the number of workers who have provided the first-seen incorrect answer after m workers have provided incorrect answers, then for all m, $\theta < \frac{1-2(1-d)^\gamma}{(1-d)^\gamma}$ if and only if*

$$(1 - (1 - d)^\gamma) \cdot E[\frac{X_m}{m + \theta}] > (1 - d)^\gamma.$$

Proof. Let I_j be a random variable indicating whether or not the jth worker that has provided an incorrect answer sits at the table representing the first-seen incorrect answer. By definition, $P(I_1 = 1) = 1$. Then, $X_m = I_1 + \ldots + I_m$.

By our generative model,

$$\begin{aligned} P(I_j = 1) &= \sum_{k=1}^{j-1} P(X_{j-1} = k) \frac{k}{j - 1 + \theta} \\ &= E[\frac{X_{j-1}}{j - 1 + \theta}]. \end{aligned}$$

Thus,

$$\begin{aligned} E[X_m] &= (1 + E[\frac{X_1}{1+\theta}] + \ldots + E[\frac{X_{m-1}}{m-1+\theta}]) \\ &= E[X_{m-1}] + E[\frac{X_{m-1}}{m-1+\theta}] \\ &= E[X_{m-1}] + \frac{1}{m-1+\theta} \cdot E[X_{m-1}] \\ &= (1 + \frac{1}{m-1+\theta})E[X_{m-1}] \\ &= \frac{m+\theta}{m-1+\theta}E[X_{m-1}] \end{aligned}$$

We see then, that $E[X_m] = \frac{m+\theta}{1+\theta}$. Therefore, we have $E[\frac{X_m}{m+\theta}] = \frac{1}{\theta+1}$ for all m. Therefore,

$$\begin{aligned} \theta &< \frac{1-2(1-d)^\gamma}{(1-d)^\gamma} \\ &\Updownarrow \\ \theta &< \frac{1-(1-d)^\gamma}{(1-d)^\gamma} - \frac{(1-d)^\gamma}{(1-d)^\gamma} \\ &\Updownarrow \\ \theta + 1 &< \frac{1-(1-d)^\gamma}{(1-d)^\gamma} \\ &\Updownarrow \\ \frac{(1-d)^\gamma}{1-(1-d)^\gamma} &< \frac{1}{\theta+1} \\ &\Updownarrow \\ \frac{(1-d)^\gamma}{1-(1-d)^\gamma} &< E[\frac{X_m}{m+\theta}] \\ &\Updownarrow \\ (1-d)^\gamma &< (1-(1-d)^\gamma) \cdot E[\frac{X_m}{m+\theta}]. \end{aligned}$$

□

We note that the theorem only considers the first-seen incorrect answer since the behavior of the Chinese Restaurant Process is such that the first-seen incorrect answer is generated with the highest expected probability. If the expected probability of generating the correct

answer exceeds that of the first-seen incorrect answer, the model will not produce adversarial behavior.

Finally, we observe that while the purpose of this chapter is to address open questions with infinite answer spaces, we note that Polya's Urn Scheme, the finite version of the Chinese Restaurant Process, applies equally well to finite answer spaces with many answer choices (multiple-choice tasks).

3.4 A Decision-Theoretic Agent

We now discuss the construction of our decision-theoretic controller, LazySusan. Our control problem is as follows. Given an open question as input, the goal is to infer the correct answer. At each time-step, an agent can choose one of two actions. It can either stop and submit the most likely answer, or it can create another job and receive another response to the task from another crowdsourced worker. The question is: How do we determine the agent's policy?

To solve this problem, first we define the *world state* of LazySusan to be the pair (v, d), where $v \in \mathbb{N}$ is the correct answer of the task and d is the difficulty of the task. The space of world states is infinitely large, and LazySusan cannot directly observe the world state, so it has an *agent state* $\mathbf{S}$, which at time i, is the set of tuples, $\mathbf{S} = \{(v, d) | v \in \mathbf{a_i} \cup \{\perp\} \wedge d \in [0, 1]\}$, where $\perp$ represents the case when the true answer has not been seen by the agent so far. In order to keep track of what it believes to be the correct answer v, it maintains a *belief*, which is a probability distribution over $\mathbf{S}$.

The agent state allows us to fold an infinite number of probabilities into one, since to compute the belief, one only needs to calculate $P(v = \perp, d|\mathbf{b_i})$, the probability that no workers have provided the correct answer yet given $\mathbf{b_i}$, instead of $P(v = u, d|\mathbf{b_i})$ for all possible unseen answers $u \in \mathbb{N} \setminus \mathbf{a_i}$.

Symbol	Meaning
$\mathbf{a_i}$	The set of unique responses made by workers $1, \ldots, i$.
$\mathbf{b_i}$	The multiset of responses made by workers $1, \ldots, i$ ($\{b_1, \ldots, b_i\}$)
b_i	Worker i's response to the task
C_C	The value of a correct answer
$C_{R_{i,v}}(y)$	The probability of a worker producing incorrect answer y in restaurant $R_{i,v}$.
C_W	The value of an incorrect answer
d	Difficulty of task
γ_i	Worker i's error parameter
θ	A task's bandwagon coefficient; Chinese Restaurant Process parameter
k	The number of unique worker responses ($\lvert\mathbf{a_i}\rvert$)
i	Number of ballots received so far
$Q(\mathbf{b_i}, \texttt{action})$	The utility of taking `action` with belief $\mathbf{b_i}$
$NT_{R_{i,v}}$	The probability of a worker producing an unseen answer in restaurant $R_{i,v}$.
$R_{i,v}$	An instance of the Chinese Restaurant Process instantiated using $\mathbf{b_i}$ and correct answer v.
$U(\mathbf{b_i})$	The utility of a belief derived from $\mathbf{b_i}$.
$V(a)$	The value of an answer a
v	Correct answer of task
x_i	Did worker i answer the task correctly

Table 3.1: Summary of notation used in this chapter

3.4.1 Belief Update

We now first describe specifically how LazySusan updates its posterior belief $P(v, d|\mathbf{b_i}; i, k)$ after it receives its i^{th} answer b_i. Here, $k = |\mathbf{a_i}|$ is the number of unique responses it has received. By Bayes' Rule, we have

$$P(v, d|\mathbf{b_i}; i, k) \propto P(\mathbf{b_i}|v, d; i, k)P(v, d; i, k).$$

The likelihood of the worker responses $P(\mathbf{b_i}|v, d; i, k)$ is easily calculated using our generative model:

$$P(\mathbf{b_i}|v, d; i, k) = \prod_{j=1}^{i} P(b_j|v, d, \mathbf{b_{j-1}}).$$

Instead of starting from arbitrary priors on the correct answer and difficulty, we can model it using i and k:

$$P(v, d; i, k) = P(v|d; i, k)P(d; i, k).$$

The prior we must compute describes the joint probability of the correct answer and difficulty given i responses and k distinct responses. Notice that for all $a \in \mathbf{a_i}$, we do not know $P(v = a|d; i, k)$. However, they must be all the same, because knowing the difficulty of the task gives us no information about the correct answer. Therefore, we must only determine the probability the correct answer has yet to be seen given d, i, and k. We propose the following model:

$$P(v = \perp |d; i, k) = d^i.$$

This definition is reasonable since intuitively, as the difficulty of the task increases the more likely workers have not yet provided a correct answer. On the other hand, as the number of observations increases, we become more certain that the correct answer is in $\mathbf{a_i}$.

Finally, we model $P(d; i, k)$. Suppose for the moment that workers tend to produce answers that LazySusan has seen before (θ is low). Intuitively, as k approaches i, the

difficulty should grow, because the agent is seeing a lot of different answers when it shouldn't, and as k approaches 1, the difficulty should become smaller, because everyone is agreeing on the same answer.

We choose to model $P(d; i, k) \sim Beta(\alpha, \beta)$ and define $\alpha \geq 1$ and $\beta \geq 1$ as

$$\begin{aligned} \alpha &= \left((i-1)\frac{k}{i} + 1\right)^{\frac{1}{\theta}} \\ \beta &= \left((1-i)\frac{k}{i} + i\right)^{\theta} \end{aligned}$$

First note that the bulk of a Beta distribution's density moves toward 1 as α increases relative to β, and toward 0 as β increases relative to α. Thus, as β increases, difficulty decreases, and as α increases, difficulty increases. Consider the case when $\theta = 1$. As k approaches i, α approaches i and β approaches 1, causing LazySusan to believe the difficulty is likely to be high, and as k approaches 1, LazySusan believes the difficulty is likely to be low. This behavior is exactly what we desire.

Now we consider the effect of θ. Fix i and k. As θ grows, β increases and α decreases. Therefore, for a *fixed* multiset of observations, as people become more likely to provide unseen answers, the probability that the difficulty is low becomes greater. In other words, LazySusan needs to see a greater variety of answers to believe that the problem is difficult if θ is high.

Let $\theta_1 > \theta_2$ and consider the following scenarios: Suppose k is close to i, so LazySusan believes, before factoring in θ, that the task is difficult. If $\theta = \theta_2$, LazySusan believes with more certainty that the task is difficult than if $\theta = \theta_1$. This behavior makes sense because LazySusan should expect a small k if θ is small. If $\theta = \theta_2$, LazySusan should see a smaller k than if $\theta = \theta_1$. If it sees a k that is larger than it expects, it correctly deduces that more people are getting the question wrong, and concludes the task is more difficult. Similarly, if k is close to 1, and $\theta = \theta_1$, then LazySusan believes with more certainty that the task is easy than if $\theta = \theta_2$, since even though workers tend to produce more random answers, k is small.

3.4.2 Utility Estimation

To determine what actions to take, LazySusan needs to estimate the utility of each action. The first step is to assign utilities to its beliefs. Since $\mathbf{b_i}$ solely determines its belief at time i, we denote $U(\mathbf{b_i})$ to be utility of its current belief. Next, LazySusan computes the utilities of its two possible actions. Let $Q(\mathbf{b_i}, \texttt{submit})$ denote the utility of submitting the most likely answer given its current state, and let $Q(\mathbf{b_i}, \texttt{request})$ denote the utility of requesting another worker to complete the task and then performing optimally. Then

$$\begin{aligned} U(\mathbf{b_i}) &= \max\{Q(\mathbf{b_i}, \texttt{submit}), Q(\mathbf{b_i}, \texttt{request})\}, \\ Q(\mathbf{b_i}, \texttt{submit}) &= \sum_{a \in \mathbf{a_i}} V(a) \int_d P(v = a, d | \mathbf{b_i}; i, k) \mathrm{d}d, \\ Q(\mathbf{b_i}, \texttt{request}) &= c + \sum_{a \in \mathbf{a_i}} P(b_{i+1} = a | \mathbf{b_i}) U(\mathbf{b_{i+1}}) \\ &\quad + P(b_{i+1} \notin \mathbf{a_i} | \mathbf{b_i}) U(\mathbf{b_{i+1}}), \end{aligned}$$

where c is the cost of creating another job, $P(b_{i+1}|\mathbf{b_i}) =$

$$\sum_{a \in \mathbf{a_i}} \int_d P(b_{i+1} | v = a, d, \mathbf{b_i}) P(v = a, d | \mathbf{b_i}) \mathrm{d}d,$$

and $V(a)$ is the utility of submitting answer a. To define $V(a)$, let C_C be the utility of a correct answer, C_W be the utility of an incorrect answer, and the predicted correct answer be $a^* = \text{argmax}_{a \in \mathbf{a_i}} \int_d P(v = a, d | \mathbf{b_i}; i, k) \mathrm{d}d$. Then,

$$V(a) = \begin{cases} C_C & \text{if } a = a^* \\ C_W & \text{otherwise,} \end{cases}$$

where C_C and C_W are are provided by the user to manage tradeoffs between accuracy and cost.

3.4.3 Worker Tracking

After submitting an answer, LazySusan updates its records about all the workers who participated in the task using a^*. We follow the approach of the last chapter and Dai

et al. [52], and use the following update rules: 1) $\gamma_w \leftarrow \gamma_w - d\sigma$ should the worker answer correctly, and 2) $\gamma_w \leftarrow \gamma_w + (1-d)\sigma$, should the worker answer incorrectly, where σ is a decreasing learning rate. Any worker that LazySusan has not seen previously begins with some starting $\overline{\gamma}$.

3.4.4 Decision Making

We note that the agent's state space continues to grow without bound as new answers arrive from crowdsourced workers. This poses a challenge since existing POMDP algorithms do not handle infinite-horizon problems in dynamic state spaces where there is no a priori bound on the number of states. Indeed, the efficient solution of such problems is an exciting problem for future research. As a first step, LazySusan selects its actions at each time step by computing an l-step lookahead by estimating the utility of each possible sequence of l actions. If the l^{th} action is to request another response, then it will cut off the computation by assuming that it submits an answer on the $l+1^{th}$ action.

In many crowdsourcing platforms, such as Mechanical Turk, we cannot preselect the workers to answer a job. However, in order to conduct a lookahead search, we need to specify future workers' parameters for our generative model. To simplify the computation, we assume that every future worker has $\gamma = \overline{\gamma}$.

3.4.5 Joint Learning and Inference

We now describe an EM algorithm that can be used as an alternative to the working-tracking scheme from above. Given a set of worker responses from a set of tasks, $\mathbf{b}$, EM jointly learns maximum-likelihood estimates of $\boldsymbol{\gamma}, \mathbf{d}$, and $\boldsymbol{\theta}$, while inferring the correct answers $\mathbf{v}$. Thus, in this approach, after LazySusan submits an answer to a task, it can recompute all model parameters before continuing with the next task.

We treat the variables $\mathbf{d}, \boldsymbol{\gamma}$, and $\boldsymbol{\theta}$ as parameters. Let old parameters be denoted with a dot (*e.g.* $\dot{\mathbf{d}}$). In the E-step, we keep old parameters fixed to compute the posterior probabil-

ities of the hidden true answers, $p(v_t|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}})$, for each task t:

$$\begin{aligned} P(v_t|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}; i_t, k_t) &= P(v_t|\mathbf{b_t}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}; i_t, k_t) \\ &\propto P(\mathbf{b_t}|v_t, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}; i_t, k_t) P(v_t|\dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}; i_t, k_t) \\ &= P(\mathbf{b_t}|v_t, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) P(v_t|\dot{\mathbf{d}}; i_t, k_t) \\ &= P(v_t; \dot{\mathbf{d}}, i_t, k_t) \prod_j P(b_{t,j}|v_t, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}_{\boldsymbol{j}}, \dot{\boldsymbol{\theta}}). \end{aligned}$$

The M-step uses these posterior probabilities to maximize the standard expected complete log-likelihood L over $\mathbf{d}, \boldsymbol{\gamma}$, and $\boldsymbol{\theta}$,

$$L(\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta}) = E[\ln p(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})],$$

where the expectation is taken over $\mathbf{v}$ given the old values of $\dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{d}}, \dot{\boldsymbol{\theta}}$:

$$
\begin{aligned}
& E[\ln P(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})] \\
= \; & E[\ln P(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta}) + \ln P(\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})] \\
= \; & E[\ln P(\mathbf{v}|\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta}) P(\mathbf{b}|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})] + \ln P(\mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta}) \\
= \; & E[\ln P(\mathbf{v}) P(\mathbf{b}|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})] + \ln P(\mathbf{d}) P(\boldsymbol{\gamma}|\mathbf{d}) P(\boldsymbol{\theta}|\boldsymbol{\gamma}, \mathbf{d}) \\
= \; & E[\ln \prod_t P(v_t) P(\mathbf{b}_t|\mathbf{v}, \mathbf{d}, \boldsymbol{\gamma}, \boldsymbol{\theta})] + \ln P(\mathbf{d}) P(\boldsymbol{\gamma}) P(\boldsymbol{\theta}) \\
= \; & E[\ln \prod_t P(v_t) \prod_{ti} P(b_{t,i}|v_t, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})] + \ln P(\mathbf{d}) P(\boldsymbol{\gamma}) P(\boldsymbol{\theta}) \\
= \; & E[\sum_t \ln P(v_t) + \sum_{ti} \ln P(b_{t,i}|v_t, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})] + \ln P(\mathbf{d}) P(\boldsymbol{\gamma}) P(\boldsymbol{\theta}) \\
= \; & \sum_t E[\ln P(v_t)] + \sum_{ti} E[\ln P(b_{t,i}|v_t, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})] + \ln P(\mathbf{d}) P(\boldsymbol{\gamma}) P(\boldsymbol{\theta}) \\
= \; & \sum_t E[\ln P(v_t)] + \\
& \sum_{ti} \sum_{a \in \mathbf{a_i}} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \ln P(b_{t,i}|v_t = a, d_t, \gamma_i \theta_t, b_{t,1}, \ldots, b_{t,i-1}) + \\
& \ln \prod_t P(d_t) P(\theta_t) \prod_i P(\gamma_i) \\
= \; & \sum_t \sum_{a \in \mathbf{a_i}} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \ln P(v_t = a; i_t, k_t) + \\
& \sum_{ti} \sum_{a \in \mathbf{a_i}} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1}) + \\
& \sum_t \ln P(d_t) + \sum_t \ln P(\theta_t) + \sum_i \ln P(\gamma_i).
\end{aligned}
$$

In order to find the parameters that maximize L, we can differentiate with respect to the parameters and apply iterative minimization algorithms like conjugate gradient methods (*e.g.* [74].) Letting $\Xi = |\{b \in \{b_{t,1}, \ldots, b_{t,i-1}\} : b = b_{t,i}\}|$ and $\xi = (1 - d_t)^{\gamma_i}$ and differentiating with respect to $\boldsymbol{d}$, we have that

$$
\begin{aligned}
\frac{\partial L}{\partial d_t} \; = \; & \sum_i \sum_{a \in \mathbf{a_i}} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial d_t} \\
& + \frac{1}{P(d_t)} \frac{dP(d_t)}{dd_t},
\end{aligned}
$$

where if $b_{t,i} = v_t$

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial d_t} &= \frac{\partial \ln \xi}{\partial d_t} \\ &= \frac{-\gamma_i (1-d_t)^{\gamma_i - 1}}{\xi},\end{aligned}$$

and otherwise,

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial d_t} &= \frac{\partial \ln (1-\xi)\frac{\Xi}{i-1+\theta_t}}{\partial d_t} \\ &= \frac{\gamma_i (1-d_t)^{\gamma_i - 1}}{1-\xi}.\end{aligned}$$

Differentiating with respect to $\boldsymbol{\gamma}$, we have that:

$$\begin{aligned}\frac{\partial L}{\partial \gamma_i} &= \sum_t \sum_{a \in \mathbf{a_i}} P(v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \gamma_i} \\ &\quad + \frac{1}{P(\gamma_i)} \frac{dP(\gamma_i)}{d\gamma_i},\end{aligned}$$

where if $b_{t,i} = v_t$

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \gamma_i} &= \frac{\partial \ln \xi}{\partial \gamma_i} \\ &= \frac{(1-d_t)^{\gamma_i} \ln(1-d_t)}{\xi},\end{aligned}$$

and otherwise,

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \gamma_i} &= \frac{\partial \ln (1-\xi)\frac{\Xi}{i-1+\theta_t}}{\partial \gamma_i} \\ &= \frac{-(1-d_t)^{\gamma_i} \ln(1-d_t)}{1-\xi}.\end{aligned}$$

Finally, differentiating with respect to $\boldsymbol{\theta}$, we have that:

$$\begin{aligned}\frac{\partial L}{\partial \theta_t} &= \sum_i \sum_{a \in \mathbf{a_i}} P(x_{t,i} = 0, v_t = a|\mathbf{b}, \dot{\mathbf{d}}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{\theta}}) \frac{\partial \ln P(b_{t,i}|x_{t,i} = 0, v_t = a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \theta_t} \\ &\quad + \frac{1}{P(\theta_t)} \frac{dP(\theta_t)}{d\theta_t},\end{aligned}$$

where if $b_{t,i} \notin \{b_{t,1}, \ldots, b_{t,i-1}\}$

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|x_{t,1}=0, v_t=a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \theta_t} &= \frac{\partial \ln(1-\xi)\frac{\theta_t}{i-1+\theta_t}}{\partial \theta_t} \\ &= \frac{\theta_t + i - 1}{\theta_t}\frac{i-1}{(\theta_t+i-1)^2},\end{aligned}$$

and otherwise, if $b_{t_i} \neq v_t$,

$$\begin{aligned}\frac{\partial \ln P(b_{t,i}|x_{t,i}=0, v_t=a, d_t, \gamma_i, \theta_t, b_{t,1}, \ldots, b_{t,i-1})}{\partial \theta_t} &= \frac{\partial \ln(1-\xi)\frac{\Xi}{i-1+\theta_t}}{\partial \gamma_i} \\ &= \frac{\theta_t + i - 1}{\Xi}\frac{\Xi}{(\theta_t+i-1)^2}.\end{aligned}$$

3.5 Experiments

This section addresses the following questions:

1. How deeply should the lookahead search traverse?

2. How robust is LazySusan on different classes of problems?

3. How well does LazySusan work in practice?

4. How well does our EM algorithm work in practice?

To answer the first question, we compare LazySusan at different settings of lookahead depth. Then, to answer the second question, we test the robustness of LazySusan by applying it to various kinds of problems in simulation. Next, to answer the third question, we compare LazySusan to an agent that uses majority-voting with tasks that test the workers of Amazon Mechanical Turk on their SAT Math skills. Finally, to answer the fourth question, we compare our EM algorithm to a majority-voting strategy on a visualization task.

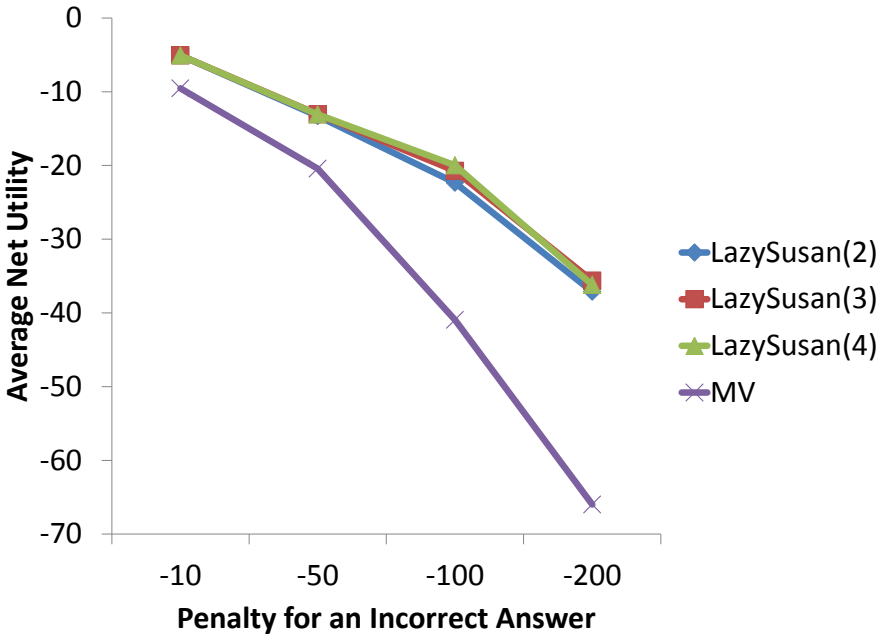

Figure 3.2: In simulation, our lookahead search does well at all depths.

3.5.1 Implementation

Since numerical integration can be challenging, we discretize difficulty into nine equally-sized buckets with centers of $0.05, 0.15, \ldots, 0.85$, and 0.95.

For the purposes of simulation only, each response that LazySusan receives also contains perfect information about the respective worker's γ. Thus, LazySusan can update its belief state with no noise, which is the best-case scenario.

In all cases, we set the learning rate $\sigma = \frac{1}{m_w+1}$, where m_w is the number of questions a worker w has answered so far. We also set the value of a correct answer to be $C_C = 0$. Finally, we set $\overline{\gamma} = 1$, and the bandwagon coefficient $\theta = 1$ for all tasks.

3.5.2 Best Lookahead Depth

We first determine the best lookahead depth among 2, 3, and 4 with simulated experiments. We evaluate our agents using several different settings of the value of an incorrect answer: $C_W \in \{-10, -50, -100\}$. Each difficulty setting is used 10 times, for a total 90 simulations per utility setting. (9 difficulty settings $\times$ 10 = 90). We set the cost of requesting a job, c, from a (simulated) worker to -1. Our simulated environment draws workers' $\gamma \in (0, 2)$ uniformly.

Figure 3.2 shows the results of our simulation. LazySusan(l) denotes a lookahead depth of l. We also examine an agent that uses a majority-of-7-votes strategy, MV. Expectedly, as the utility of an incorrect answer decreases, the average net utility achieved by each agent drops. We find that for all settings of lookahead depth, LazySusan dramatically outperforms MV. We also see that LazySusan(3) and LazySusan(4) both achieve small, but sure gains over LazySusan(2). However, LazySusan(3) and LazySusan(4) seem to do about the same. Since LazySusan(4) runs more slowly, we decide to use LazySusan(3) in all future experiments, and refer to it as LazySusan.

3.5.3 Noisy Workers

We examine the effect of poor workers. We compare two simulation environments. In the first, we draw workers' $\gamma \in (0, 1)$ uniformly and in the second, we draw workers' $\gamma \in (0, 2)$ uniformly. Thus, the first environment has workers that are much more competent than those in the second. All other parameters remain as before. Each difficulty setting is simulated 100 times, for a total of 900 simulations per environment. The results of this experiment are shown in Table 3.2. When the workers are competent, LazySusan makes small gains over MV, reducing the error by 34.3%. However, when there exist noisier workers in the pool, LazySusan more decisively outperforms MV, reducing the error by 48.6%. In both cases, LazySusan spends less money than MV, netting average net utility gains of 55% and 75.9%.

	$\gamma \in (0,1)$		$\gamma \in (0,2)$	
	LazySusan	MV	LazySusan	MV
Avg Accuracy (%)	88.7	82.8	83.3	67.5
Avg Cost	3.472	5.7	4.946	6.01
Avg Net Utility	-14.772	-22.9	-21.646	-38.51

Table 3.2: Comparison of average accuracies, costs, and net utilities of LazySusan and MV when workers either have $\gamma \in (0,1)$ or $\gamma \in (0,2)$.

3.5.4 Tasks with Correlated Errors

Next, we investigate the ability of LazySusan to deal with varying θ and d in the worst case, when all the workers are equally skilled. Recall that a high θ means that workers who answer incorrectly will tend to produce previously unseen answers. We consider the following variations of tasks: 1) Low difficulty, 2) High difficulty, high θ, and 3) High difficulty, low θ. In the first two cases, we see expected behavior. LazySusan is able to use its model to infer correct answers.

However, in the third case, we see some very interesting behavior. Since the difficulty is high, workers more often than not produce the wrong answer. Additionally, since θ is low, they also tend to produce the same wrong answer, making it look like the correct answer. If the ratio is large enough, we find that LazySusan is unable to infer the correct answer, because of the unfortunate ambiguity between high difficulty, low θ problems and low difficulty problems. In fact, as LazySusan observes more responses, it becomes more convinced that the common wrong answer is the right answer, because of the model dynamics we mention earlier (Section 3.3.1). This problem only arises, however, if the model produces adversarial θ, and we see in practice that workers on Mechanical Turk generally do not exhibit such behavior.

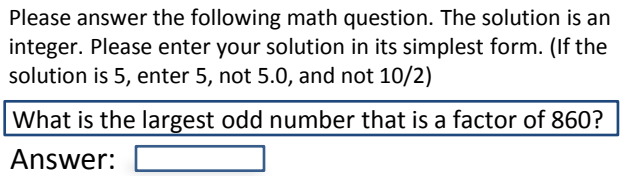

Figure 3.3: An example of an SAT Math Question task posed to workers for live experiments on Mechanical Turk.

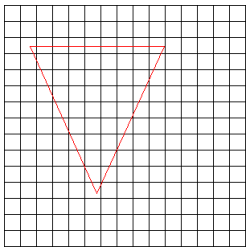

Figure 3.4: An example of a "triangle" task posed to workers for live experiments on Mechanical Turk. The area of this triangle, rounded down, is 38.

3.5.5 Experiments on Mechanical Turk

Next, we compare LazySusan to an agent using majority-vote (MV) using real responses generated by Mechanical Turk workers. We test these agents with 134 math questions with levels of difficulty comparable to those found on the SAT Math section. Figure 3.3 is an example of one such task and the user interface we provided to workers. We set the utility for an incorrect answer, C_W, to be -100, because with this utility setting, LazySusan requests about 7 jobs on average for each task, and a simple binary search showed this number to be satisfactorily optimal for MV. We find that the workers on Mechanical Turk are surprisingly capable at solving math problems. As Table 3.3 shows, LazySusan almost completely eliminates the error made by MV. Since the two agents cost about the same, LazySusan achieves a higher net utility, which we find to be statistically significant using a Student's t-test ($p < 0.0002$).

