Machine Learning from Human Preferences

Machine Learning from Human Preferences

Sang T. Truong, and Sanmi Koyejo

The MIT Press
Cambridge, Massachusetts
London, England

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in ETbb and Source Sans Pro by _____. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data is available.

ISBN: 978-0-262-55256-1

10 9 8 7 6 5 4 3 2 1

Table of Contents

1 Introduction

Machine learning is increasingly integrated into many aspects of our lives through various applications, such as healthcare, education, and scientific discovery. A key challenge in developing trustworthy intelligent agents that benefit humanity is ensuring they align with human values. Learning from human preferences offers a promising approach to accomplishing this goal. This book presents the fundamental foundations and practical applications of machine learning from human preferences. It also covers topics from related fields, such as economics, psychology, and human-computer interaction. The goal is to equip readers with the concepts and tools required to build artificial intelligence systems aligned with human values.

This book explores the challenge of defining goals and preferences in traditional machine-learning approaches by introducing the paradigm of learning from human preference. This paradigm uses human feedback to guide the learning process and overcome the limitations of manually specified goal functions.

Human feedback – whether from individuals' preferences, judgments, ratings, or other responses – plays a pivotal role in guiding AI agents through their learning journeys across various tasks and domains. The following three examples show the importance of human feedback to different AI agents. First, human feedback could guide AI agents in creating appealing and diverse content by assessing the quality, originality, and relevance of the content. Second, human feedback could also ensure that AI agents align with human needs and values by rectifying potential biases, errors, and harm caused by agents. Third, human feedback could help AI agents learn better policies in complex environments, such as those with sparse or noisy rewards, by encouraging agents to explore and learn in those environments.

This book covers various topics, from the statistical foundations and strategies for interactively querying humans to applications for eliciting information to improve learning. In more detail, we focus on the three most important aspects.

- The role of the human-in-the-loop for improving learning systems: We review the relevant foundations in microeconomics, psychology, marketing, statistics, and other disciplines and explore their applications to various domains, such as language, robotics, logistics, and more. We adopt the machine learning perspective for modeling, estimating, and evaluating human-in-the-loop learning processes.
- The characteristics and challenges of human questions: We examine the issues of bias, correctness, noisiness, rationality, and other factors that affect the quality and reliability

of human responses. We also consider the differences and similarities between individual and group responses and how they influence our approach to human-in-the-loop learning.

– The ethical implications of human-in-the-loop learning: We discuss the potential benefits and risks of learning from human preferences, opinions, and behaviors and how to address them responsibly and fairly. We also raise questions about the selection, representation, and protection of human participants and the possible consequences of exploiting or manipulating human responses.

Besides the above aspects, we also touch upon some other relevant topics, such as:

– General Artificial Intelligence: Most machine learning or AI models/algorithms involve learning from humans, as the ultimate goal is often to imitate human intelligence. Therefore, humans are the primary source of data and feedback for ML/AI systems.

– General Machine Learning: Humans define all the steps of the ML/AI process, such as selecting the problem, data sources, model architectures, optimization methods, and evaluation metrics. Therefore, humans are the main decision-makers and stakeholders for ML/AI systems.

– Expert knowledge for defining model architectures: In some cases, humans can provide valuable domain knowledge and prior information for designing and refining model architectures, especially for graphical models and causal inference. We may present a few examples, but this is not our main focus.

– Human-Computer Interaction (HCI): The interface and elicitation process matters for the quality and efficiency of human-in-the-loop learning. Therefore, HCI principles and techniques can help to improve the user experience and engagement of human participants.

Returning to human feedback, their integration can occur at various stages of the learning process, spanning from data collection and labeling to model selection, training, and evaluation. Incorporating these feedbacks enables fine-tuning the model to align with their hidden insights. The utilization of human feedback is crucial due to its ability to offer valuable signals that are challenging to acquire or delineate through other means, such as data or predefined cost functions. There are many ways to update models based on human feedback, depending on the type and level of feedback and the objective and structure of the model. A general taxonomy of feedback-update interactions can be divided into six categories:

– Observation-level & Active data collection & Asking humans for feedback on specific features, such as collecting expert labels on features.