	LazySusan	MV
Avg Accuracy (%)	99.25	95.52
Avg Cost	5.17	5.46
Avg Net Utility	-5.92	-9.94

Table 3.3: Comparisons of average accuracies, costs, and net utilities of LazySusan and MV when run on Mechanical Turk.

We examine the sequence of actions LazySusan made to infer the correct answer to the task in Figure 3.3. In total, it requested answers from 14 workers, and received the following responses: 215, 43, 43, 43, 5, 215, 43, 3, 55, 43, 215, 215, 215, 215. Since MV takes the majority of 7 votes, it infers the answer incorrectly to be 43. LazySusan on the other hand, uses its knowledge of correlated answers as well as its knowledge from previous tasks that the first three workers who responded with 43 were all relatively poor workers compared to the first two workers who claimed the answer is 215. So even though a clear majority of workers preferred 43, LazySusan was not confident about the answer. While it cost twice as much as MV, the cost was a worthy sacrifice with respect to the utility setting.

Finally, we compare our EM algorithm to MV, using real responses generated by Mechanical Turk workers. We develop a "triangle" task (Figure 3.4) that presents workers with a triangle drawn on a grid, and asks them to find the area of the triangle, rounded down. We posted 200 of these tasks and solicited 5 responses for each. These tasks are difficult since many of the responses are off by 1. Our EM algorithm achieves an accuracy of 65.5% while MV achieves an accuracy of 54.1%.

3.6 Related Work

Modeling repeated labeling in the face of noisy workers when the label is assumed to be drawn from a known *finite* set has received significant attention. Romney *et al.* [179] are one of the first to incorporate a worker accuracy model to improve label quality. Sheng *et al.* [198]

explore when it is necessary to get another label for the purpose of machine learning. Raykar *et al.* [177] propose a model in which the parameters for worker accuracy depend on the true answer. Whitehill *et al.* [242] and Dai *et al.* [52] address the concern that worker labels should not be modeled as independent of each other unless given problem difficulty. Welinder *et al.* [240] design a multidimensional model for workers that takes into account competence, expertise, and annotator bias. Kamar *et al.* [104] extracts features from the task at hand and use Bayesian Structure Learning to learn the worker response model. Parameswaran *et al.* [164] conduct a policy search to find an optimal dynamic control policy with respect to constraints like cost or accuracy. Karger *et al.* [109] develop an algorithm based on low-rank matrix approximation to assign tasks to workers and infer correct answers, and analytically prove the optimality of their algorithm at minimizing a budget given a reliability constraint. Snow *et al.* [206] show that for labeling tasks, a small number of Mechanical Turk workers can achieve an accuracy comparable to that of an expert labeler. None of these works consider tasks that have an infinite number of possible solutions.

For more complex tasks that have an infinite number of possible answers, innovative workflows have been designed, for example, an iterative improvement workflow for creating complex artifacts [141], find-fix-verify for an intelligent editor [24], and others for counting calories on a food plate [163].

An AI agent makes an efficient controller for these crowdsourced workflows. Dai *et al.* [52, 54] create a POMDP-based agent to control an iterative improvement workflow. Shahaf and Horvitz [192] develop a planning-based task allocator to assign subtasks to specific humans or computers with known abilities.Weld *et al.* [239] discuss a broad vision for the use of AI techniques in crowdsourcing that includes workflow optimization, interface optimization, workflow selection and intelligent control for general crowdsourced workflows.

3.7 Conclusion

This chapter introduces LazySusan, an agent that takes a decision-theoretic approach to inferring the correct answer of a task that can have a countably infinite number of possible

answers. We extend the probabilistic model of [52] using the Chinese Restaurant Process and use l-step lookahead to approximate the optimal number of crowdsourcing jobs to submit. We also design an EM algorithm to jointly learn the parameters of our model while inferring the correct answers to multiple tasks at a time. Live experiments on Mechanical Turk demonstrate the effectiveness of LazySusan. At comparable costs, it yields an 83.2% error reduction compared to majority vote, which is the current state-of-the-art technique for aggregating responses for tasks of this nature. Live experiments also show that that our EM algorithm outperforms majority-voting on "triangle" tasks.

Part II

INTELLIGENTLY GATHERING DATA FROM THE CROWD FOR TRAINING

Chapter 4

TO RE(LABEL), OR NOT TO RE(LABEL)

4.1 Introduction

In the first part of the dissertation, we explored two methods for obtaining clean testing data. In particular, we developed models for dynamically using multiple workflows and we presented a label aggregation method that works for tasks that require free responses and cannot be presented as multiple-choice questions.

In this second part of the dissertation, we switch our focus from testing data to training data. Because machine learning algorithms are robust to noise, training data does not necessarily have to be clean, in contrast with testing data. Instead, training data must balance between between noise, quantity, and diversity. Because of the many differences between our study of the requirements of testing data and our study of the requirements of training data, in this part of the dissertation we ignore and redefine the mathematical notation we develop in the first part.

In the next two chapters, we closely examine the tradeoff between noise and size. Our goal is to address the following fundamental question: "Is it better to spend an incremental dollar asking a worker to relabel an existing example or to label a new example?" In contrast to the first part of the dissertation, we add the assumption that all tasks are the same and all workers are independently and identically Bernoulli-distributed, allowing us to model noise using a single parameter. We maintain the assumptions that tasks have a single objective true answer, and that workers are not adversarial and do not collaborate.

We first set out to answer this noise versus size question by considering a subset of real-world datasets from the UCI Machine Learning Repository [13]. We simulate a noisy annotation process with 55% accurate workers, a fixed budget and simple, deterministic vote

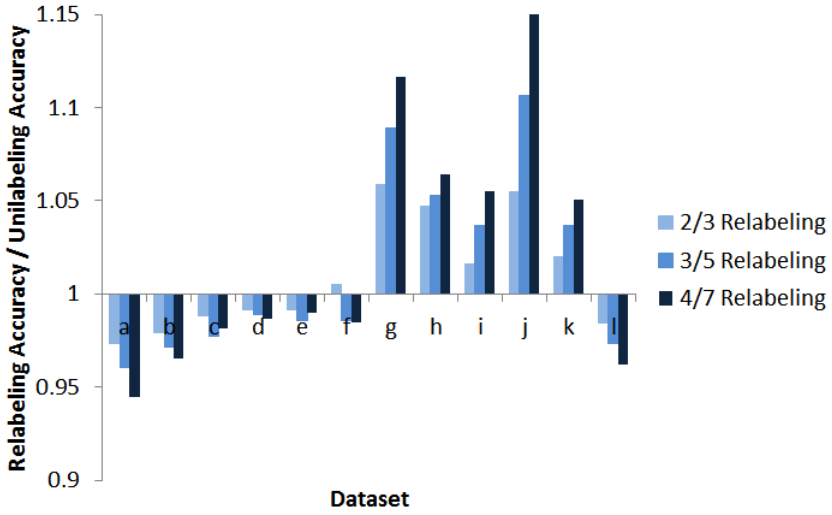

Figure 4.1: Relabeling training data improved learned-classifier accuracy on 5 out of 12 real-world domains (see Table 4.1) when workers were highly noisy (55% accurate). But why these domains and not the others?

aggregation schemes. Specifically, we consider an optimized majority voting scheme, which we call *j/k relabeling*, where we ask up to k workers to label an example, stopping as soon as $j = \lceil k/2 \rceil$ identical responses are received. *Unilabeling* refers to the strategy that only collects one label per example, i.e., 1/1 relabeling. Figure 4.1 shows the ratio of the accuracy achieved by classifiers trained using relabeling (denoted as *relabeling accuracy*) to the accuracy achieved by classifiers trained using unilabeling (denoted as *unilabeling accuracy*). The results are inconclusive and puzzling. Relabeling helps in 5 of 12 domains, but it is unclear when relabeling would be useful and what level of redundancy would lead to the best results.

In this chapter, we first try to understand the properties that govern whether a classifier will have higher accuracy when trained on m noisy examples or a smaller set of examples (say $m/3$) with more accurate labels. In particular, we look at three characteristics of learning problems that might affect the relative relabeling versus unilabeling performances. These three dimensions include (1) the inductive bias of the learning algorithm, (2) the accuracy of workers, and (3) the budget. We study the effect of each of these dimensions on relabeling

power and find general principles that will help us determine whether to unilabel or relabel for a given learning problem.

4.2 Problem Setting

We focus on binary classification problems with symmetric loss functions. We assume that workers have uniform (but unknown) error rates. Finally, we first focus on passive learning, where the next data point is randomly sampled from an unlabeled dataset, as opposed to actively picked based on current classifier performance. In the next chapter, we will consider the consequences of switching from passive learning to active learning.

Let $\mathcal{X}$ denote the space of examples and D, its distribution. We consider binary classification problems of the following form. Given some hypothesis class, $\mathcal{C}$, which contains a set of mappings from $\mathcal{X}$ to $\{0,1\}$, and a budget b, the goal is to learn some true *concept* $h^* : \mathcal{X} \mapsto \{0,1\}$ by outputting the hypothesis $h \in \mathcal{C}$ which minimizes the expected error $\epsilon = P_{x\sim D} h^*(x) \neq h(x)$. Asking a worker to label an example incurs a fixed unit cost of 1. We assume that each worker exhibits the same (but unknown) accuracy $p \in (0.5, 1]$, an assumption known as the *classification noise model* [8], and we assume worker errors are independent.

Let $\zeta_k : \{0,1\}^k \mapsto \{0,1\}$ denote the *aggregation function* that produces a single aggregate label given k multiple labels. The aggregation function represents how we consolidate the multiple labels we receive from workers. Since all workers are equally accurate in our model, majority voting, a common aggregation function that takes k votes and outputs the class that received greater than $k/2$ votes, is an effective strategy. In all our experiments we use a simple optimization of majority vote, j/k relabeling, and stop requesting votes as soon as a class obtains $j = \lceil k/2 \rceil$ votes.

We also define $\eta_{\zeta_k} : [0,1] \to [0,1]$ as the function that, given the number of labels k that will be aggregated by ζ_k and the accuracy p of those labels, calculates the probability that the answer returned by the aggregation function ζ_k will be correct. In other words, it outputs the *aggregate accuracy*, the probability that the aggregate label will be equal to the

true label ([198, 92] call this probability *integrated quality*). As we will see, η is an important function that characterizes the power of relabeling.

In order to train the best classifier possible, an intermediate goal, and our goal in this chapter, is to determine the scenarios under which relabeling is better than unilabeling, assigning a single worker to annotate each example. In other words, given the dataset and classifier, we would like to determine whether or not examples should be relabeled (and with what redundancy) in order to maximize the classifier's accuracy.

4.3 The Effect of Inductive Bias

We first consider how the inductive bias of a classifier affects the relative performance of relabeling and unilabeling. We primarily use two tools to address the question: a theoretical analysis of error bounds using PAC-learnability and a series of empirical experiments on simulated data. However, we first provide some intuition.

Recall that a classifier's inductive bias characterizes the range of hypotheses that it will consider. A weakly regularized learner willing to consider a vast number of hypotheses that can be represented with an expressive hypothesis language, e.g. decision tree induction, is said to have a *weak inductive bias*. In contrast, logistic regression, which learns a linear model, has a stronger inductive bias. Why might relabeling effectiveness depend on the strength of a classifier's inductive bias?

As classifier bias gets weaker, its ability to fit training data patterns increases and the likelihood that it overfits increases. Consider the effect of noise on overfitting. As the noise in training labels increases, overfitting starts to hurt more, because the classifier not only overfits data, but also overfits *the wrong data*. Therefore, we predict that with weaker inductive bias, relabeling will become more necessary to prevent large overfitting errors.

As an illustration, suppose we want to classify whether or not a person is a "senior citizen," based on his/her age. Let the instance space $\mathcal{X}$ be people between the ages of 0 and 100 and the distribution D uniform. Let us suppose the target concept is the simple threshold that everyone older than 65 is a senior citizen. The hypothesis class $\mathcal{H}_1$ that consists

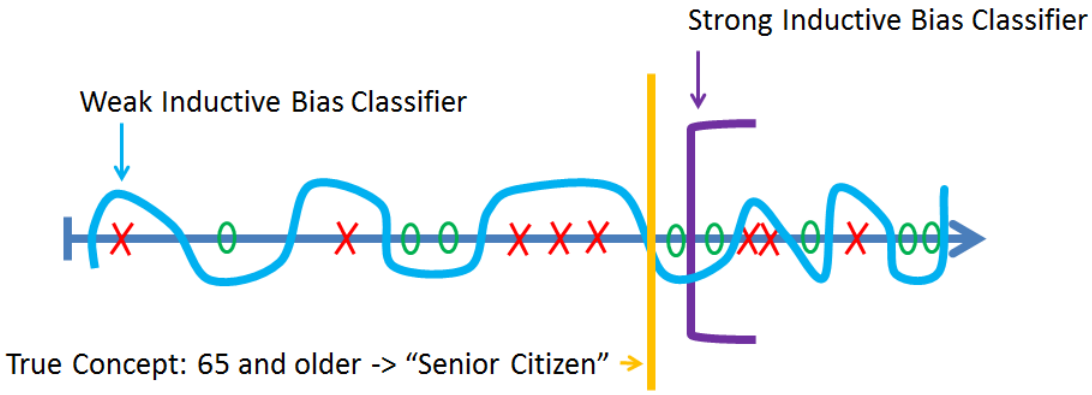

Figure 4.2: An "O" represents an example labeled as a "senior citizen" and an "X" designates an example labeled as "not a senior citizen." The target concept classifies all people older than 65 as senior citizens.

of all thresholds on $[0, \infty]$ has strong inductive bias. This hypothesis class is quite robust to noise, as shown in Figure 4.2. As long as the labeling accuracy is above 50%, a learning algorithm using $\mathcal{H}_1$ will probably get the threshold approximately correct, so relabeling is not really necessary. Furthermore, spending one's budget on additional examples increases the chance of getting examples that are close to the 65 year boundary and facilitates an accurate hypothesis.

Now consider a hypothesis class $\mathcal{H}_2$ that allows the classifier to arbitrarily subdivide the space into regions (as in a decision tree with unbounded depth). This hypothesis class is extremely expressive and has weak inductive bias. Given the set of noisy examples in Figure 4.2, a learning algorithm using $\mathcal{H}_2$ is very likely to overfit and achieve low accuracy. In this case, relabeling is very important, because it reduces the likelihood that the classifier overfits the noise.

4.3.1 Bound Analysis

We now make these intuitions precise by bounding classification accuracy in terms of a classifier's Vapnik-Chervonenkis (VC) dimension [227]. Recall that the VC dimension of a

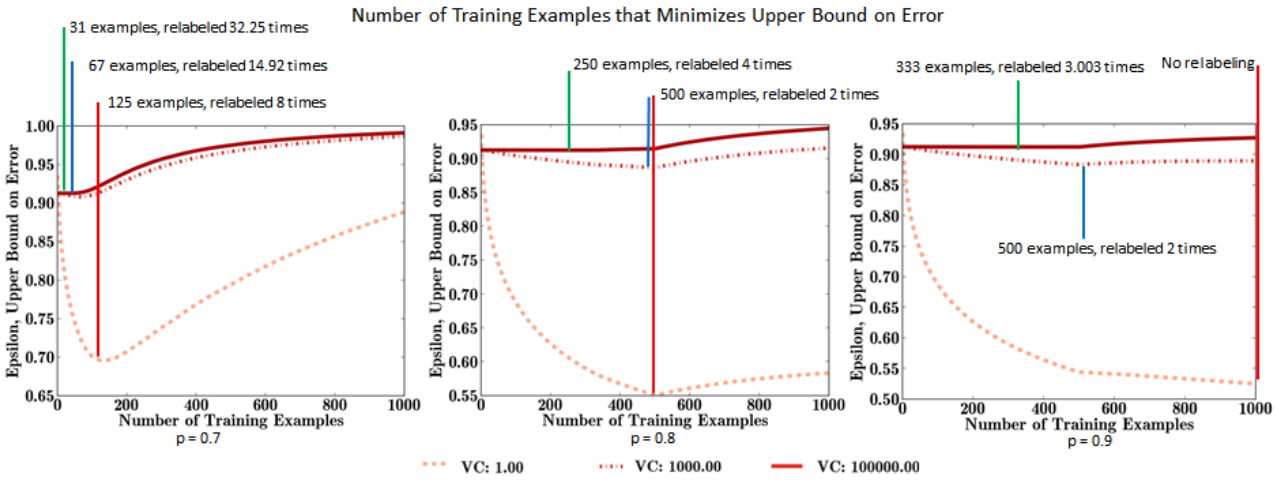

Figure 4.3: Given a fixed budget of 1000, we see in various settings of worker accuracy that as the VC dimension increases, more relabeling of a fewer number of examples achieves lower upper bounds on classification error.

classifier measures the size of the largest finite subset of $\mathcal{X}$ that it is capable of classifying correctly (shatter). For example, the VC dimension of $\mathcal{H}_1$ is equal to 2, because a threshold can correctly classify any subset of 2 points from $\mathcal{X}$, but cannot correctly classify all subsets of 3 points. A classifier may make errors when trying to learn datasets with size larger than its VC dimension, but is guaranteed to have a hypothesis that can distinguish all power sets of a dataset with size less than or equal to its VC dimension. A higher VC dimension corresponds to weaker inductive bias.

We assume that the concept class we are considering, $\mathcal{C}$, is Statistical-Query (SQ) learnable [112]. While the theory of SQ-learnability is beyond the scope of this dissertation, this assumption basically guarantees the existence of a classifier that can PAC-learn the target concept under noise. Aslam & Decatur [10] provide a sufficient bound on m, the number of samples needed to learn to a given accuracy. If each sample is incorrectly labeled with probability at most ξ_0, then an error less than ϵ with probability at least $1-\delta$ is guaranteed if m satisfies:

$$m \;=\; \mathcal{O}\left(\frac{1}{\tau^2\epsilon^2(1-2\xi_0)^2}\log^2\frac{1}{\epsilon}\cdot\left((VC+VC\log\frac{1}{\epsilon}\log\log\frac{1}{\epsilon})\cdot\log(\frac{1}{\tau\epsilon(1-2\xi_0)}\log\frac{1}{\epsilon})+\log\frac{1}{\delta}\right)\right),$$

where τ is a parameter that controls how easily the problem is SQ-learnable.

Notice that in our problem setting, the noise rate ξ_0 depends on the total budget b, the accuracy of the workers p, and the number of examples m used to train the classifier. Given a fixed budget b, as m is lowered, the noise rate ξ_0 decreases, because we use the budget for relabeling. Therefore, by fixing all parameters except for m and ϵ, we can use this bound to find the $m \leq b$ that minimizes ϵ. We now analyze this bound when the aggregation function ζ is a majority vote.

We begin by computing ξ_0 given our choice of m. Since m may not divide the budget b perfectly, one or more training examples may get an additional label. Let $K = \lfloor \frac{b}{m} \rfloor$ be the minimum number of labels each example has if we train using m examples with budget b. Let $m_{K+1} = b - (mK)$ denote the number of examples with $K+1$ labels. Let $m_K = m - m_{K+1}$ be the number of examples with K labels. Then, we compute ξ_0 as:

$$\xi_0 = \frac{m_K(1.0 - \eta_{\zeta_K}(p)) + m_{K+1}(1.0 - \eta_{\zeta_{K+1}}(p))}{m}$$

Because the function that computes aggregate accuracy for majority vote, η_{ζ_k}, is not defined for k that is even, we use the defined points for when k is odd along with their reflections across the axes and fit a logistic curve of the form

$$\eta_{\zeta_k} = \frac{c_0}{1 + e^{-c_1(k-c_2)}}$$

in order to estimate η_{ζ_k}. We use Scipy's curve fitting library [98], which uses the Levenberg-Marquardt algorithm. The resulting curve is not perfect, but accurately reflects the shape of η_{ζ_k}.

Solving for ϵ analytically is difficult, so we employ numerical methods. Since we are only concerned with the change in ϵ as a function of m across various VC dimensions, constant factors don't matter so we set $\tau = 0.1$ and $\delta = 0.1$. Now we use Scipy's optimization library [98] to solve for the error bound, ϵ, for all values of $m < b = 1000$.

Figure 4.3 shows the curves that result for various settings of worker accuracy and VC dimension. We see that as the VC dimension increases, m, the number of examples (each labeled $\approx 1000/m$ times) that minimizes the upper bound on error, decreases. Thus the optimal relabeling redundancy increases with increasing VC dimension. Furthermore, note that when workers are 90% accurate, unilabeling yields the lowest bounds at low VC dimensions while relabeling produces the lowest bounds when VC dimension is high.

Our analysis suggests a method to pick *label redundancy*, the number of times to relabel each example. We can simply choose the value of m that minimizes the upper bound on error for the given VC dimension. One caveat: the value which produces the minimum error *bound* does not necessarily guarantee the minimum *error*, unless the bound is tight. Still, we plan to experiment with this heuristic in the future.

4.3.2 Simulated Datasets

We now present experiments that empirically study the effect of inductive bias on relabeling accuracy. For these experiments we test on an artificial dataset, which allows us to control for various parameters. Our datasets contain two Gaussian clusters, which correspond to the two classes. To generate a dataset, we first we pick the number of features to be $z = 50$. Then we randomly pick two means, $\mu_1, \mu_2 \in [0, 1]^z$. Next we randomly pick two corresponding covariance matrices $\Sigma_1, \Sigma_2 \in [0, 1]^{z \times z}$.

For each Gaussian cluster (class), we generate an equal number of examples. Setting a labeling budget of $b = 500$, we can now train classifiers. We compare a unilabeling strategy against relabeling strategies using 2/3-, 3/5- and 4/7-relabeling. We simulate moderately accurate workers ($p = 0.75$) using the classification noise model. We average each strategy over 1000 runs, and use standard error to compute 95% confidence intervals.

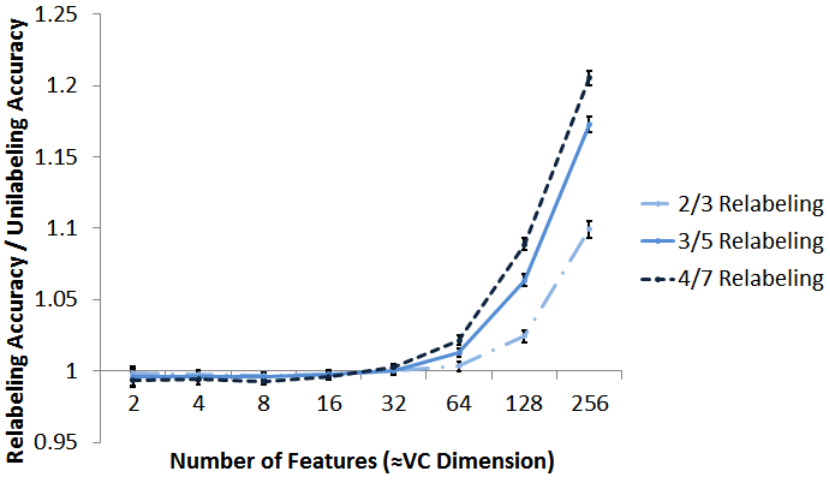

Figure 4.4: As the number of features (VC dimension) increases, relabeling becomes more and more effective at training an accurate logistic regression classifier.

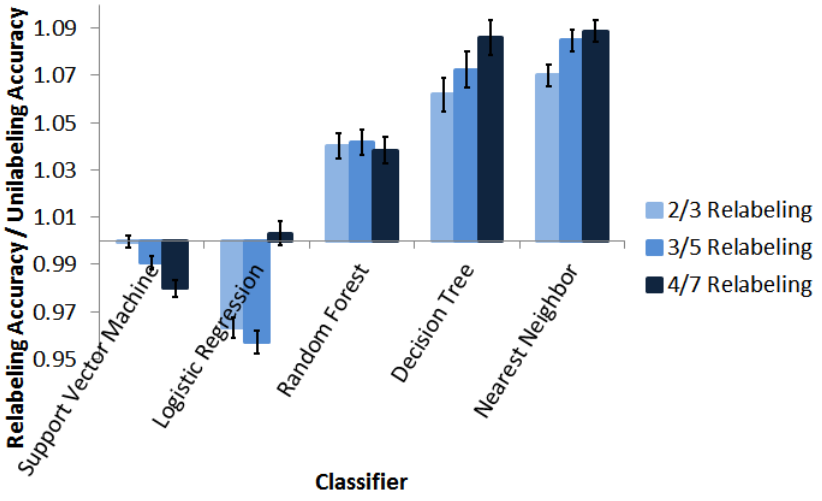

Figure 4.5: Classifiers with weaker inductive bias tend to benefit more from relabeling.

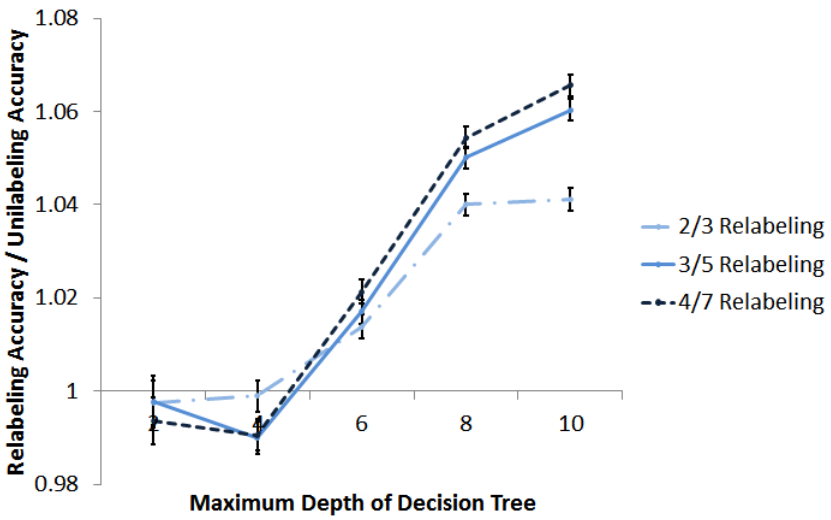

Figure 4.6: Decision trees with lower regularization tend to benefit more from relabeling.

Figure 4.4 shows how VC dimension affects the power of relabeling. Here we use a logistic regression classifier and set an $l2-$regularization with a balanced regularization constant such that relabeling strategies (which receive a fewer number of examples) are not unfairly over-regularized. We use a linear classifier so that as we vary the number of features, we vary the VC dimension, which is equal to the number of features plus one [70]. We see that as the VC dimension increases, relabeling becomes more cost effective.

Figure 4.5 shows how different types of classifiers perform. We use classifiers from the Scikit-learn [168] package in their default settings. We see that logistic regression and support vector machine (SVM) both perform best with a unilabeling strategy, but decision trees, random forests, and nearest neighbor classifiers do not, because these classifiers have high expressiveness and weak inductive bias.

Figure 4.6 shows how a decision tree classifier performs as we vary its maximum depth. Since increasing the depth of a decision tree increases the expressiveness of the corresponding logical formula, increasing depth corresponds to weaker inductive bias. We see that as the maximum depth increases, relabeling becomes the more effective strategy.

Our experiments validate the insights described previously. Overall, we believe that

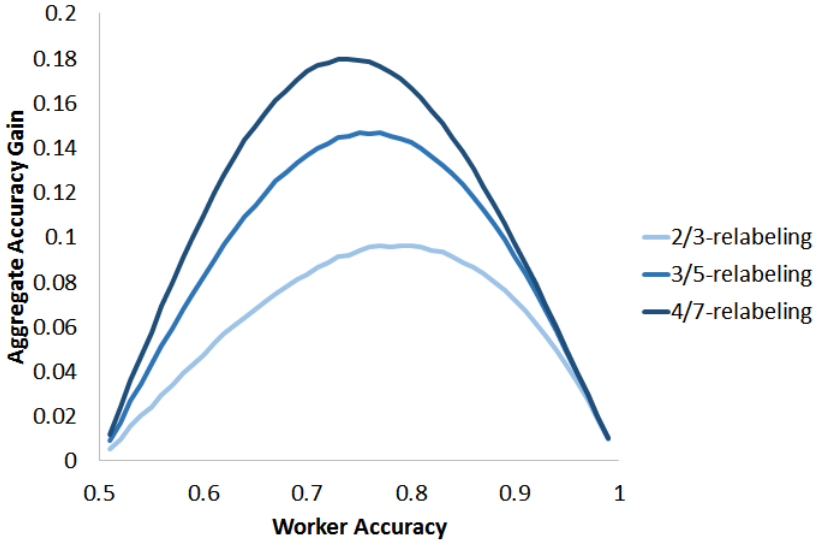

Figure 4.7: The increase in the aggregate accuracy of the training set data when using relabeling instead of unilabeling for various worker accuracies.

for low-dimensional data and strongly biased classifiers, unilabeling may be the method of choice. If, however, the VC dimension is high, then relabeling is likely preferred.

4.4 The Effect of Worker Accuracy

We now consider the effect of worker accuracy on relabeling. In the extreme case, relabeling cannot possibly help if workers are perfect. Conversely, there is seemingly great potential to improve the quality of one's training set when worker accuracies are barely over $p = 0.5$. Hence, our a priori belief was that relabeling should be more effective when workers are less accurate.