– Observation-level & Constraint elicitation & Inferring optimization constraints from insights extracted from feedback.

– Observation-level & Feature modification & Adding, removing, or preprocessing features of training/finetuning datasets.

- Domain-level & Dataset modification & Generating synthetic data that satisfy certain constraints specified by human feedback, such as fairness or weak supervision.
- Domain-level & Constraint specification & Modifying the loss function for optimizing the model based on human feedback, such as imposing fairness, interpretability, or resource constraints.
- Domain-level & Model editing & Changing the rules or weights of the model based on human feedback, such as incorporating domain knowledge or preferences.

These ways of updating models based on human feedback can help to improve the performance, behavior, and alignment of the models with human values and goals. However, they pose challenges and risks, such as communication barriers, feedback quality, and ethical issues. Therefore, designing and evaluating the feedback-update interactions carefully and responsibly is important.

Starting with an introduction to human preferences models and various approaches to modeling and understanding human preferences, the book then delves into interaction models that enable machines to learn from human preferences and feedback, including techniques such as paired comparison data analysis and game-theoretic perspectives on preference learning. Understanding human biases and incorporating them into reward models is explored in the chapter on human biases and reward models. The impact of human biases on reward inference and approaches to leverage both rational and irrational human behavior are discussed. Metric elicitation techniques are then introduced, which allow machines to learn performance metrics from pairwise comparisons and other forms of human feedback.

Active learning strategies are covered in the chapter on active learning, which enables machines to actively query humans for feedback and preferences to improve the learning process. The book also explores the design and development of adaptive user interfaces that personalize services based on user preferences, as well as the role of bandit algorithms and probabilistic methods in learning from human preferences.

Challenges and techniques involved in learning multimodal rewards and meta-reward learning are discussed, including approaches to learning from demonstrations and rewards in complex environments. Human-Computer Interaction (HCI) considerations in learning from humans are explored, emphasizing user-centered design principles. The alignment of goals and preferences among expert and non-expert stakeholders and the challenges and techniques involved are addressed.

Ensuring truthfulness and fairness in eliciting human preferences is crucial, and the book discusses mechanism design principles and techniques to incentivize truthful feedback from humans. The integration of human computing techniques in learning from human preference is also explored, highlighting the use of human intelligence to solve complex problems and improve machine learning algorithms.

The application of inverse reinforcement learning in robotics focuses on how machines can infer human preferences and reward functions from observed behavior. Ethical considerations

in learning from human preference are addressed, emphasizing the importance of incorporating ethical principles in designing and deploying machine learning systems. Finally, the book explores reinforcement learning from human feedback for language models, highlighting techniques for incorporating human feedback in training language models.

The book aims to provide a comprehensive overview of machine learning from human preference. By leveraging human feedback and preferences, the aim is to develop more intelligent and reliable machine-learning systems that align with human values and preferences. The book is divided into 16 chapters:

- **Chapter 1** is an introductory chapter providing an overview of the field and motivations and outlines what will be covered.
- **Chapter 2** provides an integrated framework for understanding human preference modeling, interaction models, and the impact of human biases on decision-making. It begins by exploring the motivations and applications of human preference modeling, using examples from health coaching, social media, and shopping. The chapter then discusses various rationality assumptions and traditional models such as Luce's axiom of choice and Boltzmann Rationality, highlighting their roles in capturing the probabilistic nature of human choices. It also addresses advanced interaction models using pairwise and rank-order sampling techniques to analyze and predict preferences, alongside a case study on the LESS model for handling duplicates in decision-making scenarios. Finally, it delves into the ethical and practical challenges of collecting and utilizing human feedback to ensure robust and well-calibrated reward models.
- **Chapter 5** introduces a framework for eliciting multi-class performance metrics from an oracle through pairwise comparisons of confusion matrices. It describes eliciting linear metrics that consider only the diagonal elements of confusion matrices, representing correct predictions. Such metrics are known as Diagonal Linear Performance Metrics (DLPMs). The chapter outlines an algorithm for eliciting DLPMs by finding the Bayes optimal confusion matrix that maximizes a DLPM through binary searches of the space of possible confusion matrices.
- **Chapter 6** discusses different methods for active learning, with a focus on selecting training examples that maximize improvement to the learner's performance. It describes how active learning aims to strategically query new data points by estimating how their addition would hypothetically impact the model if trained on them. Various strategies are explored, including reducing the learner's variance, exploiting ambiguity and domain knowledge in ranking and comparisons, and balancing exploration versus exploitation. Computational methods are presented and analyzed empirically on applications like robotics to demonstrate how active learning can enhance models using significantly less labeled data.
- **Chapter 7** discusses adaptive user interfaces, which aim to provide personalized experiences by learning individual user preferences from interactions. It presents the design of adaptive interfaces as involving modeling users, collecting user traces, learning models from the data, and applying the models to adapt recommendations. The applications of

adaptive interfaces mentioned include route advisors, destination selection assistants, and scheduling tools, with the goal of improving systems through intelligent personalization.

- **Chapter 8** discusses different bandit algorithms and their applications. It introduces the multi-armed bandit problem and explores strategies like epsilon-greedy and UCB to balance exploration and exploitation. Two important extensions are examined more thoroughly: contextual bandits, which incorporate context into decisions, and dueling bandits, which learn from pairwise preferences instead of explicit rewards. A wide range of domains are also presented where bandit methods, such as healthcare, recommendations, and dialogue systems, have proved useful.
- **Chapter 9** examines modeling human rewards that have complex, multi-modal structures and techniques for meta-learning reward functions.
- **Chapter 10** analyzes important human-computer interaction considerations for systems that learn from humans, like cognitive constraints and user experience.
- **Chapter 11** tackles challenges around aligning learned models with values from diverse expert and non-expert stakeholders. Issues of truthfulness and the notion of agreement are discussed.
- **Chapter 12** focuses on mechanism design theory and how it can be applied to develop protocols and systems for truthfully learning preferences at scale.
- **Chapter 13** looks at how human computation frameworks can enable large-scale preference elicitation by crowd-sourcing tasks to many individuals.
- **Chapter 14** presents applications of inverse reinforcement learning using human feedback for robotics, such as learning helicopter control policies from demonstrations.
- **Chapter 15** discusses ethical issues that arise in interaction models and approaches for designing preference elicitation systems considering fairness, privacy, and other socio-technical factors.
- **Chapter 16** covers reinforcement learning techniques that can leverage human feedback to guide language models, for example, by providing feedback on the generated text.

Machine learning from human feedback, especially reinforcement learning from human feedback (RLHF), stands as a promising avenue for training AI systems through human input. However, it confronts several intricate challenges and its efficacy encounters notable limitations stemming from the intricacies of human feedback and the complexity of aligning AI with human values.

The acquisition of representative and unbiased feedback from humans presents a formidable hurdle, rooted in inherent limitations of human evaluators. Human fallibility and the incapacity to assess ML/AI model's output accurately hinder the quality and reliability of feedback. Moreover, there exists an inherent tradeoff between the efficiency and richness of feedback. While extensive, detailed feedback such as prolonged conversations promises deeper insights, its acquisition proves arduous and resource-intensive.

Within the domain of RLHF, the construction of a comprehensive reward model poses significant difficulties. Capturing the intricacies of complex and context-dependent human values and preferences within a singular reward function stands as a formidable challenge. The inherent inconsistency in modeling human behavior further complicates this endeavor. Consequently, reward models are susceptible to misgeneralization, resulting in imperfect proxies that pave the way for "reward hacking". Agents may veer towards optimizing these flawed proxies rather than pursuing the genuine objectives intended by human feedback.