However, this intuition is faulty as we now explain. Indeed, [92] show that typical relabeling strategies have the maximum effect *on the accuracy of a training set* (not on the resulting classifier), when workers are of intermediate abilities. Consider Figure 4.7, which plots the increase in aggregate accuracy of the training data when relabeling instead of unilabeling as a function of worker accuracy. The three peaks happen between 0.73–0.79.

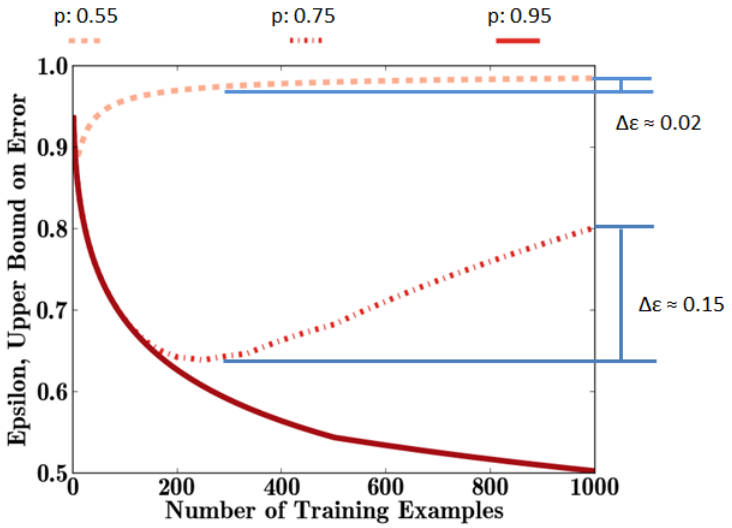

Figure 4.8: When the workers are moderately accurate ($p = 0.75$), the decrease in the upper bound on error from additional relabeling is greatest.

We also observe that 2/3-relabeling only improves accuracy by about 0.1 in the best case, whereas 4/7-relabeling can get to an almost 0.2 increase. Furthermore, as the amount of relabeling is increased, the peak in accuracy gain moves to the left, suggesting that strategies with increasing amounts of relabeling have their maximum effect as the workers become less accurate.

But these past results only apply to the quality of a training set, not the accuracy of the resulting classifier. By considering the accuracy of the classifier, we must address the confounding factor that eschewing relabeling frees budget to be spent labeling new examples. To study this scenario further we continue the analysis technique from the previous section to produce Figure 4.8, which compares various upper bound curves for different settings of worker accuracy in the setting of VC=1 and budget=1000. Consider the difference in error bound as m ranges between 333 (when every example is labeled 3 times) to 1000 (when thrice as many examples are labeled once). This delta is much greater when the workers are moderately accurate ($p = 0.75$) than for other settings of worker skill. These differences in error bound support the belief that typical relabeling strategies are most likely to reduce

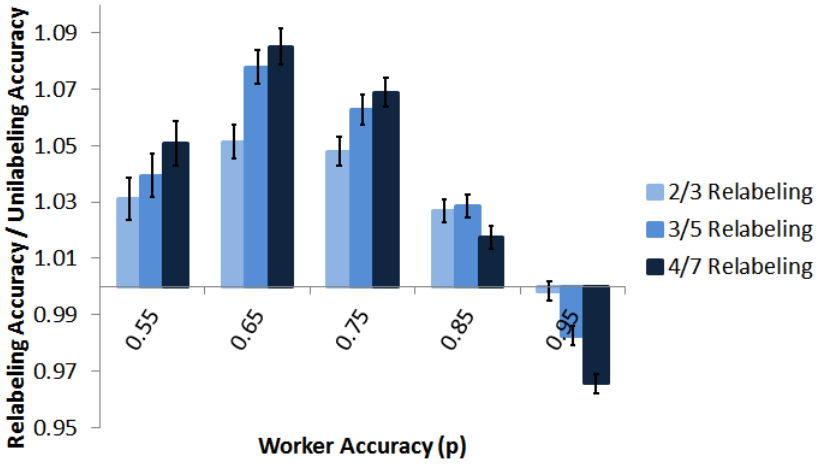

Figure 4.9: For simulated Gaussian datasets, relabeling strategies based on majority-vote are most powerful at moderate values of worker accuracy.

classifier error when p is not an extreme value.

4.4.1 Simulated Datasets

To confirm these insights, we again present experimental analysis using our artificial Gaussian datasets, and use varying settings of worker accuracy. As in the previous section, we fix the number of features to be $z = 50$, and the budget to be $b = 500$. We train using decision trees and set the maximum depth to be 10. For this experiment, instead of averaging over 1000 runs, we average over 2000 runs in order to create tight confidence intervals across varying worker accuracies.

Figure 4.9 shows our results. The more highly redundant approaches, 4/7- and 3/5-relabeling, clearly have their maximum benefit when workers are 65% accurate. On the other hand, 2/3-relabeling has its maximum benefit somewhere between $p = 0.65$ and $p = 0.75$. These results mirror our intuition and our theoretical analysis. Thus, choosing the correct amount of relabeling redundancy is a complex decision which ideally should be informed by knowledge of worker accuracy.

4.5 The Effect of Budget

We now investigate the effect of budget on relabeling power. Intuitively, one might think that as the budget increases, relabeling will become the more effective strategy, because in the extreme case of when the budget is infinitely large, we should clearly label each example infinitely many times. Such a strategy allows us to train the classifier using the entire set of noiseless examples. However, this extreme case does not arise in typical finite budgets, and in fact our experiments show quite an opposite trend.

We again train a decision tree with a maximum depth of 10 using our simulated Gaussian datasets with $z = 50$ features and moderately accurate workers ($p = 0.75$), and we vary the total budget. We plot the resulting learning curves in Figure 4.10, which are averaged over 1000 runs. We see that while initially relabeling strategies achieve higher accuracy than unilabeling, eventually unilabeling overtakes relabeling somewhere around a budget of $60,000$, because the slope of the unilabeling learning curve is higher than that of the relabeling curves. Indeed, increasing the amount of relabeling decreases the slope while increasing the initial accuracy.

Upon reflection, such a result makes sense. With very low budgets, classifiers are unable to generalize well with noisy data. However, with a large enough budget, the noise becomes simply that: irrelevant noise. There are enough accurate examples such that the classifier can learn a correct hypothesis that ignores the noisy data.

We also plot learning curves using poor workers ($p = 0.55$) in Figure 4.11 in order to show the effect more clearly. When the budget reaches approximately $4,000$, unilabeling begins to achieve higher accuracies than relabeling. Interestingly, these two figures also show the effect of worker accuracy. Unilabeling takes much longer to start achieving higher accuracies when the workers are moderately accurate because relabeling strategies are most powerful in this setting.

We conclude that increasing the budget tends to benefit unilabeling, and the point at which unilabeling defeats relabeling is controlled by other factors, like worker accuracy.

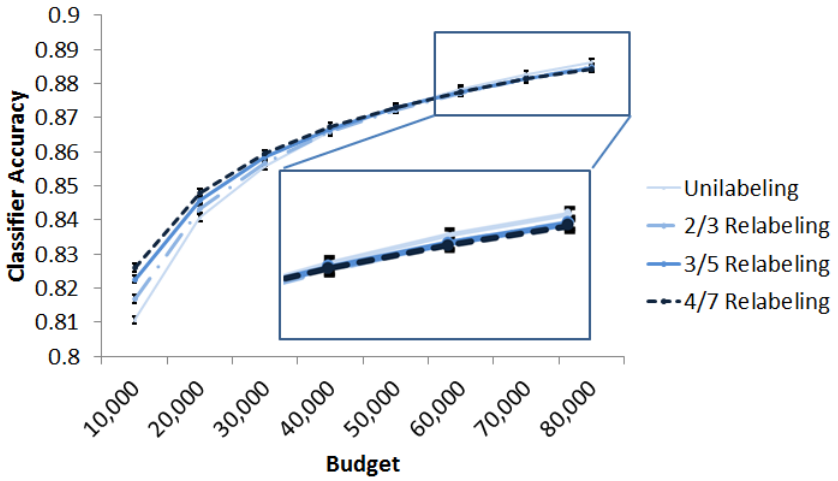

Figure 4.10: When $p = 0.75$, relabeling strategies initially achieve higher accuracies than unilabeling, but are eventually defeated.

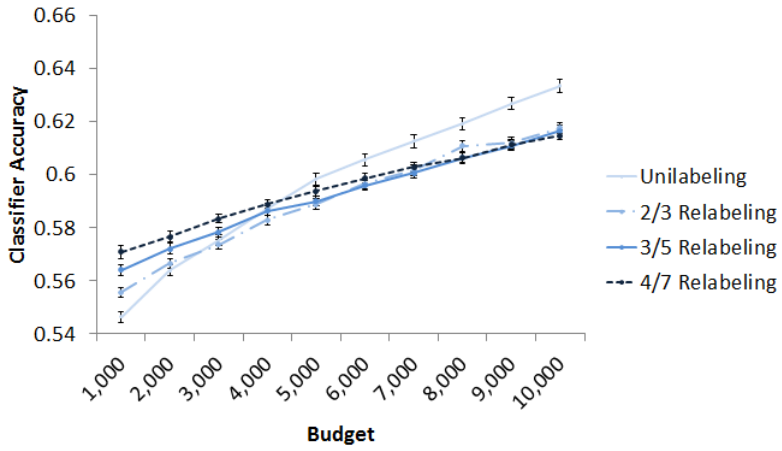

Figure 4.11: When $p = 0.55$, relabeling strategies initially achieve higher accuracies than unilabeling, but are defeated earlier than when $p = 0.75$.

Dataset	# Features	# Examples
(a) Breast Cancer	9	699
(b) Bank Note Authentication	4	1372
(c) Seismic Bumps	18	2584
(d) EEG Eye State	14	14980
(e) Sonar	60	208
(f) Breast Cancer Diagnostic	30	569
(g) Hill-Valley	100	606
(h) Hill-Valley with Noise	100	606
(i) Internet Ads	1558	2359
(j) Gisette	5000	6000
(k) Farm Ads	54877	4143
(l) Spambase	57	4601

Table 4.1: The 12 datasets we use, with the total number of examples and number of features in each.

4.6 Real Dataset Experiments

We now compare unilabeling and relabeling using 12 datasets from the UCI Machine Learning Repository [13], with the goal of matching trends observed in simulated datasets with real-world datasets. We list the datasets in Table 4.1. We use half of the available examples as the budget, and hold out 15% of the examples for testing. We simulate workers at accuracies of $p = 0.55$ and $p = 0.75$. We train a logistic regression classifier (a linear classifier) so that we can observe trends with varying VC dimension. Our results, averaged over 1000 runs, are shown in Figures 4.1 and 4.12.

We see that when the workers are poor ($p = 0.55$), domains are split evenly with respect to which strategy is performing better. This can be explained based on the VC dimension. In

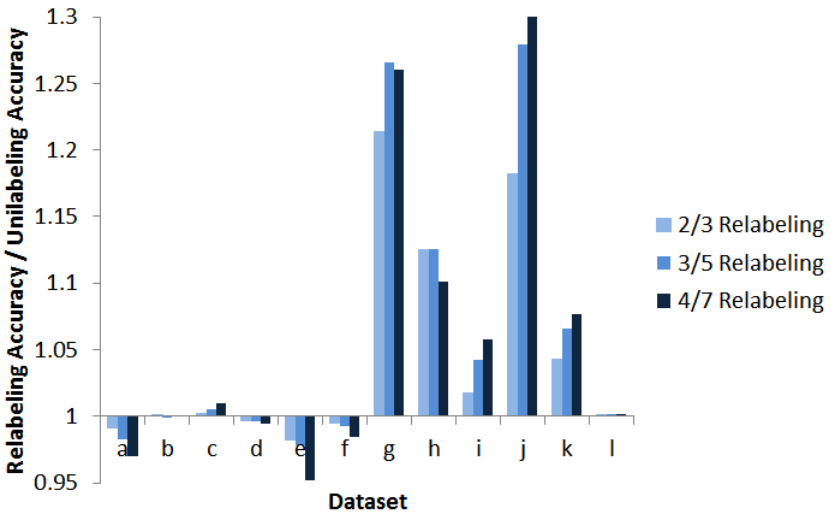

Figure 4.12: Unilabeling obtains better classifiers in some datasets, even when the workers are moderately accurate (p=0.75).

the five domains with high VC dimensions (100 or more features) relabeling is outperforming unilabeling.

When comparing Figures 4.1 and 4.12 we observe that relabeling performance has improved with higher worker accuracy. This result is directly explained by the analysis in the section on worker accuracy. We see that 75% accuracy results in quick improvement of training quality using majority vote; 55% accuracy is a bad setting since relabeling only results in slow improvement.

Overall, we find that the experiments on real datasets confirm the trends discussed earlier and shown in our simulation experiments.

4.7 Related Work

A large body of work develops various methods for aggregating labels. For example, see [58, 242, 177, 133]. Of note is BBMC [236], which develops a model that integrates active learning with data curation and model learning. Their algorithm can potentially trade off between relabeling and acquiring labels for new examples, but it is not general and is tied

to their own classifier. Further, they do not consider this tradeoff. Note that the previous two chapters consider automated methods to decide when to relabel, but the goal is data accuracy instead of classifier accuracy.

Several researchers have considered *how* to pick examples or workers for (re)labeling when active learning or selective sampling [67, 66, 68, 246, 59, 198, 253]. However, unlike our work, these do not answer the fundamental question of *when* to relabel.

Agnostic Learning [113, 15, 80] is a general learning setting that makes little to no assumptions about learners and datasets. In this setting, noise refers to the labels that are inconsistent with the best hypothesis that is available to the learner. Thus agnostic learning can be viewed as a setting in which the goal is to train a classifier that fits both the noise and the data as well as possible. This scenario is inherently different than the one we consider, where noise is an incorrect label, not an inconsistent one, and we want to learn a classifier that fits the ground truth despite the noise.

Many works (*e.g.* [161, 114, 48]) design noise-tolerant classifiers. However, these works are orthogonal to ours in purpose. We focus on the tradeoff between unilabeling and relabeling for any black-box classifier. Our results can inform the relabeling strategy for noise-tolerant classifiers.

Several works seek to understand the sample complexity of classifiers under noise. [122, 8] derive bounds for classifiers that minimize their training error. The Statistical Query Model [112] can show that many PAC learning algorithms can be transformed into ones which tolerate classification noise.

4.8 Conclusion

We have shown that when using crowdsourcing to learn the most accurate classifier possible with a fixed budget, relabeling examples should not be a default go-to strategy, as unilabeling often results in higher accuracies. We provide theoretical justification and empirical evidence using simulated and real datasets to show the following:

- Relabeling provides the most benefit to expressive classifiers with weak inductive bias. When the classifier being trained is linear, a relabeling strategy (and, indeed, higher levels of redundancy) is more likely to be appropriate when the domain has a large number of features.

- Typical relabeling strategies provide the most benefit when workers are moderately accurate, and not when they are extremely error-prone, as one might naïvely suspect. Unilabeling is preferred when workers are very accurate.

- As the labeling budget increases, unilabeling provides increasing benefits, but relabeling is often the more effective strategy when the budget is small.

Chapter 5

RE-ACTIVE LEARNING: ACTIVE LEARNING WITH RELABELING

5.1 Introduction

In the last chapter, we examined how various properties of learning algorithms affect the noise versus size tradeoff when collecting training data. However, we did not prescribe a method for actually making this tradeoff, which is the goal of this chapter. To accomplish this goal, we maintain the assumptions of the last chapter (all tasks are the same and all workers are independently and identically Bernoulli-distributed), and we build on the long-studied subfield of *active learning*.

Active learning algorithms reason about the best examples to annotate in order to minimize the expense of labeling training data. However, traditional active-learning methods assume a *single* annotator, either perfect [190] or noisy [113]. In this setting, such algorithms always pick a *new* example to label, since they can gather no new information about examples which have already been labeled. We extend this traditional active learning framework to allow for relabeling in a generalization we call *re-active learning*. In re-active learning, we wish to answer the crucial question "Which example should we label or re-label next in order to maximize classifier performance?"

Standard active learning strategies like uncertainty sampling [126] and expected error reduction [182, 108] can be naïvely extended for re-active learning by allowing them to pick *any* (labeled or unlabeled) point using their existing approaches. Unfortunately, we show that such extensions do not perform well, because they do not utilize all sources of knowledge, are myopic, or both. These extensions often suffer from infinite looping, when the learner repeatedly annotates the same example, because they only consider information gleaned from

the classifier and ignore the valuable information stored in the number and agreement of the gathered labels.

Therefore, we introduce alternative extensions of uncertainty sampling that can reason about the information in the labels. We also propose a new class of algorithms, *impact sampling*, which picks the example to label that has the potential to change the classifier the most. By reasoning about whether an additional label can change the classifier, impact sampling elegantly eliminates the starvation problems described above. Many active learning methods use greedy search to reduce combinatorial explosion, but the resulting myopia is especially problematic with relabeling — if the first two labels agree, then a third may have no effect. To combat this problem, we introduce a novel technique, called pseudo-lookahead, that can tractably mitigate myopia. We then characterize the relationship between impact sampling and uncertainty sampling, and show that, surprisingly, in many noiseless settings, impact sampling can be viewed as a generalization of uncertainty sampling. Finally, we conduct empirical experiments on both synthetic and real-world datasets, showing that our new algorithms significantly outperform traditional active learning techniques and other natural baselines on the problem of re-active learning.

5.2 Preliminaries

We now set up the framework for re-active learning, which is very similar to the framework for the last chapter. Let $\mathcal{X}$ denote the space of examples, $\mathcal{Y} = \{0, 1\}$ a set of labels, and D, a distribution over $\mathcal{X}$. Let the true concept be $h^* : \mathcal{X} \to \mathcal{Y}$. Let $\mathcal{H}$ be a class of hypotheses, from which our learning algorithm, $\mathcal{A}$, tries to select the $h \in \mathcal{H}$ that minimizes the error $\epsilon(h) = P_{x \sim D}(h(x) \neq h^*(x))$. We assume that each worker exhibits the same (but unknown) accuracy $p \in (0.5, 1]$, an assumption known as the *classification noise model* [8], and we assume worker errors are independent. We assume that acquiring a label for an example incurs a fixed unit cost.

Let $\mathcal{X}_L \subseteq \mathcal{X}$ denote the current set of labeled examples, and $\mathcal{X}_U = \mathcal{X} - \mathcal{X}_L$ denote the set of unlabeled examples. Let $L = \{(x_i, y_i)\}$ denote the multiset of example and label pairs,

and for each $x_i \in \mathcal{X}_L$, let $L_{x_i} = \{l_i^1, \ldots, l_i^{\Xi_i}\}$ be the multiset of labels for x_i, where Ξ_i is the number of labels for x_i. Let $\zeta(L_{x_i})$ output an aggregated label for an example given the noisy labels for that example. ζ can be as simple as majority vote or use more complex statistical techniques (e.g., [58, 242, 133]). We run our learning algorithm $\mathcal{A}$ using L and the corresponding aggregated labels output by ζ. Given the current L, the goal of *re-active learning* is to select an example $x \in \mathcal{X}$ (not $x \in \mathcal{X}_U$, as in traditional active learning) such that acquiring a label for x and adding it to L minimizes the long-term error of the classifier output by $\mathcal{A}$.

5.3 Algorithms For Re-Active Learning

5.3.1 Uncertainty Sampling

Uncertainty sampling [126] is one of the most popular algorithms for active learning [190]. To pick the next example to label, it simply computes a measure of the classifier's uncertainty for each example in the unlabeled set, $\mathcal{X}_U$, and then returns the most uncertain one. Let $P_{\mathcal{A}}(h^*(x_i) = y)$ denote the probability output by the learning algorithm that $h^*(x_i) = y$ after training on L. Then let $M_A(x_i) = -\sum_{y \in \mathcal{Y}} P_{\mathcal{A}}(h^*(x_i) = y) \log P_{\mathcal{A}}(h^*(x_i) = y)$ denote the entropy of those probabilities for an example x_i. We denote as $\texttt{US}_{\mathcal{X}_U}$ the strategy that returns the example in $\mathcal{X}_U$ with highest entropy: $\text{argmax}_{x \in \mathcal{X}_U} M_A(x)$.

One could naïvely apply uncertainty sampling to re-active learning by allowing it to sample from the full sample space $\mathcal{X} = \mathcal{X}_U \cup \mathcal{X}_L$. We denote this algorithm $\texttt{US}_{\mathcal{X}}$. Unfortunately, $\texttt{US}_{\mathcal{X}}$ can result in extremely poor performance.

Consider Figure 5.1. Suppose that x_1 and x_2 are the only labeled points in this distribution of diamonds and circles and h is the current hypothesis. Then, $\texttt{US}_{\mathcal{X}}$ will pick x_1 or x_2 to relabel, because they are the closest to h, and thus have the highest uncertainty. In fact, $\texttt{US}_{\mathcal{X}}$ will likely pick x_1 or x_2 to relabel repeatedly, because each time it picks x_1 or x_2, it will receive a label that will most likely not change the aggregated label, $\zeta(L_{x_i})$ (assuming low label noise). Since the labels used to train the classifier will not change, the classifier itself

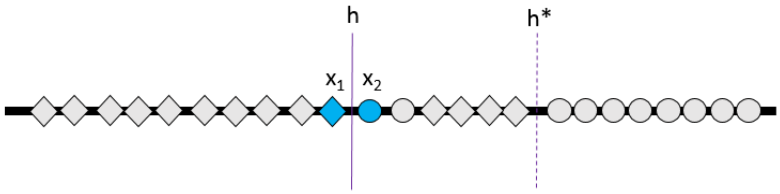

Figure 5.1: An example domain in which a naïve extension of uncertainty sampling to re-active learning, $\texttt{US}_{\mathcal{X}}$, can fall into a trap. Given that x_1 and x_2 are the only labeled points in this distribution of diamonds and circles and h is the current hypothesis, uncertainty sampling is likely to converge to behavior in which it will always pick these same examples to relabel and cause an infinite loop in which no learning takes place. The true hypothesis, h^*, will never be learned.

will not change, forming an infinite loop during which other useful points get starved and no learning takes place, causing convergence to a suboptimal hypothesis.

This weakness is similar to the hasty generalization problem found in active learning [57], but is distinct, because points are not relabeled in traditional active learning. We find experimentally that $\texttt{US}_{\mathcal{X}}$ gets stuck in unproductive infinite loops quite often. The problem is that in many cases, the most uncertain example (according to the classifier) could be a labeled example, which is actually quite certain (according to the current label multiset).

Extensions of Uncertainty Sampling for Re-Active Learning

Clearly, any extension of uncertainty sampling to re-active learning needs to consider both $M_A(x_i)$, the classifier's uncertainty, and the *label's* uncertainty, which we denote $M_L(x_i)$. We define $M_L(x_i)$ as the entropy of the labels themselves: $M_L(x_i) = -\sum_{y \in \mathcal{Y}} P(h^*(x_i) = y \mid L_{x_i}) \log P(h^*(x_i) = y \mid L_{x_i})$, where the label posterior $P(h^*(x_i) \mid L_{x_i})$ is computed by applying Bayes' rule to the observed labels L_{x_i} on top of a uniform prior:

$$\begin{aligned} P(h^*(x_i) \mid L_{x_i}) &\propto P(L_{x_i} \mid h^*(x_i)) \\ &= \prod_{l_i^j \in L_{x_i}} \mathbb{1}(h^*(x_i) = l_i^j)p + \mathbb{1}(h^*(x_i) \neq l_i^j)(1-p). \end{aligned}$$

We propose a new aggregate uncertainty measure, which is a weighted average of these

two uncertainties: $(1-\alpha)M_A(x_i)+\alpha M_L(x_i)$, where $\alpha \in [0,1]$. We denote this new algorithm, which picks the example x_i with the highest aggregate uncertainty, as $\texttt{US}^{\alpha}_{\mathcal{X}}$. By definition, $\texttt{US}^{0}_{\mathcal{X}}$ is equivalent to $\texttt{US}_{\mathcal{X}}$. However, we also note the following interesting and counter-intuitive fact that weighting *label* uncertainty highly (large α) results in behavior equivalent to that of standard uncertainty sampling for active learning ($\texttt{US}_{\mathcal{X}_U}$), which only considers classifier uncertainty. We first provide an intuitive explanation before presenting a formal proof.

First, notice that the label uncertainty, M_L, of any unlabeled example is the highest possible. There must exist a weighting such that combination of the highest possible classifier uncertainty, M_A, and the second highest possible label uncertainty is not large enough to exceed the label uncertainty of an unlabeled example. With this weighting, $\texttt{US}^{\alpha}_{\mathcal{X}}$ will never relabel an example, and instead pick among the unlabeled examples. Since all the unlabeled examples have the same label uncertainty, the example it picks will be determined based on the classifier's uncertainty, just as in $\texttt{US}_{\mathcal{X}_U}$.

Theorem 5.1. *For any given learning problem with no existing labels or only singly-labeled examples, there exists an $0 < \alpha' < 1$ such that for all $\alpha \in [\alpha', 1]$, $\texttt{US}^{\alpha}_{\mathcal{X}}$ and $\texttt{US}_{\mathcal{X}_U}$ will always select the same example to label. Thus, $\texttt{US}^{\alpha}_{\mathcal{X}}$ is equivalent to $\texttt{US}_{\mathcal{X}_U}$.*

Note that this theorem does not say that $\texttt{US}^{\alpha}_{\mathcal{X}}$ will select the same example to label as $\texttt{US}_{\mathcal{X}_U}$ given an arbitrary L constructed using another method. But it does say that if they both start with an empty label multiset L, they will always select the same example to label next at every timestep through the re-active learning process.

Proof. We first show that the theorem is true when $\mathcal{X}_L$ only contains singly-labeled examples. By definition, $\texttt{US}^{\alpha}_{\mathcal{X}}$ will always pick the unlabeled example with the highest classifier uncertainty $x^*_u = \text{argmax}_{x_u \in \mathcal{X}_U} M_A(x_u)$, if the following Condition holds: α is set such that $(1-\alpha)M_A(x_l) + \alpha M_L(x_l) < (1-\alpha)M_A(x^*_u) + \alpha M_L(x^*_u)$ for all $x_l \in \mathcal{X}_L$. There are two cases: when $M_A(x_l) \leq M_A(x^*_u)$ and when $M_A(x_l) > M_A(x^*_u)$. In each case, we find an α' such that for all $\alpha > \alpha'$ the Condition holds, and then use the larger α' to satisfy the theorem.

We first look at the first case. Since x_l is singly-labeled and x_u^* is unlabeled, $M_L(x_u^*) > M_L(x_l)$ (because $p > 0.5$). Therefore, $\alpha M_L(x_u^*) > \alpha M_L(x_l)$ for all $\alpha > 0$. Similarly, because we assume $M_A(x_l) \leq M_A(x_u^*)$, we have $(1-\alpha)M_A(x_l) \leq (1-\alpha)M_A(x_u^*)$ for all $\alpha < 1$. Let $\alpha' = 0$. Then we have that for all $\alpha \geq \alpha'$ and $\alpha \leq 1$, the Condition holds.

Now we look at the second case. We see via algebra that $(1-\alpha)M_A(x_l) + \alpha M_L(x_l) < (1-\alpha)M_A(x_u^*) + \alpha M_L(x_u^*)$ when $\alpha > \frac{M_A(x_l)-M_A(x_u^*)}{M_A(x_l)-M_A(x_u^*)+M_L(x_u^*)-M_L(x_l)}$. Let $\alpha' = \max_{x_l \in \mathcal{X}_L} \frac{0.69}{0.69+(M_L(x_u^*)-M_L(x_l))}$. We first prove that $0 < \alpha' < 1$. As we noticed in the first case, $M_L(x_u^*) > M_L(x_l)$, so we have that $M_L(x_u^*) - M_L(x_l) > 0$. Therefore, $0 < \alpha' < 1$. Now we prove that the Condition holds true when $\alpha > \alpha'$. Since M_A is an entropy of a binary random variable, by definition, the maximum possible entropy is $-2(0.5)\ln(0.5) = 0.69$. Therefore, $M_A(x_l) - M_A(x_u^*) \leq 0.69$ for all x_l which means that $\frac{0.69}{0.69+(M_L(x_u^*)-M_L(x_l))} \geq \frac{M_A(x_l)-M_A(x_u^*)}{M_A(x_l)-M_A(x_u^*)+M_L(x_u^*)-M_L(x_l)}$ for all x_l. Since α' is an upper-bound on $\frac{0.69}{0.69+(M_L(x_u^*)-M_L(x_l))}$, we have $\alpha' \geq \frac{M_A(x_l)-M_A(x_u^*)}{M_A(x_l)-M_A(x_u^*)+M_L(x_u^*)-M_L(x_l)}$ for all x_l. Therefore, the Condition holds true when $\alpha > \alpha'$ for all x_l.

Because in the first case α can be set arbitrarily between 0 and 1, the α' set in the second case renders the Condition true for all $x_l \in \mathcal{X}_L$. Thus, we have shown the theorem is true when $\mathcal{X}_L$ only contains singly-labeled examples. Now, since both $\texttt{US}_{\mathcal{X}}^{\alpha}$ and $\texttt{US}_{\mathcal{X}_U}$ start with $\mathcal{X}_L = \emptyset$, by induction, $\mathcal{X}_L$ will only ever contain singly-labeled examples, and so these two strategies are equivalent. □

An alternative way we can use the popular framework of uncertainty sampling to fit the new framework of re-active learning is to simply use $\texttt{US}_{\mathcal{X}_U}$, but label each new example a fixed number of times. We use $\texttt{US}_{\mathcal{X}_U}^{j/k}$ to denote the algorithm that picks a new example to label via $\texttt{US}_{\mathcal{X}_U}$, and then relabels that example using j/k-relabeling, where the algorithm can request up to k labels, stopping as soon as $j = \lceil k/2 \rceil$ identical labels are received. Unfortunately, for both $\texttt{US}_{\mathcal{X}}^{\alpha}$ and $\texttt{US}_{\mathcal{X}_U}^{j/k}$, learning optimal values for hyperparameters, α or k, can be difficult.

5.3.2 Expected Error Reduction

Expected error reduction (EER) [182, 108] is another active learning technique that can be extended to the framework of re-active learning. EER simply chooses to label the example that maximizes the expected reduction in classifier error, where the expected error of a classifier is estimated using the probabilities output by that classifier.

To extend EER to consider relabeling, we must first compute the probability, $Q(y|x_i)$, that we receive label y if we query example x_i. Let $P_{\mathcal{A}}(h^*(x_i) = y \mid L_{x_i})$ be the probability that the correct label of x_i is y; we may compute it by first using the currently learned classifier's beliefs as a prior and then performing a Bayesian update with the observed labels L_{x_i}:

$$\begin{aligned} P_{\mathcal{A}}(h^*(x_i) \mid L_{x_i}) &\propto P(L_{x_i} \mid h^*(x_i)) P_{\mathcal{A}}(h^*(x_i)) \\ &= P_{\mathcal{A}}(h^*(x_i)) \prod_{l_i^j \in L_{x_i}} \mathbb{1}(h^*(x_i) = l_i^j)p + \mathbb{1}(h^*(x_i) \neq l_i^j)(1-p). \end{aligned}$$

Then, $Q(y|x_i) = pP_{\mathcal{A}}(h^*(x_i) = y \mid L_{x_i}) + (1-p)P_{\mathcal{A}}(h^*(x_i) \neq y \mid L_{x_i})$. Now, let $L \oplus \{(x_i, y)\}$ denote the addition of the element (x_i, y) to the multiset L, and let $\epsilon(\mathcal{A}|L)$ denote the error of the classifier returned by running $\mathcal{A}$ using the labels, L. Conceptually, EER returns $\text{argmin}_{x \in \mathcal{X}}[\sum_{y \in \mathcal{Y}} Q(y|x)\epsilon(\mathcal{A}|L \oplus \{(x, y)\})] - \epsilon(\mathcal{A}|L)$, and it approximates ϵ using the probabilities output by the learned classifier. In particular, if $P_{\mathcal{A}}(h^*(x_i) = y)$ is the probability output by $\mathcal{A}$ that the correct label of x_i is y, then ϵ can be computed as $\sum_{x_i \in \mathcal{X}} \min_y P_{\mathcal{A}}(h^*(x_i) = y)$.

Unlike $\texttt{US}_{\mathcal{X}}$, EER is able to incorporate knowledge about label uncertainty. Unfortunately, like $\texttt{US}_{\mathcal{X}}$, EER can starve points. Suppose that the expected reduction in error from querying any unlabeled point x_u is negative. Such a result can occur due to the myopic nature of EER and the amount of uncertainty in the problem. Adding a single additional example may, in the short-term, cause the classifier to make more mistakes than before. Next, suppose that the expected reduction in error from querying any labeled point x_l is 0. Such a result is common, since oftentimes a single extra label will not change the resulting aggregated labels output by f, especially if x_l has already been labeled many times. For example, if f is a

simple majority vote, and L_{x_l} contains at least 2 more labels of one class than of another, then adding an additional label to L_{x_l} will have zero expected error reduction. In this scenario, EER will query x_l over and over, starving valuable data points as did $\texttt{US}_{\mathcal{X}}$.

As a concrete example, consider the toy distribution in Figure 5.2. Suppose we are training a classifier $\mathcal{A}$ that is max-margin, and which outputs probability predictions $P_{\mathcal{A}}(h^*(x_i) = \text{diamond})$ and $P_{\mathcal{A}}(h^*(x_i) = \text{circle})$ that scale linearly with distance from the hyperplane. Next, suppose x_5 is the only unlabeled example and it is so close to h that $P_{\mathcal{A}}(h^*(x_5) = \text{diamond}) = P_{\mathcal{A}}(h^*(x_5) = \text{circle}) = 0.5$. Finally, suppose that x_1, x_2, x_3, and x_4 are the only labeled examples, which have already been labeled many times so that they have converged, and they are far enough from h so that $P_{\mathcal{A}}(h^*(x_1) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_2) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_3) = \text{circle})$, and $P_{\mathcal{A}}(h^*(x_4) = \text{circle})$ are all equal to 1. Then the error of the current hypothesis h is 0.5, because of negligible error from x_1, x_2, x_3, and x_4 and 0.5 error from x_5. Adding an additional label to the labeled examples will not cause the hypothesis to move, and so the error remains 0.5, leading to a zero expected error reduction. However, labeling x_5 will move the hypothesis to h', which *increases* the expected error from 0.5 to 0.75, because of negligible error from the two examples furthest away and 0.25 error from the three examples closest to h'. Thus, in this example, EER will never query x_5 and label x_1, x_2, x_3, and x_4 infinitely many times.

This infinite-looping problem is exacerbated by the fact that full EER is computationally intractable. Just computing the expected future error for a single new label requires retraining a classifier twice. In order to make EER practical, one must restrict the set of examples from which EER may recommend points. However, reducing the number of points that EER may query from only increases the likelihood that EER falls prey to infinite looping.

5.3.3 Impact Sampling

We now introduce a new class of algorithms, impact sampling, which addresses the problems with uncertainty sampling and EER. Impact sampling algorithms pick the next example to (re)label that will impact the classifier the most, the intuition being that an example

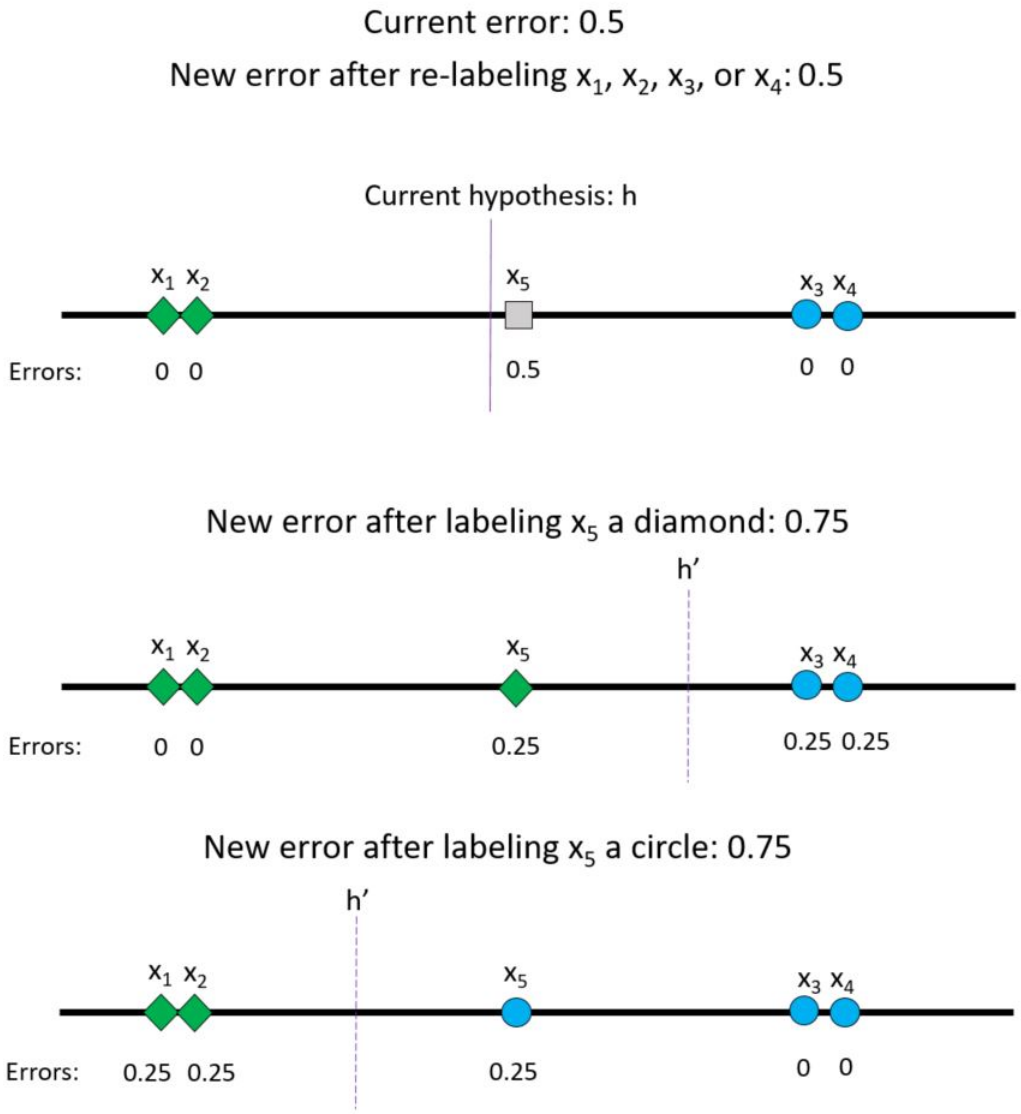

Figure 5.2: Suppose x_1, x_2, x_3, and x_4 have been labeled multiple times and have converged to the correct label. Suppose x_5 is the only unlabeled example. If we are learning a max-margin classifier, then h would be the current hypothesis. If x_1, x_2, x_3, and x_4 are far enough away from h so that $P_{\mathcal{A}}(h^*(x_1) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_2) = \text{diamond})$, $P_{\mathcal{A}}(h^*(x_3) = \text{circle})$, and $P_{\mathcal{A}}(h^*(x_4) = \text{circle})$ are equal to 1, and x_5 is close enough to h so that $P_{\mathcal{A}}(h^*(x_5) = \text{diamond}) = P_{\mathcal{A}}(h^*(x_5) = \text{circle}) = 0.5$, then the error of h is 0.5. Labeling one of x_1, x_2, x_3, or x_4 will not change the hypothesis, and so the expected error reduction is 0. However, labeling x_5 will result in a hypothesis that *increases* the expected error from 0.5 to 0.75! As a result, EER is likely to converge to behavior in which it will always relabel x_1, x_2, x_3, and x_4 forever.

that heavily impacts the learned classifier must be a good example to label. Let h_L be the hypothesis learned using learning algorithm $\mathcal{A}$, labels L, and aggregation function ζ. We define the *impact* of example x_i given label y, $\psi_y(x_i)$, as the probability that adding the additional label y for x_i changes the predicted label of an example in $\mathcal{X}$: $\psi_y(x_i) = P_{x\sim\mathcal{D}}(h_L(x) \neq h_{L\oplus\{(x_i,y)\}}(x))$.

Algorithm 5.1 describes the framework for computing the impact of an example x_i. First, it trains the classifier using the labeled data, producing a baseline hypothesis, h_L. Then, it trains one classifier supposing that it received a label 0 for example x_i, producing hypothesis h_{L0}, and another classifier supposing that it received the label 1 for example x_i, producing hypothesis h_{L1}. Next, it computes $\psi_0(x_i)$ and $\psi_1(x_i)$ by taking a sample from $\mathcal{X}$ and comparing the predictions of h_{L1} and h_{L0} against the predictions of h on that sample, and computing the fraction of predictions that changed for each. Alternatively, it can reason about how classifiers change with training. Finally, it returns some `weightedImpact`$(\psi_0(x_i), \psi_1(x_i))$, as the total impact of an example, $\psi(x_i)$. We construct variations of impact sampling by redefining $\oplus$ or implementing `weightedImpact` in various ways, which we now describe.

Optimism

The most straightforward way to implement `weightedImpact` is to use label posteriors to compute the total expected impact $\psi(x_i) = \sum_{y\in\mathcal{Y}} Q(y|x_i) \cdot \psi_y(x_i)$. We use `EXP` to denote impact sampling algorithms that compute an expected impact (e.g., `impactEXP`).

However, when training a classifier with noisy labels, the learned classifier often outputs beliefs that are wildly wrong. This can mislead the expected impact calculations and starve certain examples that could substantially improve the classifier, all because the classifier believes the impactful labels are unlikely. Furthermore, for most labeled examples in $\mathcal{X}_L$, at least one of ψ_0 and ψ_1 will be 0, biasing `impactEXP` to undervalue their impact over that of an unlabeled point. Computing the expectation also requires knowledge of label accuracy, which, like in EER, must be learned. To mitigate these we inject impact sampling with some optimism by implementing `weightedImpact` so that instead of returning the expected

Algorithm 5.1 Computation of Impact of an Example x_i

Input: Learning algorithm $\mathcal{A}$, Example $x_i \in \mathcal{X}$, Examples $\mathcal{X}$, aggregation function ζ, and label multiset L.

Output: Total impact of example x_i, $\psi(x_i)$

Initialize $\psi_0 = 0$, $\psi_1 = 0$.

$L1 = L \oplus \{(x_i, 1)\}$, $L0 = L \oplus \{(x_i, 0)\}$

$h_L = \texttt{retrain}(\mathcal{A}, \zeta, L)$

$h_{L1} = \texttt{retrain}(\mathcal{A}, \zeta, L1)$

$h_{L0} = \texttt{retrain}(\mathcal{A}, \zeta, L0)$

$\mathcal{X}_S = sample(\mathcal{X})$

for $x_j \in \mathcal{X}_S$ **do**

  if $h_{L0}(x_j) \neq h_L(x_j)$ **then**

    $\psi_0 = \psi_0 + \frac{1}{|\mathcal{X}_S|}$

  end if

  if $h_{L1}(x_j) \neq h_L(x_j)$ **then**

    $\psi_1 = \psi_1 + \frac{1}{|\mathcal{X}_S|}$

  end if

end for

$\psi = \texttt{weightedImpact}(\psi_0, \psi_1)$

Return ψ

impact, it instead returns the *maximum impact*: $\psi = \max(\psi_0, \psi_1)$. This leads the system to pick the example that could produce the largest possible impact on the classifier. We use `OPT` to denote impact sampling with optimism.

Pseudo-Lookahead

The most straightforward way to implement $\oplus$ is to simply add the new hypothetical labels for x_i into the multiset L_{x_i}. However, such an implementation makes the algorithm myopic, because for certain multisets, a single additional label may have no effect on the aggregated label, $\zeta(L_{x_i})$, and thus no effect on the learned classifier. For example, if ζ is majority vote and L_{x_i} currently has 2 positive votes and 0 negative votes, any single new annotation will have zero impact. Myopicity is problematic because correcting labeling mistakes may require gathering multiple labels for the same example. To alleviate this problem, we introduce the "pseudo-lookahead."

Whenever impact sampling is considering an example $x_i \in \mathcal{X}_L$ from the labeled set (the myopicity problem does not exist when considering a new unlabeled example), we implement the $\oplus$ operator so that instead of directly adding the new label l_i^{new} into the current multiset L_{x_i}, we ensure that the classifier is trained with l_i^{new} as the aggregated label, instead of $\zeta(L_{x_i})$. Let ρ be the minimum number of additional labels needed to flip the aggregate label to l_i^{new} (ρ is zero if l_i^{new} is already $\zeta(L_{x_i})$). We implement `weightedImpact` so that the computed impact of that label, $\psi_{l_i^{new}}$, is divided by $\max(1, \rho)$ before any additional computation: $\psi_{l_i^{new}} = \psi_{l_i^{new}} / \max(1, \rho)$. Intuitively, this pseudo-lookahead is computing a normalized impact of a single label, if impact sampling trains $\mathcal{A}$ with aggregate label l_i^{new} after receiving multiple such labels. We denote algorithms that use pseudo-lookahead with `PL`. We note that introducing pseudo-lookahead can theoretically cause impact sampling to starve certain examples of labels, but that we almost never see this behavior in our experimentation.

Implementation Issues

We can construct four different impact sampling algorithms with these different implementations of $\oplus$ and `weightedImpact`: `impactEXP`, `impactOPT`, `impactPLEXP` and `impactPLOPT`. In practice, optimism and pseudo-lookahead are methods for biasing towards more relabeling, so we implement them for computing impacts of labeled examples only.

Computing the impact of a single point can require retraining a classifier twice (much like EER), and thus picking a single point to (re)label can require up to $2|\mathcal{X}|$ retrainings. To speedup the implementation, we restrict impact sampling to choose only between 2 points instead of all of $\mathcal{X}$: the point recommended by $\texttt{US}_{\mathcal{X}_U}$, and the point recommended by $\texttt{US}_{\mathcal{X}_L}$ (uncertainty sampling applied to only $\mathcal{X}_L$.) We also try versions that choose among 7 points: the 2 points as before, and the 5 points returned by $\texttt{US}^{\alpha}_{\mathcal{X}}$ where $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. In the experiments, we indicate the number of points from which the strategy draws in parentheses.

Computing the impact also requires iteration over a sample of $\mathcal{X}$. Significant speedups may be achieved by sampling examples from regions that that are likely to have been affected by the additional hypothetical labels. One can keep track of such examples using k-d trees and geometric reasoning about the domain. We leave such an optimization for future work.

5.4 Behavior of Impact Sampling

We now formally verify that as long as the the way labels are aggregated is reasonable, `impactEXP` indeed solves the infinite looping problems that starve traditional active learning algorithms like $\texttt{US}_{\mathcal{X}}$ and EER when they are applied to re-active learning. We first define a notion of reasonableness to capture the idea that an aggregation function should become more confident of its predictions as more labels are received. Then, we show that given such an aggregation function, relabeling an example multiple times will cause the impact of that example to eventually eventually decrease to zero, and thus `impactEXP` will prefer to label examples that have been labeled fewer times.

Definition 5.2. Let $\mathcal{X}$ be a set of examples and ζ be an aggregation function. Let LP denote any labeling process that labels each example in $\mathcal{X}$ infinitely many times, and assume that each label is drawn independently according to the worker accuracy p. Let $L^1_{\mathcal{X}} \subset L^2_{\mathcal{X}} \subset \ldots$ denote an infinite sequence of label multisets created by LP at times $t = 1, 2, \ldots$. Note that each $L^t_{\mathcal{X}}$ is a random variable. We denote the aggregation function ζ as *convergent* if and only if there almost surely (with probability 1 over all possible sequences of label multisets) exists an infinite sequence $t_1 < t_2 < \ldots$ such that for all $L' \in \{L^{t_1}_{\mathcal{X}}, L^{t_2}_{\mathcal{X}}, \ldots\}$ and any $x_i \in \mathcal{X}$, adding a single additional label for x_i to L' does not change any of the aggregated labels for L'.

Intuitively, this definition says that if an aggregation function is convergent, then as one keeps adding labels, one can always keep finding cases when adding an additional label won't change any of the aggregated labels. Note that with this definition, common aggregation functions like EM-based algorithms and majority vote are convergent.

Theorem 5.3. *Let $\mathcal{X}$ be a set of examples and ζ the aggregation function. If (1) `impactEXP` breaks ties randomly such that each of the tied examples has a nonzero probability of being chosen, and (2) ζ is convergent, then almost surely (with probability 1 over all possible label multiset sequences constructed by `impactEXP`) there does not exist a time t such that after t, only a strict subset of $\mathcal{X}$ will be labeled by `impactEXP` and each example in that subset is labeled infinitely many times.*

Proof. Suppose for contradiction that there exists a time t such that for all $t' > t$, only examples from a strict subset of examples $\mathcal{X}' \subset \mathcal{X}$ are labeled by `impactEXP`, and each example in $\mathcal{X}'$ is labeled infinitely many times. Let $\overline{\mathcal{X}'} \subset \mathcal{X}$ denote the complement of $\mathcal{X}'$. By assumption that ζ is convergent, with probability 1 there exists an infinite sequence of times $T = \{t_1, t_2, \ldots\}$ at which adding any additional label for any example $x_j \in \mathcal{X}'$ to any of the label multisets $\{L^{t_1}_{\mathcal{X}'}, L^{t_2}_{\mathcal{X}'}, \ldots\}$ does not change any aggregated labels. Note that the impact of any example $x_j \in \mathcal{X}'$ is 0 at any time $t'' \in T$. There are two cases: 1) there exists some $x_k \in \overline{\mathcal{X}'}$ such that $\psi(x_k) > 0$, and 2) $\psi(x_k) = 0$ for all $x_k \in \overline{\mathcal{X}'}$.

In the first case, by assumption there exists some time $t''' \in T$ such that $t''' > t'$. Then at time t''', `impactEXP` will label some $x_k \in \overline{\mathcal{X}'}$, which is a contradiction.

For the second case, with probability 1, `impactEXP` will at some time in T pick some $x_k \in \overline{\mathcal{X}'}$ to label, which is a contradiction.

□

While impact sampling and uncertainty sampling are ostensibly optimizing for two different objectives, we now show that, surprisingly, impact sampling can be considered a generalization of uncertainty sampling to the re-active learning setting. Specifically, in some *noiseless* learning problems (where relabeling is unnecessary) `impactEXP` reduces to $\texttt{US}_{\mathcal{X}_\mathcal{U}}$. Yet when relabeling is allowed, `impactEXP` behaves quite differently than $\texttt{US}_{\mathcal{X}}$, the extension of $\texttt{US}_{\mathcal{X}_\mathcal{U}}$ that *can* relabel.

We present a formal proof that `impactEXP` reduces to $\texttt{US}_{\mathcal{X}_\mathcal{U}}$ for the following learning setting, which we denote $\mathcal{P}$: The goal is to learn a 1-D threshold t^* on $\mathcal{X} = [a, b], a, b \in \mathbb{R}, a < b$, where $\mathcal{D}$ is uniformly distributed on $\mathcal{X}$, and the data are linearly separable. We assume that workers are perfect, and thus no relabeling is necessary. We assume that that active learning methods are initialized with one positive example (class 1) and one negative example (class 1). We also assume that we are training a max-margin classifier that outputs probability predictions $P_\mathcal{A}(h^*(x_i) = 0)$ and $P_\mathcal{A}(h^*(x_i) = 1)$ that are symmetric and monotonic with respect to distance from the predicted threshold. Symmetric means if $t \in \mathcal{X}$ is the classifier's currently learned threshold, then for all $x_1, x_2 \in \mathcal{X}$ such that $x_1 \neq x_2$, $|x_1 - t| = |x_2 - t|$ if and only if $P_\mathcal{A}(h^*(x_1) = 1) = P_\mathcal{A}(h^*(x_2) = 0)$. Monotonic means either for all $x_1, x_2 \in \mathcal{X}$, $x_1 > x_2$ if and only if $P_\mathcal{A}(h^*(x_1) = 1) > P_\mathcal{A}(h^*(x_2) = 1)$, or for all $x_1, x_2 \in \mathcal{X}$, $x_1 > x_2$ if and only if $P_\mathcal{A}(h^*(x_1) = 1) < P_\mathcal{A}(h^*(x_2) = 1)$. Max-margin classifiers like SVMs can easily output symmetric and monotonic probability predictions (e.g., by normalizing the distance to the hyperplane). We note that this learning problem we consider is popular in the active learning literature [55, 56]. Finally, we assume that all active learning methods subsample from $\mathcal{X}$ first to get a finite pool of unlabeled examples, $\mathcal{X}_U$, and they subsample

so that no two examples are the same distance from the threshold t (so that uncertainty sampling does not need to tiebreak.)

For ease of notation in our proofs, we denote with shorthand $p_0(x_i) = P_{\mathcal{A}}(h^*(x_i) = 0)$ and $p_1(x_i) = P_{\mathcal{A}}(h^*(x_i) = 1)$. We also note that because we are considering a setting with no noise, the total expected impact of a point x_i is $\sum_{y \in \mathcal{Y}} p_y(x_i)\psi_y(x_i)$.

We now first prove the following lemma, which describes conditions under when the expected impact of an example will be greater than the expected impact of another example.

Lemma 5.4. *If*

1. $(\psi_1(x_i) - \psi_0(x_i)) > \frac{\psi_0(x_j) - \psi_0(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j)}{p_1(x_i)}$, *or*
2. $(\psi_0(x_i) - \psi_1(x_i)) > \frac{\psi_1(x_j) - \psi_1(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j)}{p_0(x_i)}$, *or*
3. $(\psi_0(x_i) - \psi_1(x_i)) > \frac{\psi_0(x_j) - \psi_1(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j)}{p_0(x_i)}$, *or*
4. $(\psi_1(x_i) - \psi_0(x_i)) > \frac{\psi_1(x_j) - \psi_0(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j)}{p_1(x_i)}$,

then, the total expected impact of x_i is larger than that of x_j: $\sum_{y \in \mathcal{Y}} p_y(x_i)\psi_y(x_i) > \sum_{y \in \mathcal{Y}} p_y(x_j)\psi_y(x_j)$.

Proof. For condition (1), we have that $\sum_{y \in \mathcal{Y}} p_y(x_i)\psi_y(x_i)$

$$
\begin{aligned}
&= p_0(x_i)\psi_0(x_i) + p_1(x_i)\psi_1(x_i) \\
&= p_1(x_i)(\psi_1(x_i) - \psi_0(x_i)) + \psi_0(x_i) \\
&> p_1(x_i)\frac{\psi_0(x_j) - \psi_0(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j)}{p_1(x_i)} + \psi_0(x_i) \\
&= \psi_0(x_j) - \psi_0(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j) + \psi_0(x_i) \\
&= \psi_0(x_j) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j) \\
&= \sum_{y \in \mathcal{Y}} p_y(x_j)\psi_y(x_j).
\end{aligned}
$$

For condition (2), we have that $\sum_{y\in\mathcal{Y}} p_y(x_i)\psi_y(x_i)$

$$
\begin{aligned}
&= p_0(x_i)\psi_0(x_i) + p_1(x_i)\psi_1(x_i) \\
&= p_0(x_i)(\psi_0(x_i) - \psi_1(x_i)) + \psi_1(x_i) \\
&> p_0(x_i)\frac{\psi_1(x_j) - \psi_1(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j)}{p_0(x_i)} + \psi_1(x_i) \\
&= \psi_1(x_j) - \psi_1(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j) + \psi_1(x_i) \\
&= \psi_1(x_j) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j) \\
&= \sum_{y\in\mathcal{Y}} p_y(x_j)\psi_y(x_j).
\end{aligned}
$$

For condition (3), we have that $\sum_{y\in\mathcal{Y}} p_y(x_i)\psi_y(x_i)$

$$
\begin{aligned}
&= p_0(x_i)\psi_0(x_i) + p_1(x_i)\psi_1(x_i) \\
&= p_0(x_i)(\psi_0(x_i) - \psi_1(x_i)) + \psi_1(x_i) \\
&> p_0(x_i)\frac{\psi_0(x_j) - \psi_1(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j)}{p_0(x_i)} + \psi_1(x_i) \\
&= \psi_0(x_j) - \psi_1(x_i) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j) + \psi_1(x_i) \\
&= \psi_0(x_j) + (\psi_1(x_j) - \psi_0(x_j))p_1(x_j) \\
&= \sum_{y\in\mathcal{Y}} p_y(x_j)\psi_y(x_j).
\end{aligned}
$$

For condition (4), we have that $\sum_{y\in\mathcal{Y}} p_y(x_i)\psi_y(x_i)$

$$
\begin{aligned}
&= p_0(x_i)\psi_0(x_i) + p_1(x_i)\psi_1(x_i) \\
&= p_1(x_i)(\psi_1(x_i) - \psi_0(x_i)) + \psi_0(x_i) \\
&> p_1(x_i)\frac{\psi_1(x_j) - \psi_0(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j)}{p_1(x_i)} + \psi_0(x_i) \\
&= \psi_1(x_j) - \psi_0(x_i) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j) + \psi_0(x_i) \\
&= \psi_1(x_j) + (\psi_0(x_j) - \psi_1(x_j))p_0(x_j) \\
&= \sum_{y\in\mathcal{Y}} p_y(x_j)\psi_y(x_j).
\end{aligned}
$$

□

Theorem 5.5. *In the noiseless learning problem $\mathcal{P}$ (with no relabeling), $\texttt{US}_{\mathcal{X}_U}$ is equivalent to* impactEXP.