The optimization of policies within RLHF presents its own set of challenges. Effectively fine-tuning policies through RL techniques encounters obstacles, notably susceptibility to adversarial exploitations. Furthermore, even if training rewards are accurately derived, policies may exhibit poor performance upon deployment due to discrepancies between the training and deployment distributions. Agents might prioritize maximizing their influence or power, diverging from the intended goals outlined by the feedback.

In the realm of joint training, where reward models and policies undergo simultaneous refinement, intricate issues surface. The amalgamation of these components can induce detrimental distributional shifts as errors accumulate throughout the training process. Balancing training efficiency while circumventing overfitting proves to be a complex undertaking. Policies exploring areas where the reward model exhibits inaccuracies further complicate the delicate balance between efficient learning and avoiding overfitting. These challenges underscore the intricate landscape of RLHF, necessitating nuanced strategies and innovative approaches to surmount the complexities inherent in aligning AI systems with human feedback effectively.

Looking forward, future developments in RLHF necessitate a nuanced approach. Enhancements in human feedback processes, potentially leveraging AI assistance, fine-grained annotations, and demonstrative techniques, hold promise in ameliorating feedback quality. Moreover, addressing the challenges of modeling uncertainty and handling discrepancies in reward models emerges as a crucial area for improvement. Integrating RLHF with complementary techniques, such as formal verification and interpretability, offers avenues to bolster its effectiveness. Moreover, a pivotal direction lies in broadening the scope of RLHF beyond singular reward frameworks to accommodate the oversight of diverse stakeholder objectives. Embracing multi-objective oversight is pivotal to authentically representing the multifaceted goals of varied stakeholders within AI systems. Simultaneously, ensuring public transparency concerning technical intricacies fosters a better understanding of strengths, limitations, and the developmental trajectory of RLHF.

However, it is imperative to underscore that RLHF should not be perceived as a comprehensive solution but rather as a facet within a comprehensive "defense in depth" strategy integrating multiple safety measures. The progress of RLHF and broader advancements in AI alignment demand persistent efforts to navigate fundamental choices and challenges inherent in aligning AI systems with human values and goals.

The book is intended for researchers, practitioners, and students who are interested in the intersection of machine learning, human-computer interaction, and artificial intelligence. The

book assumes some basic knowledge of probability, statistics, and machine learning, but provides sufficient background and references for the readers to follow the main ideas and results. The book also provides code examples and datasets for some of the methods and applications discussed in the book. The field of machine learning from human preference is a vibrant and growing area of research and practice, with many open challenges and opportunities. We hope that this book will inspire and inform the readers to further explore and advance this exciting and important field.

2 Human Decision Making and Choice Models

2.1 Introduction

Human preference modeling aims to capture humans' decision making processes in a probabilistic framework. Many problems would benefit from a quantitative perspective, enabling an understanding of how humans engage with the world. While human decision-making is only somewhat understood, we can use real-world data representing the outcomes of decisions to align human-facing systems with user preferences. Through our exploration of human preference models, we will ground ourselves in building a health coaching system that can provide meal recommendations aligned with a user's dietary needs and preferences. Examples of scenarios which can benefit from a model of how humans make choices include:

1. **Health coaching**: Humans express their preferences every time they pick lunch for consumption. Humans may have several goals related to nutrition, such as weight loss and improving concentration. We can learn how a given individual or set of individuals prefer to eat to provide personalized recommendations to help them attain their goals. This chapter will use this use case to ground human preference modeling in a real-life application.
2. **Social media**: Platforms have a far greater amount of content than one can consume in a lifetime, yet such products must aim to maximize user engagement. To accomplish this, we can learn what specific things people like to see in their feeds to optimize the value they gain out of their time on social media. For example, the video feed social media platform TikTok[1] has had viral adoption due to its notorious ability to personalize a feed for its users based on their preferences.
3. **Shopping**: Retail corporations largely aim to maximize revenue by making it easy for people to make purchases. Recommendation systems on online shopping platforms provide a mechanism for curating specific items based on an individual's previous purchases (or even browsing history) to make shoppers aware of items they may like and, therefore, purchase.

[1] https://www.tiktok.com/

Table 2.1
Examples of machine learning tasks and their interpretation as modeling human preferences.

Application	Human Preference
Computer vision: train a neural network to predict bounding boxes delineating all instances of dogs in an image	This is how humans process images by identifying the position and geometry of the things we see in them
Natural language processing: train a model to generate coherent text	Coherent text is itself a human-created and defined concept, and we prefer that any synthetically generated text matches that of humans
Computer vision: train a diffusion model to generate realistic images of nature	Humans prefer that images accurately capture the world as observed by humans, and this generative model should reflect the details that comprise that preference

In this chapter, we will explore how one can model human preferences, including different formulations of such models, how one can optimize these models given data, and considerations one must understand to create such systems. We note that the exact assumptions we make about human preferences in this chapter differentiate the *specific* human preference learning problem we are considering from the discriminative and generative tasks we describe in Table 2.1. We describe these assumptions in Section 2.2.

2.2 Foundations of Preference Models

We introduce a framework for discussing human preferences. The different methods to model these preferences Section 2.3 all build upon this framework.

Axiom 1: Preference models model choice

Human preference models model the preferred choice or choices amongst a set of options. In our health coaching example, this could be modeling which meal from a set of options a person will most likely choose. An alternative framework we will explore is ranking, in which we can model an ordering of given choices from most to least desirable. It is certainly possible that there is an infinite set of options (such as in a continuous action space); in this case, our model will have to reason about a discretized set of options and may fail to capture the full space of possibilities a human would choose from in the real world.

Choices are *collectively exhaustive*, *mutually exclusive*, and *finite*. Human preference models must enumerate an *action space*, or the set of all possible choices included in a human decision. As such, we must ensure that the choices we enumerate capture the entire domain (collectively exhaustive) but are indeed distinct (mutually exclusive) choices. In our health coaching example, a person either chooses to eat chicken or fish. Choosing one does not affect the other.

A discrete set of choices is a constraint we canonically impose to ensure we can tractably model preferences and aptly estimate the parameters of preference models. This is usually sufficiently expressive to create a powerful human preference model (for example, recent generative language models have vocabulary sizes of 40,000+ and can model nearly arbitrary language sequences (Radford et al. 2018)). While in theory, one can imagine a continuous domain for choices, a discrete set fits nicely with most decision-making processes humans face. While human thought is extremely nuanced, most thoughts are expressed as discrete words or discrete decisions in every step humans take in the world.

Axiom 2: Preference captures decision-making

There are certainly cases in which human preferences don't reflect the human decision-making process, for example if there are external factors (social, political, economic) which govern a human's choices, or if one is explicitly choosing to go against their preferences in the context of exploration. However, human preference models will always do their best to model the ultimate decision, and we assume that they are in some way accounting for these other factors (and any lack of such accounting will result in a biased model). Human preferences are generally classified into two categories:

1. Revealed preferences are those one can observe retroactively from existing data. The implicit decision-making knowledge can be captured via learnable parameters and their usage in models which represent relationships between input decision attributes that may have little human interpretability, but enable powerful models of human preference. For health coaching, we may have information about which foods an individual has chosen previously in different contexts, allowing us to build a model from their decisions. Such data may be easier to acquire and can reflect real-world outcomes (since they are, at least theoretically, inherently based on human preferences). However, if we fail to capture sufficient context in such data, human preference models may not sufficiently capture human preferences.
2. Stated preferences are those individuals explicitly indicate in potentially experimental conditions. The explicit knowledge may be leveraged by including inductive biases during modeling (for example, the context used in a model) which are reasonable assumptions for how a human would consider a set of options.This may include controlled experiments or studies. This may be harder to obtain and somewhat biased, as they can be hypothetical or only accurately reflect a piece of the overall context of a decision. However, they enable greater control of the decision-making process.