Proof. We prove the theorem by showing that given any labeled set $\mathcal{X}_L$ that contains at least one positive and one negative example (we assume $\texttt{US}_{\mathcal{X}_U}$ and `impactEXP` are initialized with these), if $x_i \in \mathcal{X}_U$ is the example chosen by $\texttt{US}_{\mathcal{X}_U}$, x_i will also be the example chosen by `impactEXP`. Let $t \in \mathcal{X}$ be the currently learned threshold, $x_< = \max\{x \in \mathcal{X}_L : x < t\}$ denote the current greatest labeled example less than the threshold, and $x_> = \min\{x \in \mathcal{X}_L : x > t\}$ denote the current smallest labeled example greater than the threshold. Assume without loss of generality that the negative examples are less than the threshold and the positive examples are greater than the threshold. Let $x_i \in \mathcal{X}_U$ be the point chosen by uncertainty sampling. We show that for x_i and all other unlabeled points $x_j \in \mathcal{X}_U$, $x_j \neq x_i$, one of the conditions of Lemma 5.4 holds, thus showing that the expected impact of x_i is larger than the expected impact of all other points x_j, and thus impact sampling will pick x_i, just like uncertainty sampling. (Because the learning problem $\mathcal{P}$ is noiseless, we only need to consider the unlabeled points $\mathcal{X}_U$, since the impact of any labeled point is 0 and the impact of any unlabeled point is greater than 0. Thus `impactEXP` will never select an already labeled point.)

There are four cases: 1) $x_i \geq t$ and $x_j \geq t$, 2) $x_i \geq t$ and $x_j < t$, 3) $x_i < t$ and $x_j \geq t$, and 4) $x_i < t$ and $x_j < t$. We prove the first case when $x_i \geq t$ and $x_j \geq t$, and then describe afterward how the other three cases are also consequently proved.

In the first case, we first notice that $x_j > x_i$, because of two reasons. First, $x_j \neq x_i$ by the subsampling uniqueness assumption, and second, x_i was picked by uncertainty sampling, which picks the unlabeled example closest to the threshold (because the classifier is monotonic and symmetric), so if $x_j < x_i$, then x_i would not have been picked by uncertainty sampling. Now we define $d_{*_1,*_2}$ to be the proportion of points in $\mathcal{X}$ between points $*_1$ and $*_2$. More precisely, if $*_1 < *_2$, then

$$d_{*_1,*_2} = P_{x\sim\mathcal{D}}(x \in \{x : *_1 < x < *_2\}),$$

and otherwise, if $*_2 \leq *_1$, then

$$d_{*_1,*_2} = P_{x\sim\mathcal{D}}(x \in \{x : *_2 < x < *_1\}),$$

For example, $d_{x_<,t}$ is the proportion of points between $x_<$ and t, and $d_{t,x_j} \geq d_{t,x_i}$ because $x_j > x_i$.

Now we show that Condition 1 of Lemma 5.4 is satisfied, that $(\psi_1(x_i) - \psi_0(x_i)) > \frac{\psi_0(x_j)-\psi_0(x_i)+(\psi_1(x_j)-\psi_0(x_j))p_1(x_j)}{p_1(x_i)}$ for all $x_j > x_i$. We have that for all x_j, $\psi_0(x_j) = d_{t,x_j} + \frac{d_{x_j,x_>}}{2}$, because labeling x_j as a negative example will cause the threshold t to move from its current position to halfway between x_j and $x_>$ (because the classifier is max-margin and the data are linearly separable). Similarly, $\psi_1(x_j) = d_{x_<,x_j} - (\frac{d_{x_<,x_j}}{2} + d_{t,x_j})$, because labeling x_j as a positive example will cause the threshold to move to halfway between x_j and $x_<$. Therefore, $\psi_1(x_j) - \psi_0(x_j)$

$$\begin{aligned}
&= d_{x_<,x_j} - (\frac{d_{x_<,x_j}}{2} + d_{t,x_j})) - (d_{t,x_j} + \frac{d_{x_j,x_>}}{2}) \\
&= d_{x_<,t} - \frac{d_{x_<,x_j}}{2} - \frac{d_{x_j,x_>}}{2} - d_{t,x_j} \\
&= d_{x_<,t} - d_{x_<,t} - d_{t,x_j} \\
&= -d_{t,x_j}.
\end{aligned}$$

Next, we have that $\frac{\psi_0(x_j)-\psi_0(x_i)+(\psi_1(x_j)-\psi_0(x_j))p_1(x_j)}{p_1(x_i)}$

$$\begin{aligned}
&= \frac{d_{t,x_j} + \frac{d_{x_j,x_>}}{2} - (d_{t,x_i} + \frac{d_{x_i,x_>}}{2}) - d_{t,x_j}p_1(x_j)}{p_1(x_i)} \\
&= \frac{d_{x_i,x_j} - 0.5(d_{x_i,x_j}) - d_{t,x_j}p_1(x_j)}{p_1(x_i)} \\
&= \frac{0.5d_{x_i,x_j} - d_{t,x_j}p_1(x_j)}{p_1(x_i)}.
\end{aligned}$$

And then,

$$\begin{aligned}
\frac{0.5d_{x_i,x_j} - d_{t,x_j}p_1(x_j)}{p_1(x_i)} &< -d_{t,x_i} = (\psi_1(x_i) - \psi_0(x_i)) \\
&\Updownarrow \\
0.5d_{x_i,x_j} - d_{t,x_j}p_1(x_j) &< -d_{t,x_i}p_1(x_i) \\
&\Updownarrow \\
0.5d_{x_i,x_j} &< d_{t,x_j}p_1(x_j) - d_{t,x_i}p_1(x_i) \\
&\Updownarrow \\
0.5d_{x_i,x_j} &< d_{t,x_j}[p_1(x_i) + \beta] - d_{t,x_i}p_1(x_i), \quad \beta = p_1(x_j) - p_1(x_i) \\
&\Updownarrow \\
0.5d_{x_i,x_j} &< p_1(x_i)d_{x_i,x_j} + \beta d_{t,x_j}, \quad \beta = p_1(x_j) - p_1(x_i)
\end{aligned}$$

$\beta > 0$ because $x_j > x_i$, and $p_1(x_i) > 0.5$ because $x_i > t$, and therefore $0.5d_{x_i,x_j} < p_1(x_i)d_{x_i,x_j} + \beta d_{t,x_j}$, and the first case is proved.

The three other cases: 2) $x_i \geq t$ and $x_j < t$, 3) $x_i < t$ and $x_j \geq t$, and 4) $x_i < t$ and $x_j < t$, are proved after proving the first case by the following symmetry and change of variables argument. First, observe that if $x_1 \in \mathcal{X}$ is a reflection of $x_2 \in \mathcal{X}$ across the threshold t, in other words if $|x_1 - t| = |x_2 - t|$ and $x_1 \neq x_2$, then $\psi_1(x_1) = \psi_0(x_2)$ and $\psi_1(x_2) = \psi_0(x_1)$.

For case 2, let z be the reflection of x_j across the threshold t. Then $z > t$, and we have shown that for $x_i \geq t$ and $z \geq t$, Condition 1 of Lemma 5.4 holds. Now note that Condition 4 of Lemma 5.4 is equivalent to Condition 1 by replacing $\psi_0(z)$ in Condition 1 with $\psi_1(x_j)$, $\psi_1(z)$ with $\psi_0(x_j)$, and $p_1(z)$ with $p_0(x_j)$ (by classifier's symmetry). Thus, Condition 4 holds for $x_i \geq t$ and $x_j < t$, and the second case is proved. In the same manner, we can show that Condition 2 of Lemma 5.4 holds for case 3, and Condition 3 of Lemma 5.4 holds for case 4, so the theorem is proven.

□

5.5 *Experiments*

We now present empirical experiments on both synthetic and real-world datasets to compare the performances of $\texttt{US}^{\alpha}_{\mathcal{X}}$, $\texttt{US}^{j/k}_{\mathcal{X}_U}$, and impact sampling. As baselines, we include EER, $\texttt{US}_{\mathcal{X}_U}$ (uncertainty sampling *without* relabeling), and passive learning (random selection).

We begin with a synthetic domain containing two random Gaussian clusters, which correspond to two classes, and train using L2-regularized logistic regression. We generate a dataset by randomly picking two means, $\mu_1, \mu_2 \in [0,1]^z$, and two corresponding covariance matrices $\Sigma_1, \Sigma_2 \in [0,1]^{z \times z}$. We ensure a pool of 1,000 examples for each Gaussian cluster (class) exists at every timestep, in order to simulate an infinite pool of examples from which the algorithms may choose. All experiments are averaged over 250 random datasets (all figures show faded 95% confidence intervals, most of which are extremely small). We vary the number of features among $z \in \{10, 30, 50, 70, 90\}$. We seed training with 50 examples, use a total budget of 1,000, and test on 300 held-out examples. Note that our budget is much larger than what is common in the active learning literature, with many works experimenting with budgets 10-40 times smaller than ours (e.g., [57, 80]). We use a larger budget for two reasons. First, to make our experiments more realistic than prior work, and second, to account for slower convergence due to noise. We assume each label is independently flipped from the true label, $h^*(x)$, with probability 0.25 ($p = 0.75$) and use majority vote for ζ, the label aggregation function. For our experiments on synthetic data, we assume that the label accuracy p is known, but later relax this assumption in our experiments with real data.

Figure 5.3 shows the performance of several strategies when setting the number of features $z = 90$. Figure 5.3(a) compares $\texttt{US}^{\alpha}_{\mathcal{X}}$ with $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We find that $\alpha = 0.1$ does the best, and that as we increase α from 0.1, the performance of the learned classifier drops, though not drastically. Figure 5.3(b) compares $\texttt{US}^{j/k}_{\mathcal{X}_U}$ with $j/k \in \{1/1, 2/3, 3/5, 4/7, 5/9\}$. We see that performance improves as j/k increases from 1/1 to 3/5 but then decreases as we continue to increase the amount of relabeling redundancy. Different settings of α and j/k work better depending on the number of features, and

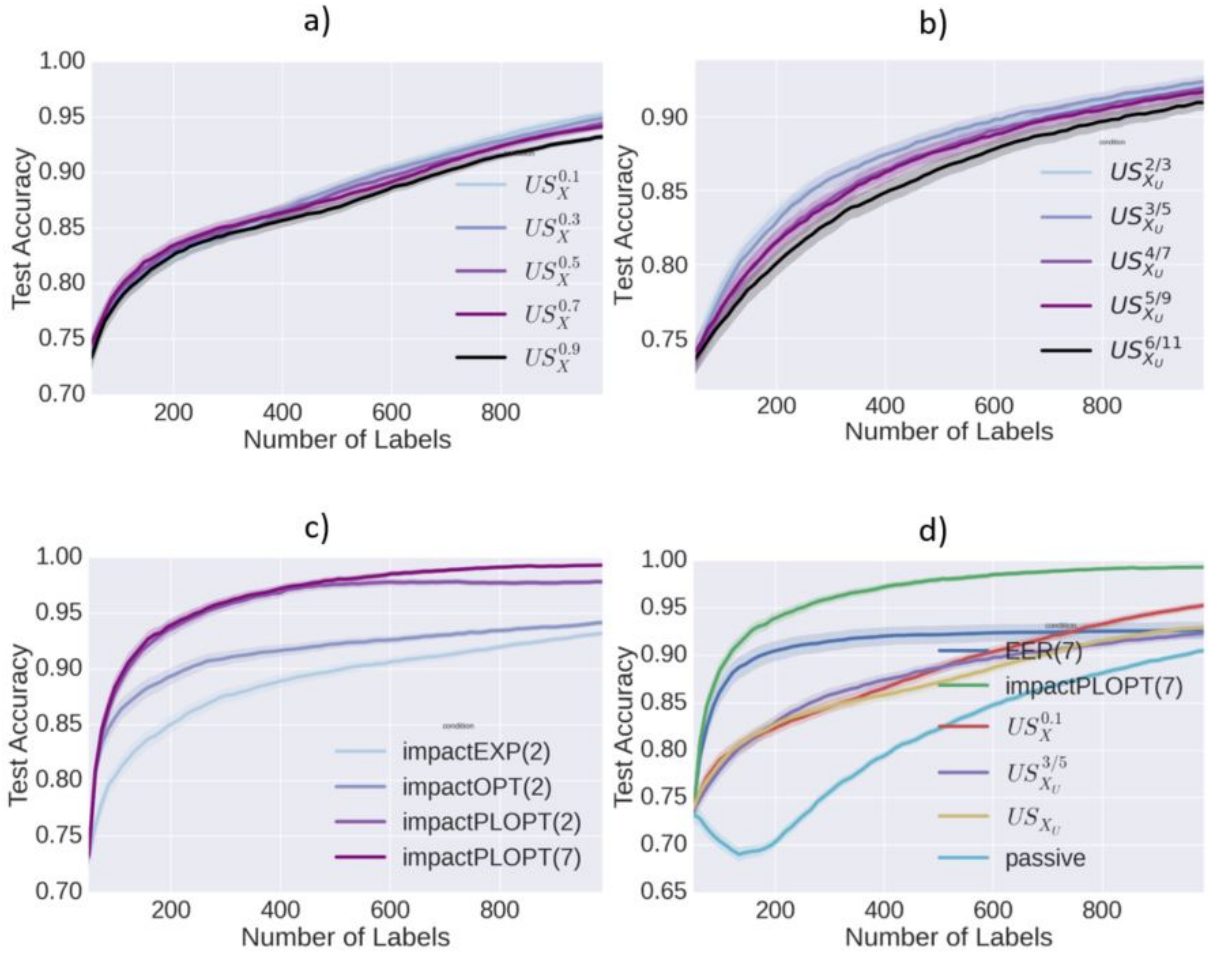

Figure 5.3: Average generalization accuracy of logistic regression over randomly generated Gaussian datasets with 90 features and label accuracy 0.75, when trained using various strategies. a) compares various settings of $\texttt{US}_{\mathcal{X}}^{\alpha}$ and shows $\alpha = 0.1$ is best. b) compares various settings of $\texttt{US}_{\mathcal{X}_U}^{j/k}$ and shows that $j/k = 3/5$ is best. c) compares various impact sampling strategies and shows `impactPLOPT(7)` is best. d) compares impact sampling, uncertainty sampling, and EER, and shows that impact sampling produces the best results.

the best hyperparameter for a given number of features is difficult to predict.

Figure 5.3(c) shows the effect of introducing optimism and pseudo-lookahead into impact sampling. We notice that these methods are able to substantially increase performance. `impactOPT(2)` outperforms `impactEXP(2)`, and `impactPLOPT(2)` performs significantly better than `impactOPT(2)`. We also see that allowing impact sampling to choose from a larger pool of candidate examples (`impactPLOPT(7)`) improves performance over the 2-example version (`impactPLOPT(2)`). Henceforth, in subsequent figures we show only the performance of our most robust impact sampling algorithm, `impactPLOPT(7)`.

Figure 5.3(d) compares impact sampling to uncertainty sampling, EER, and passive learning. We first see that as expected, passive learning results in the worst performance. Next, we observe that `impactPLOPT(7)` significantly outperforms all other methods (p-value< 0.001 using a Welch's t-test on the average accuracies at the maximum budget). Although not shown, we find that even `impactEXP(2)`, the weakest impact sampling strategy, strictly dominates vanilla $\texttt{US}_{\mathcal{X}_U}$.

Figure 5.4 compares impact sampling against uncertainty sampling and passive learning on datasets from the UCI Machine Learning Repository [13] with synthetically-generated labels. We use a budget that is equal to half of the size of the dataset, and randomly generate a training set of 70% of the examples, a held-out set of 15% of the examples, and a test set of the remaining 15% of the examples. We again present results averaged over 250 of these randomizations and display 95% confidence intervals. In both datasets, by the time the budget is exhausted, `impactPLOPT(7)` is substantially and statistically significantly better than $\texttt{US}_{\mathcal{X}_U}$ (p-value< 0.001), which is also substantially and statistically significantly better than passive learning (p-value< 0.001).

Finally, we investigate how well our methods work on real-world datasets with real human-generated labels. In the following experiments, we use F-score as the metric because of high skew. Whenever a strategy requests a label, we sample from the set of available annotations. Unlike the previous experiments, we modify `impactPLOPT(7)` so that it does *not* assume knowledge of label accuracy p. To make this relaxation, first, because computing expected

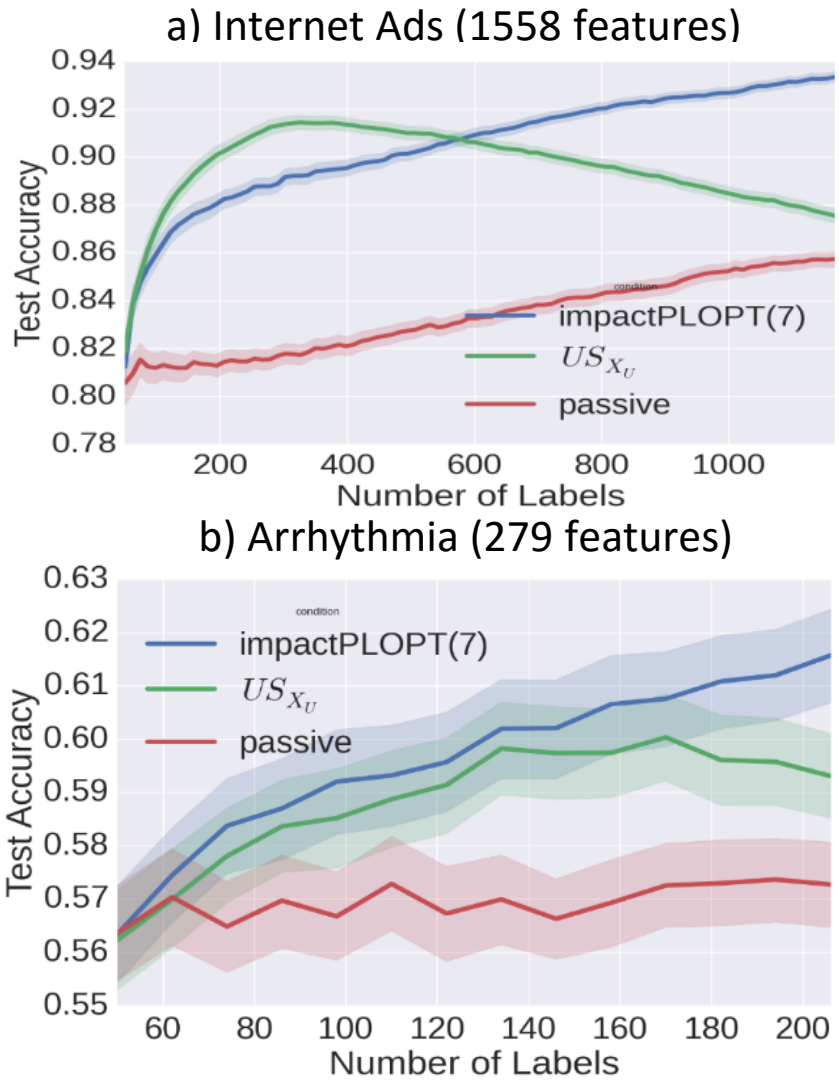

Figure 5.4: Comparison of impact sampling versus uncertainty sampling on real-world datasets, Internet Ads and Arrhythmia, using simulated labels modeling annotators whose accuracy is 0.75. Impact sampling shows significant gains.

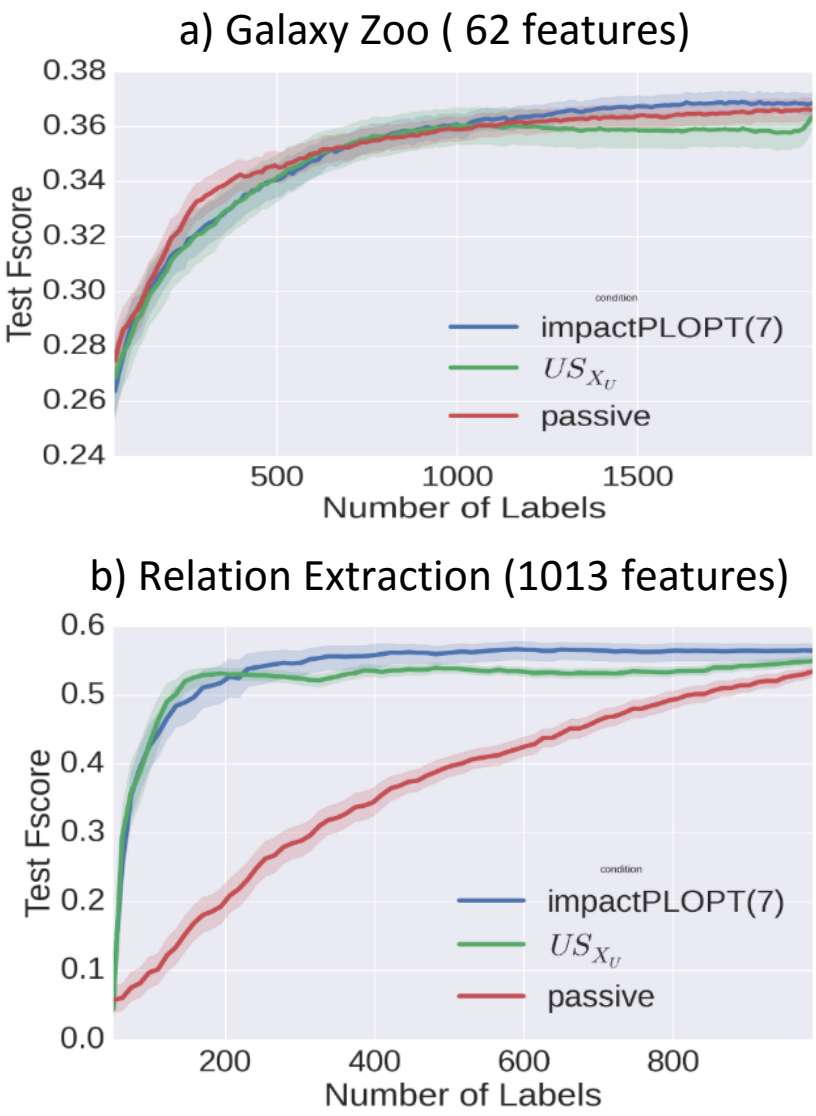

Figure 5.5: Impact sampling performs well against uncertainty sampling on the real-world datasets with real human-generated labels.

impact requires knowledge of p, we always apply optimism to all examples that we consider, instead of only when examples will be relabeled. Second, because $\texttt{US}^{\alpha}_{\mathcal{X}}$ requires knowledge of p, instead of including 5 points returned by $\texttt{US}^{\alpha}_{\mathcal{X}}$ among the subset of 7 from which impact sampling chooses, we include 5 random points.

We first consider a popular dataset [104, 138, 140] from Galaxy Zoo, a website on which volunteer workers contribute annotations for images of galaxies gathered by the Sloan Digital Sky Survey. Each image has 62 features obtained from machine vision analysis, and workers are asked to label each galaxy as "elliptical," "spiral," or "other." Each galaxy is labeled by at least 10 workers. We use a small version of the dataset containing 3937 galaxies, and train a logistic regression classifier to predict whether a galaxy is elliptical. About 12.8% of galaxies are in the positive class. Figure 5.5a shows that the differences between `impactPLOPT(7)`, $\texttt{US}_{\mathcal{X}_U}$, and passive learning are not statistically significant. However, the fact that $\texttt{US}_{\mathcal{X}_U}$ is unable to improve upon passive learning suggests that any method for intelligent re-active learning is unlikely to be of much benefit on this particular dataset, so our result, which shows `impactPLOPT(7)` matching the performance of the two baselines, is evidence for the robustness of `impactPLOPT(7)`.

Next, we consider a dataset gathered for the task of relation extraction. Given a sentence, e.g., "Barack Obama was born in Hawaii," two entities in that sentence, "Barack Obama" and "Hawaii," and a relation, "*born-in*," the goal is to determine whether the given sentence expresses the given relation between the two entities. In this example, the sentence does express the relation that Barack Obama was born in Hawaii. The dataset contains 2000 sentences, each labeled by 10 Amazon Mechanical Turk workers with at least a 95% approval rating. Each example sentence is described by 1013 features. We train a logistic regression classifier to predict whether an example expresses the "*born-in*" relation. Positive examples comprise about 7.4% of the dataset. Figure 5.5b shows that `impactPLOPT(7)` makes significant gains (p-value< 0.01) against the baselines.

5.6 Discussion

Our experiments show that impact sampling can train better classifiers than previous active learning algorithms *given the same labeling budget*. However, this assessment does not consider the cost of computational resources or time. One drawback of our method, shared with active learning techniques such as expected error reduction [182, 108], is the high computational cost of picking the next data point to (re)label. This computational burden stems from impact sampling's need to retrain a classifier twice per candidate data point. In some cases this computational load may be prohibitive, especially if the training cost of the classifier is high, such as for multi-layer perceptrons or non-linear SVMs or in problems with high dimensional feature spaces. While one can speed up impact sampling by restricting its example choices to a small subset of $\mathcal{X}$ instead of all of $\mathcal{X}$ (as in our implementation), or by using classifiers that have mechanisms for incremental training (e.g., stochastic gradient descent initialized with the previous solution or incremental decision tree induction), whether or not impact sampling is truly useful in practice is debatable and may depend on the complexity of the learning problem.

For example, in our final experiment on relation extraction, impact sampling takes over 7 hours[1] to choose the set of 1000 examples to (re)label. In other words, impact sampling takes, on average, about 26 seconds to compute the best example to send to a human, who would likely take about the same time to label the example. In contrast, uncertainty sampling and random sampling are much faster, taking approximately 210 and 70 seconds, respectively, to select their 1000 examples. If one is faced with a time constraint, we suggest using $\texttt{US}_{\mathcal{X}_U}$, uncertainty sampling with no relabeling, which is a fast and effective algorithm.

Finally, we note that through our experiments, we have shown the robustness of impact sampling, but like with all active learning techniques, there exist datasets in which our methods do not perform as well. For example, when training logistic regression classifiers

[1]Recall that this domain has 1013 features. Experiments are programmed in Python using an Intel Xeon E7-4850-v2 processor (2.3 GHz, 24M Cache) with 512 GB of RAM.

to learn a Twitter sentiment dataset [78, 159, 199] with 1000 features and 1540 examples using real crowd labels collected by Goel *et al.* [79], we find that $\texttt{US}_{\mathcal{X}_U}$ achieves an average accuracy about 3% higher than `impactPLOPT(7)` after a budget of 1000 labels. However, we hypothesize that the poor performance is due to the difficulty of the task. First, the task of classifying the sentiment of a tweet is quite difficult, with many sentences expressing neither positive nor negative sentiment, and thus worker labels are likely to be very noisy. Second, the gold labels themselves are noisy, because they are the output of a distantly supervised machine learning algorithm. Further work is necessary to understand the properties of learning problems that affect the performance of re-active learning algorithms.

5.7 Related Work

Prior work [198, 92] identifies the tradeoff between relabeling examples and gathering new examples. They show that relabeling examples is useful for supervised learning, and they also consider how to pick examples to relabel when relabeling is the only option. However, crucially, unlike our work, they do not consider any combination of relabeling and the labeling of new examples.

Kääriäinen [100] shows that if each label is independently corrupted by noise, then active learning with noise is theoretically almost as easy as active learning without noise, because one can repeatedly label each point a sufficient number of times to cancel out the noise. In contrast, we show that the small theoretical difference can be quite large in practice, and thus reasoning about whether or not to relabel is extremely important.

The active learning algorithm of Wauthier et al. [236] can potentially trade off between relabeling and acquiring labels for new examples. However, they do not identify the tradeoff explicitly, nor explore it analytically or experimentally. Furthermore, their solution requires gold labels, and is tied to their own custom classifier.

Several researchers consider how to pick workers for either labeling [66, 246] or relabeling [59]. Unlike our work, none of these works answers the fundamental question of *when* to relabel, or whether or not relabeling is necessary. Researchers have also considered how to

decide when to relabel when the objective is data accuracy [51, 134, 133, 35, 105] or ranking [43] instead of the accuracy of a classifier learned using the data.

Agnostic Active Learning [113, 15, 80] is a general learning setting for active learning with noise. However, like the rest of the active learning literature, it does not consider relabeling.

Zhang et al. [249] consider an active learning setting where instead of relabeling, one can choose between querying an expert labeler who is more expensive or a noisy labeler who is cheaper. They learn a classifier that predicts when these labelers differ in their labels and only ask the expert when they predict disagreement.

On-the-job Learning [241] is an extension of online active learning that allows the learning algorithm to query the crowd (if doing so is likely to help) prior to making a prediction. In contrast, we focus on how to train the best classifier possible offline, and thus relabeling examples may not always be necessary.

Finally, impact sampling can be considered a generalization of selection by expected gradient length [191], which queries the example that would create the greatest change in the gradient of the objective function, but impact sampling does not require knowledge of the classifier's objective function.