Human Rationality

Modeling decision-making must also take into account the rational and irrational behaviour of humans. Therefore we consider *rationality assumptions* as a fundamental aspect of understanding how individuals make decisions. These assumptions provide a framework for predicting and

modeling human behavior by outlining the principles that guide decision-making processes (Keisler and Lee 2003).

Perfect rationality posits that individuals always make decisions that maximize their utility. It assumes that individuals have complete information and the cognitive ability to process this information to make optimal choices (Miljkovic 2005). This assumption is often used in economic models to predict how rational agents would behave under ideal conditions. However, numerous studies have shown that this assumption frequently fails to describe actual human behavior, as individuals do not always act in ways that maximize their utility due to various constraints and biases (Miljkovic 2005). Bounded rationality, on the other hand, acknowledges that individuals operate within the limits of their information and cognitive capabilities. Decisions are made using heuristics or rules of thumb rather than through exhaustive analysis, reflecting the practical constraints of real-world decision-making (Simon 1972). This concept, introduced by Herbert Simon, recognizes the limitations of human cognitive processing and the impact of these limitations on decision-making. Simon's theory suggests that instead of optimizing, individuals satisfy, seeking solutions or decisions that are "good enough" under the circumstances (Simon 1972). Noisy rationality assumes that decisions are influenced by random noise, resulting in probabilistic choice behavior. This means that while individuals aim to maximize their utility, random factors can lead to deviations from perfectly rational choices. This approach is useful for modeling behavior in situations where decisions are not entirely deterministic and are subject to variability (Miljkovic 2005). This probabilistic approach aligns with findings from behavioral economics and psychology, which indicate that human decision-making is often inconsistent and influenced by various random factors (Miljkovic 2005).

Understanding rationality assumptions is crucial for modeling and predicting human behavior in various decision-making scenarios. These assumptions provide the foundation for developing models that can simulate and analyze how individuals interact with one another and their environment. By incorporating different types of rationality, researchers can create more accurate and realistic models that reflect the complexities of human decision-making. This comprehensive approach enhances the predictive power of models and improves the understanding of human behavior in economic and social contexts (Miljkovic 2005; Simon 1972).

Luce's axiom of choice (Luce 1977) and Boltzmann's Rationality provide a probabilistic framework for modeling noisily-rational human behavior. Luce's axiom of choice addresses the likelihood of a human selecting an option o from a set O. Desirability is represented by a value function $v : O \to \mathbb{R}^+$, with the selection probability calculated as $P(o) = \frac{v(o)}{\sum_{o' \in O} v(o')}$. Assuming there is an underlying reward for each option $R(o) \in \mathbb{R}$ such that $v(o) = e^{R(o)}$, we get $P(o) = \frac{e^{R(o)}}{\sum_{\bar{o} \in \mathcal{O}} e^{R(\bar{o})}}$. Essentially, "A human will act out a trajectory with a probability proportional to the exponentiated return they receive for the trajectory." This probabilistic approach challenges the traditional assumption of perfect economic rationality, where individuals always make decisions that maximize their utility. When choices involve trajectories $\xi \in \Xi$ (sequences of actions), the Boltzmann model (Neumann and Morgenstern 1945) is used. Here, the reward R is typically a function of a feature vector $\phi : \Xi \to \mathbb{R}^k$, and the probability density is given by $p(\xi) = \frac{e^{R(\phi(\xi))}}{\int_{\Xi} e^{R(\phi(\bar{\xi}))} d\bar{\xi}}$. Boltzmann Rationality serves a critical role in

human preferences and decision-making. It captures the probabilistic nature of human choices, recognizing that decisions are often noisy and influenced by various factors. This model is instrumental in preference modeling, accommodating human preferences' inherent variability and uncertainty.

However, the Luce choice axiom and Boltzmann Rationality encounter a known issue called the "duplicates problem," where there is no concept of similar actions (e.g., choosing between using a car or a train for transportation, with no particular preference). The probability of making the decision is 50% for either option. However, if we now have 100 cars, under Luce/Boltzmann, we would have a 99% probability of choosing a car, which is unrealistic.

To address this issue, various extensions have been proposed. One such extension is the attribute rule, which interprets options as bundles of attributes. In this rule, attributes X are associated with options, and they have desirability values $w(x)$. An attribute intensity function $s(x, o)$ indicates the degree to which an attribute is expressed in an option. The probability of choosing option o is calculated as:

$$P(o) = \sum_{x \in X_o} \frac{w(x)}{\sum_{\bar{x} \in X_o} w(\bar{x})} \cdot \frac{s(x, o)}{\sum_{\tilde{o} \in \mathcal{O}} s(x, \bar{o})}$$

This equation describes a two-step process where an attribute $x \in X_O$ is first chosen according to a Luce-like rule and then an option $o \in O$ with that attribute is selected using another Luce-like rule. This approach handles duplicates gracefully by effectively creating a two-layer hierarchy in choosing an option.

Boltzmann Rationality finds practical applications in various fields, particularly in reinforcement learning, where it models decision-making in uncertain environments. It also applies to trajectory selection, where the probability of a sequence of actions (trajectory) is proportional to the exponential return. These applications enhance the accuracy of models that interact with or predict human behavior, making Boltzmann Rationality a vital component of the models of interaction.

We next explore a case study to deepen our understanding of rationality: Limiting Errors due to Similar Selection (LESS) (Bobu et al. 2020). LESS takes inspiration from the attribute rule and extends it to continuous trajectories (Bobu et al. 2020). The key insight is that instead of creating "attributes", which group together similar discrete options, it introduces a similarity metric on the space of continuous actions, thereby creating similar groupings on trajectories.

First, discussing the distinction between trajectory and feature space is important. The LESS similarity metric could be defined in trajectory space, where the trajectory is some theoretical notion of all states and actions one passes through over time. However, it is instead defined on the measured feature vector $\phi(\xi)$ associated with the agent's trajectory ξ. Why? In practice, one can never measure the exact trajectory with perfect fidelity. The feature vector will almost necessarily map in a one-to-many fashion with trajectories. Formally, let $\phi \in \Phi$ be the set of all possible feature vectors $\xi \in \Xi$ the set of all trajectories. The set of feature vectors belonging

to a set of trajectories $\Xi' \subseteq \Xi$ is $\Phi_{\Xi'}$. We begin with equation (4) and substitute our similarity metric on feature vectors of trajectories.

$$P(\xi) = \frac{e^{R(\phi(\xi))}}{\sum_{\bar{\phi} \in \Phi_\Xi} e^{R(\hat{\phi})}} \cdot \frac{s(\phi(\xi), \bar{\xi})}{\sum_{\hat{\xi} \in \Xi} s(\phi(\xi), \bar{\xi})}$$

In this formulation, the first half of the product is simply Boltzmann equation. The probability of choosing trajectory ξ is proportional to the exponentiated reward for the agent's measured trajectory $\phi(\xi)$, normalized by the sum of all rewards over all possible measured trajectories. The second half of the product is a normalization factor based on how similar the current trajectory is to other trajectories in feature space. We can define the similarity function as an indicator function, where $s(x, \xi) = 1$ only if $x = \phi(\xi)$. That means that multiple trajectories with the same feature vector will effectively be considered a single option. Thus, we achieve the "bundling" of trajectories, in the same way that the attribute rule bundled options under different attributes.

However, setting the similarity metric as an indicator function isn't sufficiently flexible. We want a proper metric that acts more as a continuous distance over the feature space. We instead define s to be a *soft similarity metric* $s : \Phi \times \Xi \to \mathbb{R}^+$. It has the following properties:

1. $s(\phi(\xi), \xi) = \max_{x \in \phi, \bar{\xi} \in \Xi} s(x, \hat{\xi}))\forall(\xi \in \Xi)$
2. Symmetric: $s(\phi(\xi), \bar{\xi}) = s(\phi(\bar{\xi}), \xi)$
3. Positive Semidefinite: $s(x, \xi) \geq 0$