5.8 Conclusion

This chapter tackles the problem of re-active learning, a generalization of active learning that explores the tradeoff between decreasing the noise of the training set via relabeling and increasing the size of the (noisier) training set by labeling new examples. We introduce two re-active learning algorithms: an extension of uncertainty sampling, and a novel of class of impact sampling algorithms. We characterize their theoretical properties and investigate interrelationships among these algorithms. We find that impact sampling is equivalent to uncertainty sampling for some noiseless cases, but provides significant benefit for the case of noisy annotations. We empirically demonstrate the effectiveness of impact sampling on synthetic domains, real-world domains with synthetic labels, and real-world domains with real human-generated labels.

Chapter 6

SKEW-ROBUST LEARNING POWERED BY THE CROWD

6.1 Introduction

In the last two chapters, we considered the tradeoff that machine learning practitioners must make between noise and quantity when gathering training data. In this chapter, we look at how to maintain class balance in training sets, which is especially important in *high-skew* environments, environments with extreme class imbalance. We leave the noisy environment of the last four chapters and assume that workers are perfect in order to make progress on this problem.

In high-skew environments, which are ubiquitous in real-world supervised machine learning [172, 167], traditional strategies for obtaining labeled training examples perform poorly. Tasking crowd workers with labeling randomly-selected or even intelligently-selected examples (*e.g.* via active learning) is ineffective because the probability that any given example belongs to the minority-class is virtually zero [11]. Furthermore, labeling examples using heuristics like distant supervision [49, 156] or data programming [89, 71] is not always applicable, because most concepts do not exist in any knowledge base, and a trained predictor does not exist when starting from scratch.

To address this problem, Attenberg *et al.* [11] propose *guided learning*, a class of strategies for obtaining labeled training examples that ask crowd workers to *find*, as opposed to just *label*, training examples. Because of the increased human effort, guided learning is more expensive per example. However, they show that in domains with high skew, the added cost translates to more effective initial learning, as guided learning can quickly obtain minority-class examples while other strategies flounder, looking for a needle in a haystack.

Of course, guided learning isn't appropriate in all situations. In balanced domains, crowd-

labeling of randomly-selected examples will outperform guided learning, which will be far more costly than necessary. And even in high-skew domains, Attenberg *et al.* show that switching to crowd-labeling of actively-selected examples partway through training can result in better performance, after a classifier has been sufficiently bootstrapped through guided learning.

We observe that the key differences between these various strategies are their costs and the distributions of data that they obtain, and that ultimately, the best strategy at any given time may differ depending on a number of factors, including the skew in the domain and the progress in training. Therefore, for optimal learning, solving the following decision problem is crucial: suppose you have a set of strategies for obtaining labeled training examples, or *example-acquisition primitives* (EAPs) (*e.g.* generate minority-class example, label examples selected using uncertainty sampling [126]). Given a budget, an unlabeled corpus, a classifier previously trained on N examples (where N could be 0), and a labeled training set (possibly empty), which EAP should you use next to obtain another labeled example in order to maximize performance of the classifier at the end of training? In our exploration of this problem, we make the following contributions:

- We list existing EAPs, propose several novel ones, and consider their various properties.

- We develop two online algorithms that adapt multi-armed bandit methods [12, 117] to dynamically choose EAPs based on evolving estimates of how cheaply they can obtain minority-class examples.

- In experiments with both real and synthetic data, we first investigate the value of several EAPs for training classifiers. Then we compare and contrast the behavior of our algorithms in multiple skew settings, and we show that our best-performing algorithm can yield up to a 14.3 point gain on average in F1 AUC compared to the state-of-the-art baseline.

In the rest of this chapter, we assume the binary classification setting and that the positive class is the minority class.

6.2 Example-Acquisition Primitives

We begin by listing existing example-acquisition primitives (EAPs) and proposing several novel ones. We categorize the primitives into three types, Generation Primitives, Labeling Primitives, and Machine Primitives. Generation and Labeling Primitives rely on crowd work, while Machine Primitives can be executed without any human involvement. In general, Generation Primitives ask the crowd to generate or find examples, and are therefore more costly, but can retrieve examples of a certain class on-demand. Labeling Primitives only ask the crowd to label examples, and are therefore cheaper, but offer less control over the examples that are obtained.

6.2.1 Generation Primitives

- Crowd-Gen+ Task a crowd worker with generating/finding a positive example. This primitive is a quintessential guided learning strategy [11], and should improve recall.

- Crowd-Modify+To- Task a crowd worker with minimally modifying a positive example to turn it into a negative example. A training set with many examples that are very similar to each other yet have different labels can help a classifier learn important features and adding negative examples that are nearly positive should increase precision.

- Crowd-Modify-To+ Task a crowd worker with minimally modifying a negative example to turn it into a positive example. In contrast to Crowd-Modify+To-, adding positive examples that are nearly negative should increase recall.

6.2.2 Labeling Primitives

- Crowd-Label-Active Ask a crowd worker to label an example selected using an active learning technique (*e.g.*, uncertainty sampling [126]).

- Crowd-Label-Random Sample a random example from the unlabeled corpus and ask a crowd worker to label it. While Crowd-Label-Active should on average do better, randomly-selected examples may still be helpful in reducing active learning bias and understanding "unknown unknowns," the examples that a classifier is confidently incorrect about [123].

- Crowd-Label-PredPos Ask a crowd worker to label an example that the current classifier predicts to be positive. Using the current classifier to find positive examples is another method for sifting through a high-skew environment. This primitive should improve precision because it can identify examples that have been mistakenly predicted as positive.

6.2.3 Machine Primitives

- Insert-Random-Negative Pick a random example from the unlabeled corpus and insert it into the training set as a negative example. This primitive does not require any crowd work. In a high-skew setting, this primitive provides free and relatively clean negative examples. However, in a low-skew setting, many positive examples may be inserted erroneously into the training set as negative examples.

We note that many more possible EAPs may exist. We provide above a sampling of interesting and potentially useful EAPs to show the potential of our problem setting.

6.3 Algorithms

We now introduce two new algorithms for the online selection of EAPs: `MB-CB` (Make Balanced - Cost Bound) and `MB-T` (Make Balanced - Thompson), which can dynamically make

decisions based on observations they make.

Before describing the algorithms, we first observe that a large factor in which EAP will work well is how cheaply they can obtain positive examples. Labeling Primitives work well in low-skew settings because they can obtain positive examples at low cost. In contrast, Generation Primitives work well in high-skew domains because obtaining positive examples via Labeling Primitives can be extremely expensive.

`MB-CB` and `MB-T` work by exploiting this intuition. For every EAP, they compute how expensive obtaining a single positive example using that primitive would be, and then choose the primitive that is the cheapest by this metric. For example, suppose that Crowd-Label-Active costs \$0.03 per example and Crowd-Gen+ costs \$0.15 per example. In a domain in which 999 out of 1000 examples are negative, any algorithm would, at least *initially*, need to execute Crowd-Label-Active 1000 times in order to obtain one positive example in expectation. In this case, `MB-CB` and `MB-T` would compute that a positive example from Crowd-Label-Active costs \$30, a positive example from Crowd-Gen+ costs \$0.15, and subsequently execute Crowd-Gen+. However, after the classifier has been sufficiently trained, Crowd-Label-Active may obtain positive examples 50% of the times it is executed. Then, a positive example from Crowd-Label-Active would only cost \$0.06, which is much cheaper than the \$0.15 it takes to obtain a positive examples from Crowd-Gen+. At this point, `MB-CB` and `MB-T` would execute Crowd-Label-Active.

Unfortunately in many cases, the expected cost of obtaining a single positive from an EAP is unknown and must be learned, and the only way to learn these costs is to execute the primitives. We observe that this problem can be modeled as a multi-armed bandit problem, in which arms correspond to EAPs, and the reward of each arm is the negative expected cost of obtaining a single positive example from each EAP. Any control algorithm that tries to solve this problem must make a tradeoff between exploiting the knowledge they currently have (by executing the primitive they believe is cheapest), and exploring the unknown (by executing primitives in order to learn more about their costs, which may be nonstationary).

`MB-CB` manages this tradeoff by adapting UCB/UCT [12, 117] from the multi-armed

bandit literature. It maintains a lower bound on the cost of a single positive example for every primitive. Each lower bound is computed using an exploitation term and an exploration term. The exploitation term is determined using the history of costs from the respective primitive and the exploration term is determined based on the number of times the primitive has been executed. As a primitive is executed more, its corresponding exploration term decreases. An exploration constant c_e can be set to change the importance of the exploration term relative to the exploitation term in the lower bound.

At each timestep, `MB-CB` selects the EAP with the lowest bound to execute next. And every time `MB-CB` executes an EAP and receives an observation about the cost of a positive example from using that EAP, it updates the lower bound for that primitive. Algorithm 6.1 presents pseudocode for `MB-CB`.

`MB-T` manages the explore/exploit tradeoff by using Thompson Sampling [219, 42]. It uses Beta distributions to represent its belief about the probability that a primitive will return a positive example. Then, it samples from its belief, executes the primitive that has the lowest cost according to the sample, and updates its belief after receiving an observation. Algorithm 6.2 presents pseudocode for `MB-T`. We note that `MB-T` is very similar to `MB-CB`; it simply maintains its beliefs about costs in a more Bayesian manner.

6.4 Experiments

We now present a series of experiments with both real and synthetic data to answer three questions. The first two questions investigate the value of various EAPs, and the last question investigates the effectiveness of our algorithms at selecting EAPs.

1. How cost-effective is gathering near-miss negative examples (Crowd-Modify+To-) compared to negatively-labeling randomly sampled examples from an unlabeled corpus (Insert-Random-Negative)?

2. How does labeling examples that are predicted as positive (Crowd-Label-PredPos) compare against labeling examples selected using Active Learning (Crowd-Label-Active)?

Algorithm 6.1 `MB-CB`

Input: EAPs $\mathcal{V}$, budget b, exploration constant c_e.

Initialize $costSoFar = 0$

$p_c = \{\}$ //Track estimated cost of positive per primitive

$p_\alpha = \{\}$ // Track # positives obtained per primitive

$p_\beta = \{\}$ // Track # negatives obtained per primitive

$p_n = \{\}$ //Track number of times each primitive is called

for $v \in \mathcal{V}$ **do**

$p_c[v] = 0$, $p_n[v] = 0$, $p_\alpha[v] = 0$, $p_\beta[v] = 0$

end for

while $costSoFar < b$ **do**

$bestAction = None$, $bestCost = \infty$

for $v \in \mathcal{V}$ **do**

$explorationTerm = \sqrt{c_e \frac{\log \sum_{v \in \mathcal{V}} p_n[v]}{p_n[v]}}$

$cost = p_c[v] - explorationTerm$

if $cost < bestCost$ **then**

$bestAction = v$

$bestCost = cost$

end if

end for

Execute $bestAction$ and track $numPositives$ and $numNegatives$ obtained.

$p_\alpha[bestAction] = p_\alpha[bestAction] + numPositives$

$p_\beta[bestAction] = p_\beta[bestAction] + numNegatives$

$expectedNumPositives = (numPositives + numNegatives) \cdot \frac{p_\alpha[bestAction]}{p_\alpha[bestAction] + p_\beta[bestAction]}$

$p_c[bestAction] = \frac{bestAction.cost}{expectedNumPositives}$

$costSoFar = costSoFar + bestAction.cost$

$p_n[bestAction] = p_n[bestAction] + 1$

end while

Algorithm 6.2 `MB-T`

Input: EAPs $\mathcal{V}$, budget b

Initialize $costSoFar = 0$

$p_\alpha = \{\}$ //Track # positives obtained per primitive

$p_\beta = \{\}$ //Track # negatives obtained per primitive

for $v \in \mathcal{V}$ **do**

  $p_\alpha[v] = 1$ //Set uniform priors

  $p_\beta[v] = 1$

end for

while $costSoFar < b$ **do**

  $bestAction = None$

  $bestCost = \infty$

  for $v \in \mathcal{V}$ **do**

    Sample $s \sim Beta(p_\alpha[v], p_\beta[v])$

    $cost = \frac{bestAction.cost}{v.batchSize \cdot s}$ //v.batchSize is the # of examples v will obtain

    if $cost < bestCost$ **then**

      $bestAction = v$

      $bestCost = cost$

    end if

  end for

  Execute $bestAction$ and track $numPositives$ and $numNegatives$ obtained.

  $p_\alpha[bestAction] = p_\alpha[bestAction] + numPositives$

  $p_\beta[bestAction] = p_\beta[bestAction] + numNegatives$

  $costSoFar = costSoFar + bestAction.cost$

end while

Does skew affect which strategy performs better? How many positive examples are labeled in each case?

3. Are `MB-T` and `MB-CB` able to achieve better performance than state-of-the-art baselines in various domains across varying skews? Are these algorithms able to adapt to the skew and make intelligent decisions? How do they compare against each other?

6.4.1 The value of Crowd-Modify+To-

In order to investigate whether or not Crowd-Modify+To- is more cost-effective than Insert-Random-Negative, we test whether classifiers trained using negative examples from Crowd-Modify+To- achieve better performance than classifiers trained using negative examples from Insert-Random-Negative under a fixed budget.

We compare two strategies for spending this budget. Both strategies use Crowd-Gen+ to obtain positive examples. To obtain negative examples, one strategy uses Insert-Random-Negative. We denote this strategy `RandomNegs`. The other strategy uses Crowd-Modify+To-, and we denote this strategy `ModifyNegs`.

To do the comparison, we use the task of relation extraction, which is the task of determining whether a natural language sentence expresses a given relation between two given entities. We train classifiers to identify five different relations: "*Born in*," "*Died in*," "*Traveled to*," "*Lived in*," and "*Nationality*."

We use a dataset from Liu *et al.* [144], which we denote LD. LD contains examples of the five relations we are interested in, with gold labels inferred from labels provided by crowdsourced workers. In particular, it contains 471 positive and 17,632 negative examples of "*Born in*," 1,375 positive and 16,635 negative examples of "*Died in*," 1,339 positive and 16,136 negative examples of "*Traveled to*," 1,175 positive and 14,231 negative examples of "*Lived in*," and 1,203 positive and 16,230 negative examples of "*Nationality*." `RandomNegs` uses LD as the unlabeled corpus from which Insert-Random-Negative randomly draws examples to automatically label as negative. These examples are free, but can be noisy depending

on the skew (number of negative examples for each positive example) in the corpus, which we will artificially vary in our experimentation.

To create a dataset from which Crowd-Modify+To- will draw examples, we implement a task on Amazon Mechanical Turk that provides workers with a relation and a positive example from LD, and asks them to minimally modify the example to turn it into a negative example for that relation. For example, a good submission for the relation "*Died in*" and the sentence "He died yesterday" might be "He did not die yesterday." In an effort to increase the diversity of examples that workers submit, the task also provides a list of "taboo" words [83, 231] that workers may not use in the sentences they submit. A word becomes "taboo" if the number of times it has been used has exceeded a threshold. We use a threshold of 20. The taboo list is computed by using the words that appear in the modified sentence but not in the original sentence. We set the cost of our task to be \$0.10 after preliminary experimentation showed that this price causes workers to quickly accept our tasks and produce reasonable results. For each relation, we run this task once for each of the positive examples in LD to obtain an equally sized set of negative examples. We denote this dataset LD-Modify. `ModifyNegs` uses LD-Modify as the corpus from which Crowd-Modify+To- will draw negative examples at a cost of \$0.10.

Both strategies simulate the execution of Crowd-Gen+ at a cost of \$0.15 by randomly sampling positive examples from LD. We set the monetary cost of Crowd-Gen+ to be \$0.15 per example since we believe Crowd-Gen+ is slightly harder than Crowd-Modify+To-.

For every relation and every skew $s \in \{1, 9, 99\}$, we set a budget of $b = \frac{0.15\kappa}{2}$, where κ is the number of positive examples for the chosen relation (*e.g.*, $\kappa = 471$ for the "*Born in*" relation). Then we run `RandomNegs` and `ModifyNegs` using this budget.

More specifically, `RandomNegs` spends the entire budget on executing Crowd-Gen+ by randomly sampling $\frac{b}{0.15}$ positive examples from LD and inserting them as positive examples into the training set. Then, it executes Insert-Random-Negative by randomly sampling positive examples from LD with probability $\frac{1}{s+1}$ and negative examples from LD with probability $\frac{s}{s+1}$, and inserts all those examples as negative examples into the training set.

`ModifyNegs` executes Crowd-Gen+ by randomly sampling $\frac{b}{(0.15+0.10)}$ positive examples from LD. Then, it executes Crowd-Modify+To- by inserting the corresponding $\frac{b}{(0.15+0.10)}$ examples from LD-Modify as negative examples. Note that `ModifyNegs` obtains fewer positive examples using Crowd-Gen+ than does `RandomNegs` because Crowd-Modify+To- isn't free like Insert-Random-Negative.

We train logistic regression classifiers using the training sets constructed by `RandomNegs` and `ModifyNegs`. Depending on the examples in the training set, we train using up to 1,000 standard NLP features [156]. We evaluate using a test set from Liu *et al.* [144]. For each classifier we train, we plot a learning curve with test F-score as a function of total cost so far. Then, we compute the area under that curve to get a final F1 AUC statistic. We repeat each experiment until results are statistically significant according to 95% confidence intervals computed using standard error.

Figure 6.1 shows the results. First, we make a sanity-check observation that the performance of `ModifyNegs` does not change across skews, because neither Crowd-Gen+ or Crowd-Modify+To- should be affected by the skew in the unlabeled corpus. Next, we see that as skew decreases, `RandomNegs` becomes less effective, because Insert-Random-Negative erroneously inserts many positive examples into the training set as negative examples.

Next, we observe that at high skew, `RandomNegs` outperforms `ModifyNegs`. However, because of the decreasing effectiveness of Insert-Random-Negative as skew decreases, `ModifyNegs` can be more effective than `RandomNegs` at low skew, suggesting that Crowd-Modify+To- might have value. But generation of examples, whether positive or negative, is likely inferior to standard crowd-labeling in low-skew settings. So we also look at what happens if we simply spend the entire budget on Crowd-Label-Random. We denote this strategy `Label-Only(Random)`.

We set the cost of labeling to be \$0.03 per example (to be consistent with prior work on crowd-labeling [144]). `Label-Only(Random)` obtains $\frac{b}{0.03(s+1)}$ positive-labeled examples by randomly sampling positive examples from LD, and $\frac{b \cdot s}{0.03(s+1)}$ negative-labeled examples by randomly sampling negative examples from LD. However, in order to keep these training sets class-balanced like the previous ones, `Label-Only(Random)` only keeps $\frac{b}{0.03(s+1)}$ of the

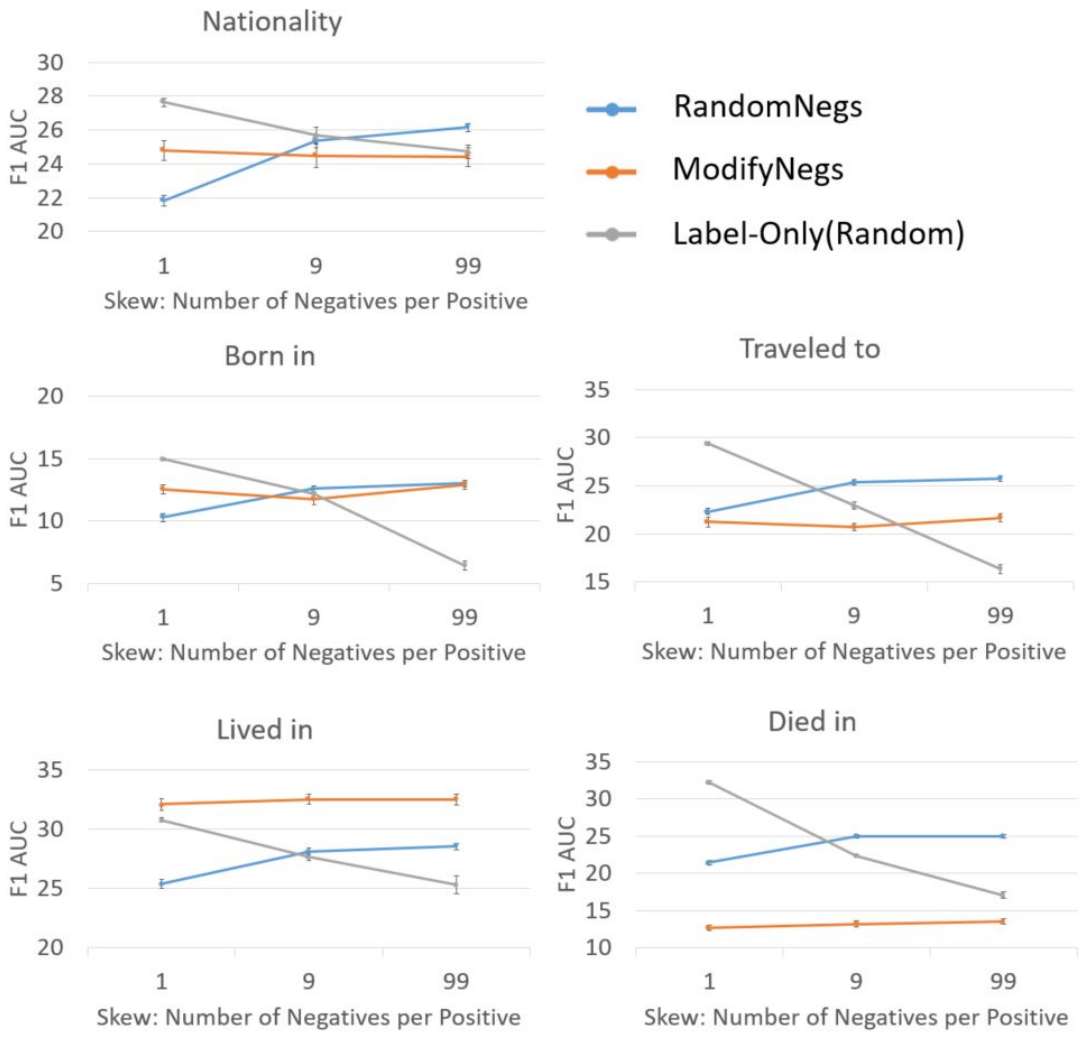

Figure 6.1: As skew decreases, `RandomNegs` become less effective than `ModifyNegs`. However, in such cases, `Label-Only(Random)` is the most effective strategy, so Crowd-Modify+To- is likely not a good EAP to use in any skew setting.

negative examples that it gathers.

Figure 6.1 shows that `Label-Only(Random)` is more effective than `ModifyNegs` at low skew. Thus, we conclude that while Crowd-Modify+To- can be more cost-effective than Insert-Random-Negative in low skew settings, ultimately it is unlikely to be the best EAP to use in *any* skew setting. Thus, we do not continue to investigate the Crowd-Modify+To- EAP. We also note that although we do not experiment with Crowd-Modify-To+ (generation of positive examples by modifying negative ones), we hypothesize that the results would look very similar to these ones that we have with Crowd-Modify+To-.

6.4.2 News Aggregator Data Set

We now describe the setup for our next two experiments. Because the algorithms that we experiment with almost always take different sequences of actions, vastly different training sets can be constructed during every experiment. This behavior makes conducting real online experiments extremely expensive. Therefore, for the next two studies, we construct synthetic datasets using a much larger dataset from the UCI Machine Learning Repository [13].

In particular, we use the News Aggregator Data Set (NADS), which consists of 422,937 news headlines that are labeled as one of four possible topics. 152,746 are labeled as "Entertainment," 108,465 are labeled as "Science and Technology," 115,920 are labeled as "Business," and 45,615 are labeled as "Health."

We use the following process to construct a synthetic dataset: First we pick a skew s and we set a topic to be the positive class (*e.g.*, "Health"). In order to maintain our binary classification setting, we consider all other topics to be part of the negative class. We construct a "Generation Set" by sampling 2000 positive examples from NADS. Anytime an algorithm executes the Crowd-Gen+ action, we randomly sample examples from this Generation Set. Then, we construct a test set by sampling 100 positive examples and $100 \cdot s$ negative examples. Finally, we construct an unlabeled corpus by sampling from the remaining examples as many positive examples as possible while maintaining the desired skew s. When an algorithm executes Labeling Primitives, we sample from this set.

6.4.3 The value of Crowd-Label-PredPos

To compare the Crowd-Label-PredPos and Crowd-Label-Active primitives, we compare two versions of `MB-CB`, denoted `MB-CB(Pos)` and `MB-CB(Active)`. `MB-CB(Pos)` picks between two primitives, Crowd-Gen+ and Crowd-Label-PredPos. `MB-CB(Active)` picks between Crowd-Gen+ and Crowd-Label-Active. Crowd-Label-Active uses uncertainty sampling [126], one of the most popular active learning algorithms, to pick examples for labeling. In both algorithms, we artificially bound the ratio of negative to positive examples in the training set to be at most 3 to 1, using the following method. Any time the algorithms pick Crowd-Gen+, they will automatically execute Insert-Random-Negative 3 times immediately afterward. Any time the algorithms pick Crowd-Label-PredPos or Crowd-Label-Active, if n is the number of obtained positive examples, then the strategy will keep all n positive examples in the training set, but keep at most $3n$ of the obtained negatives and discard the rest. We make an exception if $n = 0$. In this case, we pretend $n = 1$ and keep 3 negative examples so that we are always adding data with each EAP execution.

We do this artificial bounding because researchers have found that when trying to maximize F-score in skewed domains, utilizing a training set with slightly more minority examples than majority examples tends to work well [237].

To compare the two algorithms, we use them to train logistic regression classifiers using a bag of words model. For each skew, we run each algorithm 10 times using a budget of $100. For each run, we generate a new synthetic dataset and plot a learning curve with test F-score as a function of total cost so far. Then, we compute the area under that curve and average over the 10 runs to get a final average F1 AUC statistic.

For both algorithms, instead of making a new decision at every timestep, each EAP is executed 50 times when it is chosen. While making a new decision at every timestep is more optimal, we execute EAPs in batches in order to reduce computational costs.

As before, we set the monetary cost of Crowd-Gen+ to be $0.15 per example and the cost of Crowd-Label-PredPos and Crowd-Label-Active to be $0.03 per example. Thus each

execution of Crowd-Gen+ costs \$7.50 in total and each execution of either of the Labeling Primitives costs \$1.50 in total. While these costs are not the same, part of solving the decision problem is to reconcile differing costs, and our `MB-CB` algorithms are designed to intelligently take these costs into account. We set the exploration constant to be $c_e = 1.0$ for both algorithms, thereby equally balancing between exploration and exploitation terms.

Figure 6.2 shows the results across the four domains in our dataset. We use standard error to show 95% confidence intervals. We see that `MB-CB(Active)` dominates `MB-CB(Pos)` in the "Health" and "Science" domains, and the two algorithms perform about the same in the other two domains. This result is surprising, because `MB-CB(Pos)` uses a primitive designed specifically to locate positive examples, and yet it does no better than `MB-CB(Active)`, even at high skew. To find out why, we investigate the behavior of the two algorithms by comparing how often they execute Generation Primitives versus Labeling Primitives. Intuitively, because the goal of Crowd-Label-PredPos is to find positive examples, the cost of a positive from Crowd-Label-PredPos should be lower than the cost of a positive from Crowd-Label-Active, and therefore `MB-CB(Pos)` should execute more Labeling Primitives and fewer Generation Primitives than `MB-CB(Active)`. We investigate the behavior of the two algorithms when skew is 999, which is when positive examples are most important and we would want `EAP-CP(Pos)` to be most helpful. Surprisingly, Figure 6.3 shows that across all domains, `MB-CB(Pos)` actually executes fewer Labeling Primitives and more Generation Actions than does `MB-CB(Active)` until the budget is almost exhausted, when in some cases `MB-CB(Pos)` begins to start using more Labeling Primitives. Diving in deeper, Figure 6.4 shows that `MB-CB(Active)` actually obtains many more positive examples from Labeling Primitives than does `MB-CB(Pos)`. Again, only when the budget is nearly exhausted does `MB-CB(Pos)` begin to be able to find positive examples. We conclude that classifiers are unable to distinguish between classes early during learning, because otherwise `MB-CB(Pos)` would be able to identify positive examples. Our results show that whether Crowd-Label-PredPos has benefits over the time-tested Crowd-Label-Active is unclear, because by the time classifiers are more competent at identifying positive examples, explicitly finding such exam-

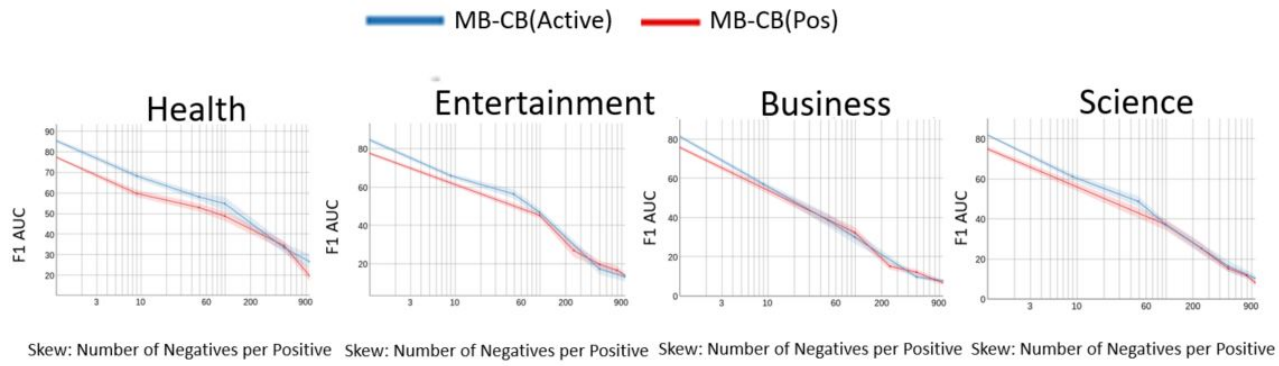

Figure 6.2: `MB-CB(Active)` outperforms `MB-CB(Pos)` in all skew settings when training classifiers to identify "Health" headlines, and shows similar performance for the other domains.

ples may be less impactful, because additional positive examples are most useful when they are difficult to find.

6.4.4 MB-CB and MB-T

In order to answer our third experimental question (how do `MB-T` and `MB-CB` compare against state-of-the-art baselines and are they able to adapt to varying skew and make intelligent decisions), we restrict our attention to algorithms that only choose among a Generation Primitive and a Labeling Primitive, because switching among these two types of EAPs is the most critical tradeoff that an algorithm must make in order to achieve robustness to skew. In particular we will compare algorithms that use Crowd-Gen+, the most crucial Generation Primitive, and Crowd-Label-Active, since we have shown that it works better than Crowd-Label-PredPos. We compare `MB-CB(Active)` and `MB-T(Active)` against two simple baselines, `Round-Robin` and `Label-Only(Active)`, and two state-of-the-art algorithms, `GL` and `GL-Hybrid` [11].

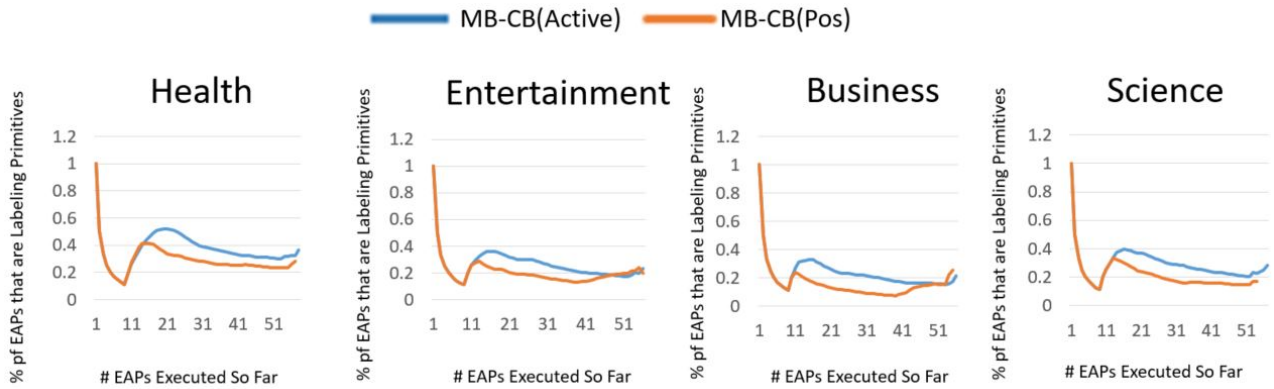

Figure 6.3: When the skew is 999, `MB-CB(Active)` executes Labeling Primitives more often and Generation Primitives less often than `MB-CB(Pos)`.

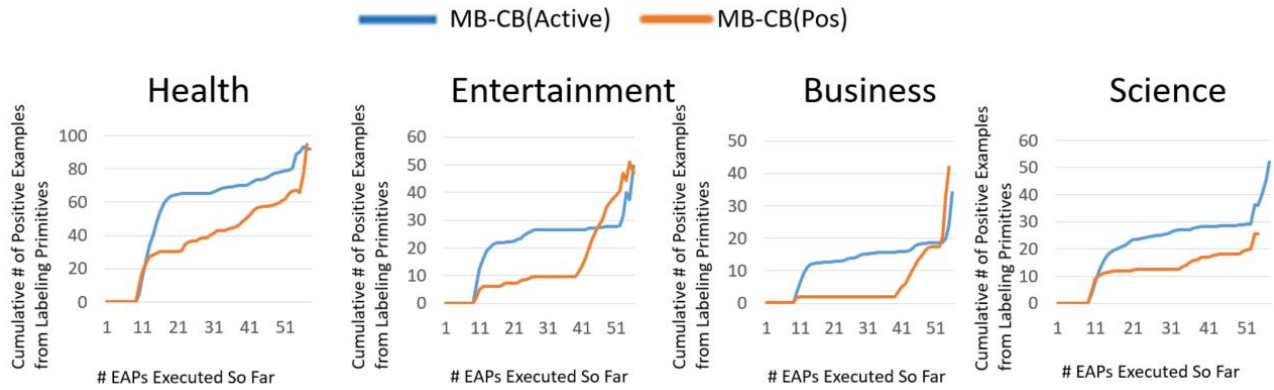

Figure 6.4: When the skew is 999, `MB-CB(Active)` surprisingly obtains more positive examples from Labeling Primitives than does `MB-CB(Pos)`.

Baselines

`GL` (Guided Learning) [11] tasks workers with generating examples at a specified class ratio. We implement this baseline by having `GL` simply cycle between executing Crowd-Gen+ once and `Insert-Random-Negative` 3 times in order to match the ratio that we use in `MB-CB(Active)` and `MB-T(Active)`. `GL` does not execute Crowd-Label-Active. `GL-Hybrid` [11] begins by executing `GL`. After every action, it estimates the classifier's performance using cross-validation and constructs a learning curve. It then estimates future expected gain in performance by estimating the derivative of the learning curve at the last computed point. When the derivative drops below a threshold, it switches to executing Crowd-Label-Active and never executes `GL` again. We set the threshold to be 0.00005 as suggested by Attenberg *et al.*.

`Round-Robin` simply cycles between GL and Crowd-Label-Active. While a non-adaptive and unsophisticated baseline, `Round-Robin` does have some nice properties. Intuitively, statically alternating between the two possible primitives means that the number of times that Round-Robin will take a subobtimal action is bounded. Furthermore, because Crowd-Gen+ may raise recall, Crowd-Label-Active may raise precision, and F-score is maximized when recall and precision are equal, alternating between these two EAPs may be an effective way to increase F-score. Our other simple baseline, `Label-Only(Active)` only executes Crowd-Label-Active and never uses Crowd-Gen+.

Synthetic Data

We first present results using the synthetic datasets that we construct. We then present results using real crowdsourced data in the next section. We use the same metrics, costs, and execution batching as in Section 6.4.3 to compare algorithms.

Figure 6.5 shows the results. We first observe that `Label-Only(Active)` achieves high F1 AUC in low-skew environments, but also produces training sets that are unable to train anything in high-skew environments. We expect this behavior, because while labeling is an

effective strategy in low-skew environments, it is ineffective by itself in high-skew environments when it will almost never find any positive examples.

Next, we observe that GL performs extremely poorly in low-skew environments. This result is unsurprising for primarily two reasons. First, GL does not utilize any Labeling Primitives, which are essential in low-skew settings and useful in high-skew settings after initial training. And second, in low-skew settings, GL also acquires many noisy negative examples from Insert-Random-Negative, which prevent quick improvements in performance.

Next, we observe that while GL-Hybrid improves upon GL, the improvement is not substantial. GL-Hybrid has a number of weaknesses. First, it is difficult to set the threshold parameter. Second, the estimations used to compute whether to switch to active learning can be wildly wrong, causing the algorithm to switch from guided learning to active learning either far too early or far too late. For example, in a low-skew setting, GL-Hybrid may execute Crowd-Gen+ for an extremely long time if the performance of the classifier consistently rises.

Next, we see that Round-Robin is not a bad strategy, achieving better results than GL and GL-Hybrid at low-skew, and Label-Only(Active) at high skew. Again, we expect such a result because the number of suboptimal actions that Round-Robin can take is limited.

Finally, we observe that MB-CB(Active) and MB-T(Active) are the best algorithms, as they are able to completely automatically eliminate excesses in low-skew environments while closely matching state-of-the-art baselines in high-skew environments. Overall, MB-T(Active) performs worse than MB-CB(Active), and we attribute this result to MB-T(Active) not being as robust as MB-CB(Active) is to the batch execution of primitives that they utilize. Interestingly, we also see that Round-Robin often shows small but significant gains over MB-CB(Active) and MB-T(Active) at very high skew settings. We attribute these gains to the very high value of Crowd-Gen+ in these settings.

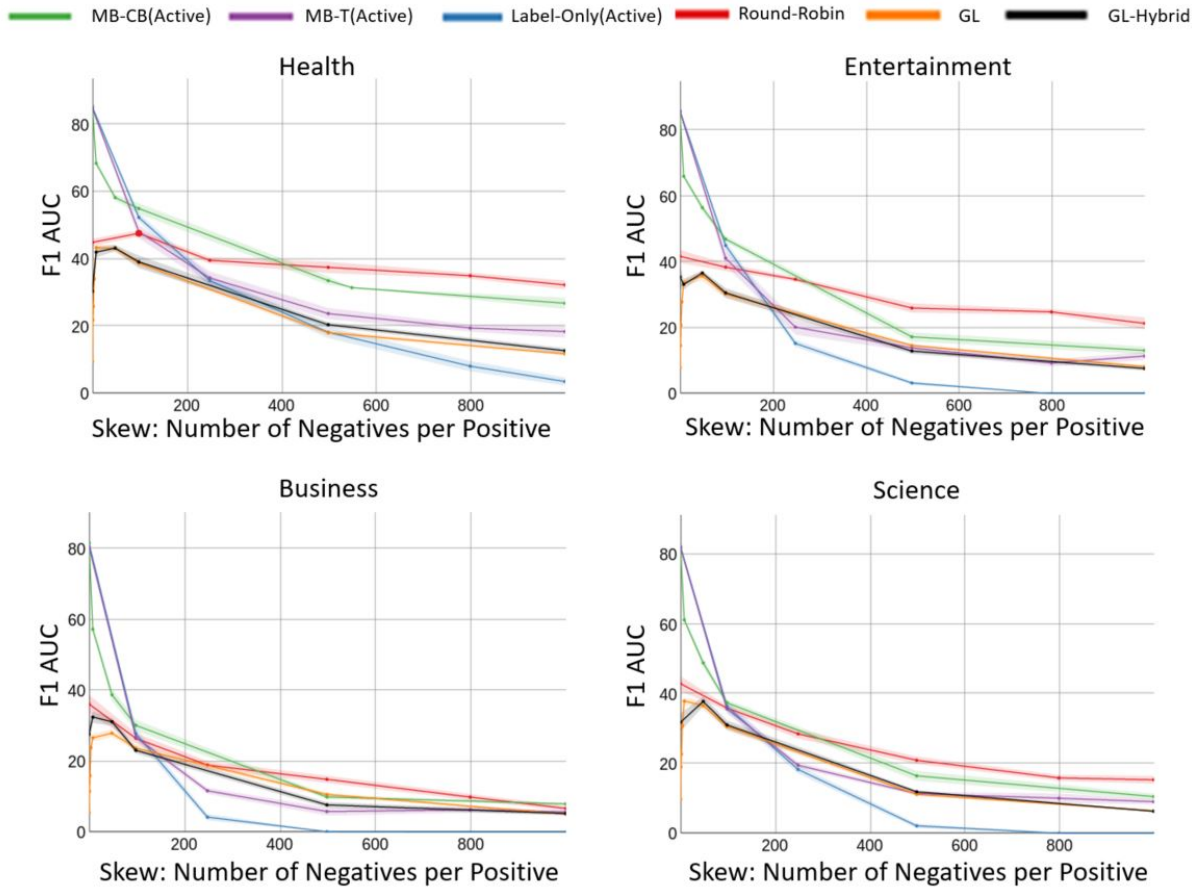

Figure 6.5: Comparison of `MB-T(Active)`, `MB-CB(Active)`, `Round-Robin`, `Label-Only(Active)`, `GL`, and `GL-Hybrid` on 4 domains with synthetic Generation Sets. `MB-T(Active)` and `MB-CB(Active)` both train better classifiers than the state-of-the-art baselines, `GL` and `GL-Hybrid`, across many domains and skew settings.

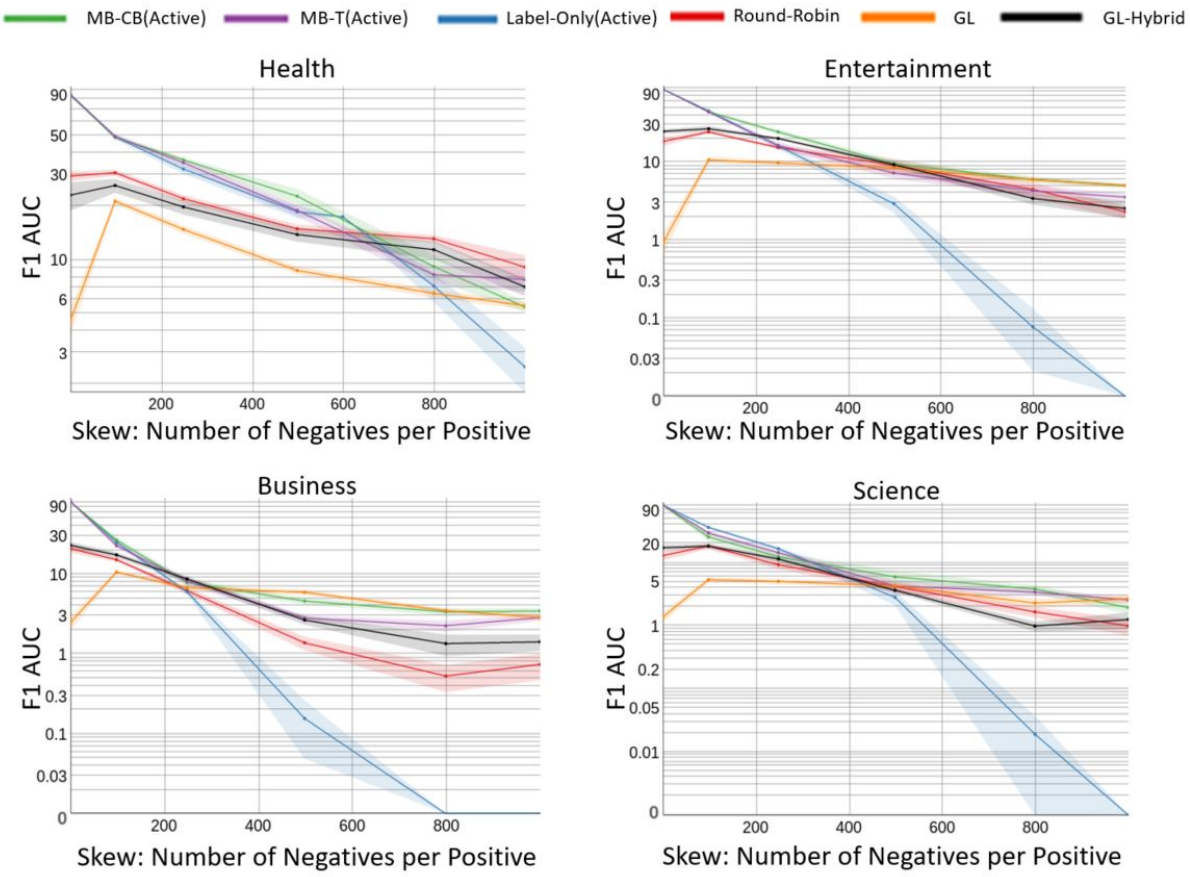

Figure 6.6: Comparison of `MB-T(Active)`, `MB-CB(Active)`, `Round-Robin`, `Label-Only(Active)`, `GL`, and `GL-Hybrid` on 4 domains using Generation Sets with real data. `MB-CB(Active)` and `MB-T(Active)` both train better classifiers than the state-of-the-art baselines, `GL` and `GL-Hybrid`, across many domains and skew settings.

Real Generation Sets

Next, we run the same experiment, but instead of using a synthetic Generation Set from which Crowd-Gen+ samples, we use real data gathered from Amazon Mechanical Turk. For each topic, we ask workers to copy and paste a news headline from a news article that is an example of that topic. We gather 999 headlines for each topic and use these as the Generation Sets. We note that these new Generation Sets contain examples that are likely from a completely different distribution than the unlabeled corpus and test set.

Figure 6.6 shows the results. We use a log scale on the y-axis in order to more clearly show the differences between the algorithms. We see that many of the patterns from before remain. `Label-Only(Active)` performs well in low-skew environments, but eventually is unable to learn anything at the highest skews; `GL` is a strong strategy only in high-skew domains; and `MB-CB(Active)` and `MB-T(Active)` are still the most robust algorithms overall, with `MB-CB(Active)` averaging a 14.3 gain in F1 AUC over state-of-the-art `GL-Hybrid` across all skews and domains. Interestingly, `Round-Robin` no longer dominates and sometimes produces worse results than `GL-Hybrid` in low to moderately skewed settings, presumably because now that the Generation Set contains examples from a different distribution than that of the test set, Crowd-Gen+ produces less useful examples. Thus, `Round-Robin`'s repeated use of Crowd-Gen+ is very detrimental. In contrast, GL-Hybrid does better because it is able to switch from Crowd-Gen+ to Crowd-Label-Active more quickly than before, because the positive examples from the Generation Set provide a much milder performance boost. Of course, these results are dependent on the price of the the Labeling Primitives and Generation Primitives. We pay workers \$0.15 per example for their generated examples, and we hypothesize that such a payment is probably much more than necessary. If we were to modestly decrease the cost of Crowd-Gen+ to \$0.10, then strategies like `GL-Hybrid`, `MB-CB(Active)`, and `MB-T(Active)` would likely do much better.

6.5 Related Work

6.5.1 High-Skew Active Learning

Many researchers have considered active learning in high-skew environments. Tropel [167] is a system that bootstraps visual detectors by selecting examples for crowd-labeling using a nearest-neighbor method. Similarly, Doersch *et al.* [65] use a nearest-neighbor method to iteratively bootstrap detectors for rare visual patterns in an unsupervised manner. Vijayanarasimhan *et al.* [229] introduce a method for training object detectors using a combination of keyword search and uncertainty sampling. Patterson *et al.* [166] model correlations between sparse image attributes in order to more intelligently ask the crowd about those attributes with the end goal of constructing a dataset with over 3 million object-attribute annotation pairs. Tomanek *et al.* [220] modify a query-by-committee approach to weight the importance of different classes. Bloodgood *et al.* [30] estimate skew by randomly sampling examples in order to set class-specific costs for the purpose of cost-sensitive SVM active learning. Zhu *et al.* [255] consider the effects of oversampling and undersampling during active learning. They also develop a bootstrap-based oversampling strategy. He *et al.* [84] propose an active learning strategy in which minority examples are assumed to be tightly grouped and can be more effectively sampled via a nearest-neighbor strategy. Li *et al.* [128] use two classifiers to predict whether examples belong to the minority class. Examples that are highly-uncertain according to one classifier and highly-certain according to the other are selected to be labeled. In a similar vein, Wallace *et al.* [232] label examples based on disagreement between two classifiers, one trained using labeled examples, and one that makes predictions using expert-provided labeled features.

Our contributions in this chapter are orthogonal to this body of work. Our goal is not to design a single new active learning algorithm. Instead, we recognize that there may be many such algorithms, each of which may be beneficial in different settings, and our goal is to be able to intelligently choose among them adaptively depending on the setting at hand.

6.5.2 Guided Learning and Example Generation

Attenberg *et al.* [11] propose *guided learning* in which humans can be directed to search for or generate positive examples. They show that in high-skew settings, guided learning is more effective than active learning, and guided learning followed by active learning is more effective than either guided learning or active learning in isolation. In contrast, we consider how to design algorithms that can dynamically adapt to various skew settings by choosing the best EAP to execute at every timestep.

Many researchers have used the idea of guided learning in various applications. Sculley *et al.* [189] implement guided learning to build models to identify bad advertisements. Sadilek *et al.* [186] use guided learning to build models for identifying Twitter tweets about foodborne illness. Wang *et al.* [233] consider how to crowdsource training examples for dialogue systems, by designing interfaces that enable workers to construct sentences that correspond to given semantic forms by paraphrasing sentences, scenarios, and lists that correspond to those semantic forms. Wang *et al.* [234] develop a grammar that maps semantic forms to English sentences, and then ask workers to paraphrase those sentences to generate a training corpus for semantic parsing. Zhang *et al.* [252] consider how to generate "syntactically similar" training examples for the task of visual question answering. Given a dataset of images and yes/no questions about those images, they balance the dataset by showing crowd workers image/question pairs and asking them to minimally modify the image so that the answer to that question flips.

In our experimentation, in order to compare algorithms for selecting among EAPs, we also develop worker tasks that instantiate guided learning for event extraction and topic modeling.

6.5.3 Heuristics for Identifying Positive Examples

Another way to ensure balanced training sets in high-skew domains is to use a heuristic to noisily label examples. In the field of information extraction, distant supervision [49, 156] is a

commonly employed strategy in which an existing knowledge base is used to label examples of text. If a relation between entities exists in a knowledge base, then any example that contains those entities is labeled as a positive example of that relation. Unfortunately, the assumption that the target concept is in a knowledge base is, in many cases, unrealistic.

Data programming [71] is a paradigm in which humans design domain-specific rules that can be programmatically used to label examples. Hoffmann *et al.* [89] design a system that allows both NLP experts and novices to create rule-based information extractors in one hour by guiding them towards promising rules. While data programming cannot be easily deployed across disparate domains, it is a strategy for obtaining examples, and can be considered an EAP.

6.6 Conclusion

We consider the problem of selecting EAPs in order to maximize classifier performance. After listing several existing EAPs and proposing some novel EAPs, we introduce two algorithms for the problem of selecting EAPs: `MB-CB` and `MB-T`. These algorithms work by computing the cost of obtaining a single positive example from each EAP and pick the EAP that has the lowest such cost. Because these costs can only be learned through execution of the EAPs, `MB-CB` and `MB-T` adapt multi-armed bandit algorithms for making the tradeoff between exploiting the EAP they believe to be cheapest and exploring the true cost of other EAPs.

Then we answer several questions about EAPs and the algorithms we present with experiments on real and synthetic datasets. We first show that gathering near-miss negative examples is not cost-effective compared to inserting random examples from the unlabeled corpus as negative. Then, we demonstrate that surprisingly, trying to label predicted positive examples actually results in finding fewer positive examples. In particular, during the early stages of training, Crowd-Label-PredPos is not as effective as Crowd-Label-Active at either producing good training data or producing positive examples. Finally, we show that unlike state-of-the-art baselines `GL` and `GL-Hybrid` [11], `MB-CB` and `MB-T` both have the ability to

adapt to domains of varying skew, with `MB-CB` yielding a 14.3 point gain on average in F1 AUC over 24 environments (6 domains $\times$ 24 skews) compared to `GL-Hybrid`.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this dissertation, we push forward the intelligent management of crowd-powered machine learning, by designing models and algorithms for the dynamic gathering of training and testing data. In the first part of the dissertation, we consider how to cheaply and accurately gather labels for testing purposes. Because test sets must be extremely clean, machine learning practitioners typically build and test several crowdsourcing workflows for their task, identify and redundantly deploy the single best-performing workflow, and then infer gold labels from the worker responses that are obtained from the various runs of that single workflow. We improve this process in two ways. First, we show that throwing away workflows in this manner is suboptimal, and design a model and agent for dynamically choosing the best workflow to deploy given the responses so far. Second, we design a model and agent for aggregating labels for tasks that are free-response, as opposed to multiple-choice. In both cases we show that our agent can improve accuracy and cut costs compared to state-of-the-art baselines.

In the second part of the dissertation, we move from the realm of test data to the realm of training data. We first consider the implications of the fact that unlike test data, training data does not have to be clean, because learning algorithms are often robust to noise. We investigate the tradeoff between accuracy (via relabeling) and size by analyzing how various properties of learning problems affect whether larger noisier training sets or smaller cleaner ones will train better classifiers. We find that expressive classifiers, moderately accurate workers, and small budgets favor strategies that do more relabeling. Then, we generalize active learning to allow for relabeling in a formalism we call re-active learning, and develop

algorithms for this new setting that outperform active learning baselines. Finally, we investigate how to collect a balanced training set, which can be difficult to achieve using standard crowd-labeling techniques if the domain is highly skewed. Given the ability to spend additional money to ask workers to generate examples of a minority class, we develop algorithms that can dynamically switch between generating and labeling examples and show that our algorithms outperform state-of-the-art baselines.

7.1.1 Guidelines for Crowd-Powered Machine Learning

We now list several key takeaways for the machine learning practitioners who wish to use the research in this thesis for their own practical applications.

1. Relabeling training data is not always the best strategy. Unless budgets are low, or classifiers are prone to overfitting, using free platform tools for quality-control (*e.g.*, restricting the set of workers to those with 95% approval rating) in combination with unilabeling will work well.

2. Use the crowd to generate, not just label, examples. Many real-world domains are highly-skewed and learning would be impossible without the ability to generate positive examples. Use our skew-robust algorithms (Chapter 6) to switch between generation and traditional crowd-labeling. They are fast and effective.

3. EM-based models that account for task difficulty and worker skill are much more effective than majority vote at aggregating labels from tasks that have a single objective true answer. Use existing EM-based models when tasks have a finite set of possible answers (*e.g.*, [242]), and our Chinese Restaurant Process-based model (Chapter 3) when tasks are open-ended.

4. Impact sampling (Chapter 5) can be an effective strategy for re-active learning given a lot of CPU time. However, time-tested vanilla uncertainty sampling (with no relabeling) [49] will work just fine if learning needs to happen quickly.

7.2 Future Work

We believe that this dissertation can spawn a long line of fruitful research. We are most excited about developing an agent that combines all the ideas in this dissertation into one decision-theoretic model in order to manage the entire machine learning pipeline, from the acquisition and labeling of training examples, to the self-testing and adjustment of hyper-parameters.

Instead of managing each part separately by solving independent control problems, the agent should have a model that jointly considers the effects of possible actions on the entire pipeline. For example, spending too much money on labeling testing examples may result in having too little money for labeling training examples. Or trying out different workflows for labeling too often may hinder efforts to increase diversity through example generation. We separately considered the noise/quantity tradeoff and the class-diversity/quantity tradeoff, but all three properties of training sets should be considered at the same time. Such an agent might also merge ideas across problems. For example, perhaps the agent should dynamically switch between workflows for generating examples.

We also believe there are a number of related promising and impactful future research directions outside of the super-agent that we envision:

- We insist throughout the dissertation that if data is to be used for evaluation, its sole requirement is to be clean. While cleanliness is indeed the primary requirement, a good test set should satisfy other requirements as well. In particular, if we want to have a high confidence estimate of a classifier's quality, not just an unbiased one, then the size and distribution of the test set are important. We imagine many of the techniques we use in the second part of the dissertation could be useful for designing models and algorithms for making tradeoffs between size, diversity, and noise when collecting data for testing. For example, perhaps the crowd can help produce examples that lie in under-tested areas of the distribution that the classifier is trying to learn.

- In Chapter 2, we manually handcraft a POMDP for the crowd-powered machine learning problem that we try to solve, just as Dai *et al.* [54] manually handcraft a POMDP for the crowd-powered machine learning problem that they try to solve. If in the future some AI non-experts want to use POMDP machinery to solve their new crowd-powered machine learning problem problem, they would also need to manually handcraft a POMDP, or more likely, hire an expert to do it for them. But all of these POMDPs would look very similar, because they would all need to reason about answers, workers, tasks, and other components related to the process of crowd-powered machine learning. We imagine a system or programming language that allows AI non-experts with no knowledge of probability or belief states to easily build POMDPs by specifying a workflow with *choice points* (*e.g.* get another label or submit final answer) that can be automatically optimized. Researchers have outlined and taken preliminary steps towards this vision [239, 238, 131], but much work remains in proving its worth and scaling it to larger and more diverse problems.

- The research in this dissertation focuses on agents that can affect the world in limited ways, for example by asking crowd workers to classify or generate examples of text. However, we believe machine learning has much potential if powered using richer interactions with both unskilled and skilled humans. The most impactful agents will be seamlessly embedded in our world, solving a wide variety of problems, both simple and difficult (*e.g.* driving cars). In order to create such agents, training examples may need to be more complex than simple images or text. For example, training sets for a household assistant might include videos of humans cleaning bathrooms or walking dogs. Are videos necessary, or would images of a sequence of steps suffice? Relabeling might involve agents proposing actions to trusted authorities while working, just as medical interns might propose diagnoses to their attending physicians. Active and reactive learning become much more interesting with a broad variety of possible training examples.

- When active learning and re-active learning can manipulate the world in more impactful ways, how can an agent reason about the real consequences that may occur? For example, how does an agent identify actions that are potentially problematic, and when it does, how should it proceed? An agent that is trying to get better at reading different kinds of fonts around the house should not ask random crowd workers to label an image of a credit card. This problem is the one of "safe exploration" in Safe AI/Reinforcement Learning [5][224]. One way to approach the first part of this problem (identifying problematic actions) is to train a problem predictor, by asking the crowd to generate scenarios in which actions may have high variance or high negative rewards. For example, for an agent that learns primarily through vision, we could ask the crowd to provide example images from the web that should not be shared. Another issue agents must be able to reason about is how often they should ask for help from its owner versus the crowd. This problem is the one of "scalable oversight" [5]. One possibility is the agent can ask for feedback from its owner before it executes every single action. But such behavior is clearly suboptimal, since the crowd can also be used as an on-demand resource when the agent determines that it does not necessarily need to bother the owner, and instead asking for help from a slightly less trusted source (like the crowd) would suffice. How often an agent should query the crowd probably depends on the preferences of the owner as well as the domain, which can be modeled and learned. As a starting point, the crowd can provide data to bootstrap generic algorithms, for example by answering hypothetical questions about how often they, as an owner, might wish to be queried, or providing ways of obscuring images so that they may be safely shown to others.

- All the research in this dissertation assumes that more learning is always valuable. But in many cases, the cost of learning may be too high to justify continued learning. For example, a medical assistant bot that cannot substantially increase its performance on identifying malignant tumors may be better off learning about warning signs of septic

shock. While researchers have looked at this problem in the context of planning [132], understanding this question in the context of learning is important as well. One way to approach this problem is to use heuristics to cut off learning when an agent's learning curve has converged. A more principled way is to use decision-theoretic machinery like POMDPs to model the tradeoff between continuing to learn a concept versus switching to a new one, but all POMDP models require utility elicitation, a difficult problem. Yet another way is to train a system that determines when learning should stop. This system could be trained by asking the crowd to label scenarios in which learning should stop or continue. More concretely, one might be able to show the crowd learning curves and ask them questions like "Which curve looks like it has the most potential to increase?" The crowd could also be asked on-demand by the agent whether it should move on.

We believe that a world in which machines can work side-by-side with humans, learning from them, teaching them, and cooperating with them, is a world in which we will be able to more quickly advance society. We hope that this dissertation provides a solid foundation upon which future work in pursuit of that world can easily grow.

BIBLIOGRAPHY

[1] 2016 tac-kbp event argument linking pilot dataset, 2016.

[2] 2017 tac-kbp event track, 2017.

[3] David J. Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII 1983*, volume 1117 of *Lecture Notes in Mathematics*, pages 1–198. Springer Berlin / Heidelberg, 1985. 10.1007/BFb0099421.

[4] Amazon. Mechanical turk case study: Castingwords. `http://aws.amazon.com/solutions/ case-studies/castingwords-interview`.

[5] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

[6] David Andre and Stuart J. Russell. Programmable reinforcement learning agents. In *NIPS*, 2001.

[7] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *AAAI*, 2002.

[8] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[9] Askville. `http://askville.amazon.com`.

[10] Javed A. Aslam and Scott E. Decatur. General bounds on statistical query learning and pac learning with noise via hypothesis boosting. *Information and Computation*, 141(2):85–118, March 1998.

[11] Josh Attenberg and Foster Provost. Why label when you can search? alternatives to active learning for applying human resources to build classification models under extreme class imbalance. In *KDD*, 2010.

[12] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[13] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[14] Maria-Florina Balcan, Christopher Berlind, Avrim Blum, Emma Cohen, Kaushik Patnaik, and Le Song. Active learning and best-response dynamics. In *NIPS*, 2014.

[15] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *ICML*, 2006.

[16] Maria-Florina Balcan, Avrim Blum, and Yishay Mansour. The price of uncertainty. In *EC: Proceedings of the tenth ACM conference on Electronic commerce*, pages 285–294, 2009.

[17] Radha-Krishna Balla and Alan Fern. UCT for tactical assault planning in real-time strategy games. In *IJCAI*, pages 40–45, 2009.

[18] Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *ACL*, 2008.

[19] Daniel W. Barowy, Emery D. Berger, and Andrew McGregor. Automan: A platform for integrating human-based and digital computation. Technical report, University of Massachusetts, Amherst, 2012.

[20] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[21] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[22] Michael Bernstein, David Karger, Rob Miller, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. In *Collective Intelligence*, 2012.

[23] Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *UIST*, 2011.

[24] Michael S. Bernstein, Greg Little, Robert C. Miller, Bjrn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: A word processor with a crowd inside. In *UIST*, 2010.

[25] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol 1, 2nd ed.* Athena Scientific, 2000.

[26] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol 2, 2nd ed.* Athena Scientific, 2001.

[27] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[28] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, and Tom Yeh. Vizwiz: nearly real-time answers to visual questions. In *UIST*, pages 333–342, 2010.

[29] Christopher M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[30] Michael Bloodgood and K. Vijay-Shanker. Taking into account the differences between actively and passively acquired data: The case of active learning with support vector machines for imbalanced datasets. In *NAACL*, 2009.

[31] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, pages 92–100, 1998.

[32] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.

[33] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2001.

[34] Jonathan Bragg, Mausam, and Daniel S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *HCOMP*, 2013.

[35] Jonathan Bragg, Mausam, and Daniel S. Weld. Parallel task routing for crowdsourcing. In *HCOMP*, 2014.

[36] S.R.K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *ACL-IJCNLP*, 2009.

[37] Emma Brunskill, Leslie Kaebling, Tomas Lozano-Perez, and Nicholas Roy. Continuous-state pomdps with hybrid dynamics. In *Symposium on Artificial Intelligence and Mathematics*, 2008.

[38] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[39] Anthony R. Cassandra. A survey of pomdp applications. In *AAAI Fall Symposium: Planning with POMDPs*, 1998.

[40] Castingwords. `http://castingwords.com`.

[41] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *AAAI*, 2000.

[42] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *NIPS*, 2011.

[43] Xi Chen, Paul N. Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, 2013.

[44] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: crowdsourcing taxonomy creation. In *CHI*, pages 1999–2008, 2013.

[45] Paul R. Cohen, David Jensen, and Producing C. Overfitting explained. In *Sixth International Workshop on Artificial Intelligence and Statistics*, pages 115–122, 1996.

[46] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beene, Andrew Leaver-Fay, David Baker, Zoran Popovic, and Foldit players. Predicting protein structures with a multiplayer online game. *Nature*, 446:756–760, 2010.

[47] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592, 2003.

[48] Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 414–422. Curran Associates, Inc., 2009.

[49] Mark Craven and John Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999.

[50] crowdflower.com. Crowdflower research board. http://blog.crowdflower.com/.

[51] Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. Pomdp-based control of workflows for crowdsourcing. *Artificial Intelligence*, 202:52–85, 2013.

[52] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

[53] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

[54] Peng Dai, Mausam, and Daniel S Weld. Artificial intelligence for artificial intelligence. In *AAAI*, 2011.

[55] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *NIPS*, 2004.

[56] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In *NIPS*, 2005.

[57] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. In *ICML*, 2008.

[58] A.P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, 28(1):20–28, 1979.

[59] Ofer Dekel, Claudio Gentile, and Karthik Sridharan. Robust selective sampling from single and multiple teachers. In *COLT*, 2010.

[60] Ofer Dekel and Ohad Shamir. Good learners for evil tecahers. In *ICML*, 2009.

[61] Ofer Dekel and Ohad Shamir. Vox populi: Collecting high-quality labels from a crowd. In *COLT*, 2009.

[62] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[63] Nan Ding and S.v.n. Vishwanathan. t-logistic regression. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 514–522. Curran Associates, Inc., 2010.

[64] Dominic DiPalantino and Milan Vojnovic. Crowdsourcing and all-pay auctions. In *EC: Proceedings of the tenth ACM conference on Electronic commerce*, pages 119–128, 2009.

[65] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (SIGGRAPH)*, 31(4):101:1–101:9, 2012.

[66] Pinar Donmez and Jaime G. Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *CIKM*, pages 619–628, 2008.

[67] Pinar Donmez, Jaime G. Carbonell, and Jeff Schneider. Efficiently learning the accuracy of labeling sources for selective sampling. In *KDD*, 2009.

[68] Pinar Donmez, Jaime G. Carbonell, and Jeff Schneider. A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *SIAM International Conference on Data Mining (SDM)*, pages 826–837, 2010.

[69] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[70] Richard M. Dudley. Central limit theorems for empirical measures. *The Annals of Probability*, 6(6):899–929, 1978.

[71] Henry R. Ehrenberg, Jaeho Shin, Alexander J. Ratner, Jason A. Fries, and Christopher Ré. Data programming with ddlite: Putting humans in a different part of the loop. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 13:1–13:6, New York, NY, USA, 2016. ACM.

[72] March 2010. `http://www.facebook.com`.

[73] Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic programming for structured continuous markov decision problems. In *UAI*, pages 154–161, 2004.

[74] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149, 1964.

[75] Freebase. Freebase, 2011. http://www.freebase.com/.

[76] Dan Geiger, Thomas Verma, and Judea Pearl. Identifying independence in bayesian networks. *Networks*, 20(5):507–534, 1990.

[77] Sylvain Gelly and Yizao Wang. Exploration exploitation in go: UCT for monte-carlo go. In *NIPS On-line trading of Exploration and Exploitation Workshop*, 2006.

[78] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(2009):12, 2009.

[79] Karan Goel, Shreya Rajpal, and Mausam. Octopus: A framework for cost-quality-time optimization in crowdsourcing. In *HCOMP*, 2017.

[80] Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *NIPS*, 2010.

[81] David Alan Grier. Error identification and correction in human computation: Lessons from the WPA. In *HCOMP*, 2011.

[82] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012.

[83] Hasbro. Taboo^{TM} `hasbro.com/common/instruct/taboo(2000).pdf`, 2000.

[84] Jingrui He and Jaime Carbonell. Nearest-neighbor-based active learning for rarer category detection. In *NIPS*, 2007.

[85] J Heyman and D. Ariely. Effort for payment: A tale of two markets. *Psychological Science*, 15(11):787–793, 2004.

[86] Chien-Ju Ho, Shahin Jabbari, and Jennifer W Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML*, pages 534–542, 2013.

[87] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[88] Leah Hoffmann. Crowd control. *CACM*, 52(3):16–17, March 2009.

[89] Raphael Hoffmann, Luke S. Zettlemoyer, and Daniel S. Weld. Extreme extraction: Only one hour per relation. *CoRR*, abs/1506.06418, 2015.

[90] John Horton and Lydia Chilton. The labor economics of paid crowdsourcing. *CoRR*, abs/1001.0627, 2010.

[91] Eric Huang, Haoqi Zhang, David C. Parkes, Krzysztof Z. Gajos, and Yiling Chen. Toward automatic task design: A progress report. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 77–85, 2010.

[92] Panagiotis G. Ipeirotis, Foster Provost, Victor S. Sheng, and Jing Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 28(2):402–441, 2013.

[93] Panagiotis G. Ipeirotis, Foster Provost, Victor S. Sheng, and Jing Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 2013.

[94] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 64–67, 2010.

[95] Panos Ipeirotis. A computer scientist in a business school, 2010. http://behind-the-enemy-lines.blogspot.com/.

[96] Panos Ipeirotis. Demographics of mechanical turk. In *Working Paper*, 2010.

[97] Shaili Jain, Yiling Chen, and David C. Parkes. Designing incentives for online question and answer forums. In *EC: Proceedings of the tenth ACM conference on Electronic commerce*, pages 129–138, 2009.

[98] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[99] Radu Jurca and Boi Faltings. Incentives for expressing opinions in online polls. In *EC: Proceedings of the 9th ACM conference on Electronic commerce*, pages 119–128, 2008.

[100] Matti Kääriäinen. Active learning in the non-realizable case. In *ALT*, 2006.

[101] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[102] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[103] Hiroshi Kajino, Yuta Tsuboi, and Hisashi Kashima. A convex formulation for learning from crowds. In *AAAI*, 2012.

[104] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*, 2012.

[105] Ece Kamar and Eric Horvitz. Planning for crowdsourcing hierarchical tasks. In *AAMAS*, 2015.

[106] Ece Kamar, Ashish Kapoor, and Eric Horvitz. Lifelong learning for acquiring the wisdom of the crowd. In *IJCAI*, 2013.

[107] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. In *VLDB*, 2013.

[108] Ashish Kapoor, Eric Horvitz, and Sumit Basu. Selective supervision: Guiding supervised learning with decision-theoretic active learning. In *IJCAI*, 2007.

[109] David R. Karger, Sewoong Oh, and Devavrat Shah. Budget-optimal crowdsourcing using low-rank matrix approximations. In *Conference on Communication, Control, and Computing*, 2011.

[110] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowd-sourcing systems. In *NIPS*, 2011.

[111] David R. Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, pages 81–92, 2013.

[112] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. In *ACM Symposium on the Theory of Computing*, 1993.

[113] Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17:115–141, 1994.

[114] Roni Khardon and Gabriel Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007.

[115] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *CHI*, pages 453–456, 2008.

[116] Frank Kleemann, Gerd Gnter Vo, and Kerstin Rieder. Un(der)paid innovators: The commercial utilization of consumer work through crowdsourcing. *Science, Technology & Innovation Studies*, 4(1):5–26, 2008.

[117] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.

[118] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1143, 1995.

[119] Georg Krempl, Daniel Kottke, and Vincent Lemaire. Optimised probabilistic active learning. *Machine Learning*, 100(2):449–476, 2015.

[120] Anand Kulkarni, Matthew Can, and Bjorn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of CSCW*, 2012.

[121] Anand Kulkarni, Philipp Gutheim, Prayag Narula, David Rolnitzky, Tapan Parikh, and Bjoern Hartmann. Mobileworks: Desigining for quality in a managed crowdsourcing architecture. In *HCOMP*, 2012.

[122] Philip D. Laird. *Learning from Good and Bad Data*. Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1988.

[123] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. Identifying unknown unknowns in the open world. In *AAAI*, 2017.

[124] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. Real-time crowd control of existing interfaces. In *Proceedings of UIST*, 2011.

[125] LDC. Linguistic Data Consortium: Annotation Overview, `ldc.upenn.edu/communications/ data-sheets/annotation-overview`, 2015.

[126] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *ICML*, 1994.

[127] Lihong Li and Michael L. Littman. Lazy approximation for solving continuous finite-horizon mdps. In *AAAI*, pages 1175–1180, 2005.

[128] Shoushan Li, Shengfeng Ju, Guodong Zhou, and Xiaojun Li. Active learning for imbalanced sentiment classification. In *EMNLP-CoNLL*, 2012.

[129] Zhifei Li, Patrick Nguyen, and Geoffrey Zweig. Optimal dialog in consumer-rating systems using a pomdp framework. In *SIGdial: Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*, pages 104–111, 2008.

[130] Beatrice Liem, Haoqi Zhang, and Yiling Chen. An iterative dual pathway structure for speech-to-text transcription. In *HCOMP*, 2011.

[131] C. H. Lin, Mausam, and D. S. Weld. A Programming Language With a POMDP Inside. *ArXiv e-prints*, August 2016.

[132] Christopher H. Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for planning under uncertainty. In *IJCAI*, 2015.

[133] Christopher H. Lin, Mausam, and Daniel S. Weld. Crowdsourcing control: Moving beyond multiple choice. In *UAI*, 2012.

[134] Christopher H. Lin, Mausam, and Daniel S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *AAAI*, 2012.

[135] Christopher H. Lin, Mausam, and Daniel S. Weld. To re(label), or not to re(label). In *HCOMP*, 2014.

[136] Christopher H. Lin, Mausam, and Daniel S. Weld. Re-active learning: Active learning with relabeling. In *AAAI*, 2016.

[137] C. Lintott, K. Schawinski, S. Bamford, A. Slosar, K. Land, D. Thomas, E. Edmondson, K. Masters, R. C. Nichol, M. J. Raddick, A. Szalay, D. Andreescu, P. Murray, and J. Vandenberg. Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies. *MNRAS*, 410:166–178, January 2011.

[138] Chris Lintott, Kevin Schawinski, Steven Bamford, Ane Slosar, Kate Land, Daniel Thomas, Edd Edmondson, Karen Masters, Robert C. Nichol, M. Jordan Raddick, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg. Galaxy zoo 1: data release of morphological classifications for nearly 900 000 galaxies. *Monthly Notices of the Royal Astronomical Society*, 410(1):166–178, 2011.

[139] Chris Lintott, Kevin Schawinski, Anze Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan VandenBerg. Galaxy zoo : Morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *MNRAS*, 389(3):1179–1189, 2008.

[140] Chris J. Lintott, Kevin Schawinski, Ane Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg. Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 389(3):1179–1189, 2008.

[141] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *KDD-HCOMP*, pages 29–30, 2009.

[142] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: human computation algorithms on mechanical turk. In *UIST*, pages 57–66, 2010.

[143] Michael L. Littman, Thomas Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *UAI*, pages 394–402, 1995.

[144] Angli Liu, Stephen Soderland, Jonathan Bragg, Christopher H. Lin, Xiao Ling, and Daniel S. Weld. Effective crowd annotation for relation extraction. In *NAACL*, 2016.

[145] Qiang Liu, Jian Peng, and Alexander Ihler. Variational inference for crowdsourcing. In *NIPS*, 2012.

[146] Contact center in the cloud, December 2009. `http://liveops.com`.

[147] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[148] Bhaskara Marthi, Stuart Russell, David Latham, and Carlos Guestrin. Concurrent hierarchical reinforcement learning. In *IJCAI*, 2005.

[149] Winter Mason and Duncan J. Watts. Financial incentives and the "performance of crowds". In *HCOMP: Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 77–85, 2009.

[150] Mausam, Emmanuel Benazera, Ronen I. Brafman, Nicolas Meuleau, and Eric A. Hansen. Planning with continuous resources in stochastic domains. In *IJCAI*, pages 1244–1251, 2005.

[151] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan and Claypool, 2012.

[152] Mechanical turk, December 2011. `http://www.mturk.com/mturk/welcome`.

[153] Katharine Mieszkowski. I make $1.45 a week and i love it, July 2006. `http://www.salon.com/technology/feature/2006/07/24/turks`.

[154] George A. Miller and William G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1 – 28, 1991.

[155] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the ACM Conference on Information and Knowledge Management*, 2008.

[156] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.

[157] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[158] Integrate human intelligence into your software, 2011. `http://mobileworks.com/`.

[159] Barzan Mozafari, Purnamrita Sarkar, Michael J. Franklin, Michael I. Jordan, and Samuel Madden. Active learning for crowd-sourced databases. *CoRR*, abs/1209.3686, 2012.

[160] Ahuva Mu'alem and Michael Schapira. Mechanism design over discrete domains. In *EC: Proceedings of the 9th ACM conference on Electronic commerce*, pages 31–37, 2008.

[161] Nagarajan Natarajan, Inderjit S. Dhillon, and Pradeep Ravikumar. Learning with noisy labels. In *NIPS*, 2013.

[162] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[163] John Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: Crowdsourcing nutrition analysis from food photographs. In *UIST*, 2011.

[164] Aditya Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: Algorithms for filtering data with humans. In *VLDB*, 2010.

[165] Ronald Parr and Stuart Russell. Reinforcement learning with hierarachies of machines. In *NIPS*, 1998.

[166] Genevieve Patterson and James Hays. Coco attributes: Attributes for people, animals, and objects. *European Conference on Computer Vision*, 2016.

[167] Genevieve Patterson, Grant Van Horn, Serge Belongie, Pietro Perona, and James Hays. Tropel: Crowdsourcing detectors with minimal training. In *HCOMP*, 2016.

[168] 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.

[169] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, 2003.

[170] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *J. Artificial Intelligence Research*, 27(1):335–380, 2006.

[171] Jervis Pinto, Alan Fern, Tim Bauer, and Martin Erwig. Robust learning for adaptive programs by leveraging program structure. In *ICMLA*, 2010.

[172] Jakub Piskorski and Roman Yangarber. *Information Extraction: Past, Present and Future*, pages 23–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[173] Josep M. Porta, Matthijs T. J. Spaan, and Nikos A. Vlassis. Robot planning in partially observable continuous domains. In *Proc. Robotics: Science and Systems*, pages 217–224, 2005.

[174] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *J. of Machine Learning Research*, 7:2329–2367, 2006.

[175] Shreya Rajpal, Karan Goel, and Mausam. Pomdp-based worker pool selection for crowdsourcing. In *Workshop on Crowdsourcing and Machine Learning at ICML (CrowdML)*, 2015.

[176] Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics*, 2011.

[177] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, and Gerardo Valadez. Learning from crowds. *Journal of Machine Learning Research*, 11:1297–1322, 2010.

[178] Daniela Retelny, Sébastien Robaszkiewicz, Alexandra To, Walter S. Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, and Michael S. Bernstein. Expert crowdsourcing with flash teams. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 75–85, New York, NY, USA, 2014. ACM.

[179] A. Kimball Romney, Susan C. Weller, and William H. Batchelder. Culture as consensus: A theory of culture and informant accuracy. *American Anthropologist*, 88(2):313 – 338, 1986.

[180] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers? shifting demographics in mechanical turk. In *CHI*, 2010.

[181] Stephane Ross, Brahim Chalb-draa, and Joelle Pineau. Bayes-adaptive POMDPs. In *NIPS*, 2008.

[182] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *ICML*, 2001.

[183] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[184] Stuart Russell and Eric Wefald. *Do the Right Thing*. MIT Press, Cambridge, MA, 1991.

[185] Sivan Sabato and Adam Kalai. Feature multi-selection among subjective features. In *ICML*, 2013.

[186] Adam Sadilek, Sean Brennan, Henry Kautz, and Vincent Silenzio. nemesis: Which restaurants should you avoid today? In *HCOMP*, 2013.

[187] E. Tjong Kim Sang and F. De Meulder. Introduction to the CoNNL-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL*, 2003.

[188] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. Technical report, NICTA and the Australian National University, 2011.

[189] D. Sculley, Matthew Eric Otey, Michael Pohl, Bridget Spitznagel, John Hainsworth, and Yunkai Zhou. Detecting adversarial advertisements in the wild. In *KDD*, 2011.

[190] Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[191] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *NIPS*, 2007.

[192] Dafna Shahaf and Eric Horvitz. Generlized markets for human and machine computation. In *AAAI*, 2010.

[193] Dafna Shahaf and Eric Horvitz. Generlized markets for human and machine computation. In *AAAI*, 2010.

[194] Guy Shani, Ronen I. Brafman, and Solomon E. Shimony. Model-based online learning of pomdps. In *ECML*, pages 353–364, 2005.

[195] Guy Shani, Ronen I. Brafman, and Solomon Eyal Shimony. Prioritizing point-based pomdp solvers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(6):1592–1605, 2008.

[196] Guy Shani, David Heckerman, and Ronen I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

[197] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[198] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[199] Aashish Sheshadri and Matthew Lease. SQUARE: A Benchmark for Research on Computing Crowd Consensus. In *Proceedings of the 1st AAAI Conference on Human Computation (HCOMP)*, pages 156–164, 2013.

[200] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, 2010.

[201] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of IJCAI*, pages 1080–1087, 1995.

[202] Christopher Simpkins, Sooraj Bhat, Charles Isbell Jr., and Michael Mateas. Towards adaptive programming: Integrating reinforcement learning into a programming language. In *OOPSLA*, 2008.

[203] Trey Smith. Zmdp software for pomdp and mdp planning.

[204] Trey Smith and Reid G. Simmons. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *AAAI*, 2006.

[205] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and A. Ng. Cheap and fast — but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP'08*, 2008.

[206] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, pages 254–263, 2008.

[207] Edward J. Sondik. *The Optimal Control of Partially Observable markov Processes*. PhD thesis, Stanford, 1971.

[208] Alexander Sorokin and David Forsyth. Utility data annotation with amazon mechanical turk. In *Computer Vision and Pattern Recognition Workshop*, pages 1–8, 2008.

[209] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[210] Speechink. `http://speechink.com`.

[211] Yu-An Sun and Christopher R. Dance. When majority voting fails: Comparing quality assurance methods for noisy human computation environment. *CoRR*, abs/1204.3516, 2012.

[212] Yu-An Sun, Shourya Roy, and Greg Little. Beyond independent agreement: A tournament selection approach for quality assurance of human computation tasks. In *Human Computation*, 2011.

[213] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[214] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning I: Introduction*. The MIT Press, 1998.

[215] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 1999.

[216] Arjun Talwar, Radu Jurca, and Boi Faltings. Understanding user behavior in online feedback reporting. In *EC'1: Proceedings of the 8th ACM conference on Electronic commerce*, pages 134–142, 2007.

[217] Brian Tanner and Adam White. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, 2009.

[218] Taskcn, March 2010. `http://www.taskcn.com/`.

[219] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[220] Katrin Tomanek and Udo Hahn. Reducing class imbalance during active learning for named entity annotation. In *K-CAP*, 2009.

[221] Michael Toomim, Xianhang Zhang, James Fogarty, and James A. Landay. Access control by testing for shared knowledge. In *CHI: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 193–196, 2008.

[222] Topcoder, December 2009. `http://topcoder.com`.

[223] Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R. Jennings. Efficient crowdsourcing of unknown experts using multi-armed bandits. In *ECAI*, pages 768–773, 2012.

[224] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes. 2016.

[225] Turker nation, 2010. `http://turkers.proboards.com/index.cgi?`

[226] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[227] Vladimir N. Vapnik and Alexey YA. Chervonenkis. On the uniform converge of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[228] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[229] Sudheendra Vijayanarasimhan and Kristen Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. In *CVPR*, 2011.

[230] Luis von Ahn. Games with a purpose. *IEEE Computer Magazine*, pages 96–98, June 2006.

[231] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI*, 2004.

[232] Byron C. Wallace, Kevin Small, Carla E. Brodley, and Thomas A. Trikalinos. Active learning for biomedical citation screening. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 173–182, New York, NY, USA, 2010. ACM.

[233] William Yang Wang, Dan Bohus, Ece Kamar, and Eric Horvitz. Crowdsourcing the acquisition of natural language corpora: Methods and observations. In *Spoken Language Technology Workshop*, 2012.

[234] Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *ACL*, 2015.

[235] Tamsyn P. Waterhouse. Pay by the bit: an information-theoretic metric for collective human judgment. In *CSCW*, pages 623–638, 2013.

[236] Fabian L. Wauthier and Michael I. Jordan. Bayesian bias mitigation for crowdsourcing. In *NIPS*, 2011.

[237] Gary M. Weiss and Foster Provost. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[238] Daniel S. Weld, Mausam, and Peng Dai. Execution control for crowdsourcing. In *UIST*, 2011.

[239] Daniel S. Weld, Mausam, and Peng Dai. Human intelligence needs artificial intelligence. In *HCOMP*, 2011.

[240] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *NIPS*, 2010.

[241] Keenon Werling, Arun Chaganty, Percy Liang, and Christopher D. Manning. On-the-job learning with bayesian decision theory. In *NIPS*, 2015.

[242] Jacob Whitehill, Paul Ruvolo, Jacob Bergsma, Tingfan Wu, and Javier Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, 2009.

[243] Incentive. http://en.wikipedia.org/wiki/Incentive.

[244] Crowdsourcing, December 2011. `http://en.wikipedia.org/wiki/Crowdsourcing`.

[245] Jason D. Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Comput. Speech Lang.*, 21(2):393–422, 2007.

[246] Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G. Dy. Active learning from crowds. In *ICML*, 2011.

[247] Jiang Yang, Lada A. Adamic, and Mark S. Ackerman. Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *EC: Proceedings of the 9th ACM conference on Electronic commerce*, pages 246–255, 2008.

[248] Hakan L. S. Younes and Michael L. Littman. Ppddl1.0: The language for the probabilistic part of ipc-4. In *IPC*, 2004.

[249] Chicheng Zhang and Kamalika Chaudhuri. Active learning from weak and strong labelers. In *NIPS*, 2015.

[250] Congle Zhang, Stephen Soderland, and Daniel S. Weld. Exploiting parallel news streams for unsupervised event extraction. In *TACL*, 2015.

[251] Haoqi Zhang, Eric Horvitz, Yiling Chen, and David C. Parkes. Task routing for prediction tasks. In *AAMAS*, pages 889–896, 2012.

[252] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Yin and yang: Balancing and answering binary visual questions. In *CVPR*, 2016.

[253] Liyue Zhao, Gita Sukthankar, and Rahul Sukthankar. Incremental relabeling for active learning with noisy crowdsourced annotations. In *IEEE Conference on Social Computing*, 2011.

[254] Dengyong Zhou, Qiang Liu, John C. Platt, and Christopher Meek. Aggregating ordinal labels from crowds by minimax conditional entropy. In *ICML*, 2014.

[255] Jingbo Zhu and Eduard Hovy. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *EMNLP*, 2007.