Using this redefined similarity metric s, we extend (5) to be a probability density on the continuous trajectory space $\mathcal{E}$, as in (3).

$$p(\hat{\xi}) = \frac{\frac{e^{R(\phi(\xi))}}{\int_\Xi s(\phi(\xi), \bar{\xi}) d\bar{\xi}}}{\int_\Xi \frac{e^{R(\phi(\hat{\xi}))}}{\int_\Xi s(\phi(\hat{\xi}), \bar{\xi}) d\bar{\xi}} d\hat{\xi}} \propto \frac{e^{R(\phi(\hat{\xi}))}}{\int_\Xi s(\phi(\xi), \bar{\xi}) d\bar{\xi}}$$

Under this formulation, the likelihood of selecting a trajectory is inversely proportional to its feature-space similarity with other trajectories. This de-weights similar trajectories, which is the desired effect for our LESS model of human decision-making. This means, though, that the "trajectory bundle" of similar trajectories still has a reasonable probability of being chosen.

Axiom 3: Preference centers around utility

Human preference models are centered around the notion of utility, which can mean a reward one attains after expressing one's preference over options.[2] In health coaching, the utility, as a function of the health choices users make, may be satiety, latent promotion of overall health,

[2] GPT-4 is good at coming up with longer-rendered answers about why some things are appropriate or not.

or even a quantitative extension of life. Of course, humans don't necessarily use an explicit measure of utility — frequently humans use qualitative factors such as emotion or external influence to make decision. However, we assume that the underlying utility mechanism of a human preference model still captures the final decision output from a human.

Utility can be interpreted as a scalar quantity representing the benefit or value an individual attains from selecting a given choice. Each choice has an associated utility. Human preference models capture both the utility of a choice (e.g. we model the utility value as a function of attributes of a given choice) and how the utilities interact to make a decision.[^2] We use the notation U_i as the utility corresponding to choice i.

1. **The utility of a choice is a stochastic function of the choice's attributes**. We will henceforth define utility as follows $U_i = H_i(z_i)$ where z_i is a variable describing the attributes of choice i and H_i is the stochastic function defining this choice's utility. As a simple example, we can use a 1-D linear stochastic function to define H_i: $U_i = H_i(z_i) = \beta z_i + \epsilon_i$, where β is a parameter of the model and ϵ_i is an unobserved factor for choice i. Generally, we assume that the ϵ_i factor is a random variable following a specified distribution, such as a standard normal distribution. The attributes we use to represent a choice (a single scalar value z_i in this example) is a critical design decision in defining the human preference model. These attributes define the context our model has in representing the human behavior we wish to capture, when choice i is made. In our health coaching example, we may hope to provide the best possible diet recommendations for an individual. However, if our vector representation z_i of their choice i does not include vital information, such as allergy risks associated to the choice or ingredients which make up the choice, our model may not have enough information to properly capture the human preference.

2. **The preferred choice is that whose corresponding utility is the largest**. Given that we model utility as the underlying benefit or value a human derives from choosing a given option, intuitively, we expect a human to choose the option with the largest utility. In our example of health coaching, if we model utility as the expected increase in lifespan, we will surely opt for the choice that maximizes this notion of utility. In our example, since $U_1 > U_2$, our model indicates that a user would opt for the burrito.

3. **Relativity of Utility**. Given the two previously defined characteristics of utility, we observe that only the relative difference in utility matters. Even if $U_1 = 0.001$ and $U_2 = 0.0005$, the model indicates the same outcome: a user prefers option 1. As such, even the scale of the utilities is irrelevant within a given set of human preference data for a given individual. In our example, we can scale the value of β without changing the overall outcome so long as we do not change the sign. The scale of utilities *is* important when comparing human preferences across datasets, or comparing the same model across different humans; since utility may be defined differently in various datasets, perhaps their exact values are not aligned in a manner which allows one to robustly compare preferences between them. A common practice to address this consideration is to standardize the utilities in each dataset based on its variance in the observed data. Furthermore, a human preference model may generate different scales of utilities across different humans

(based on the inputs and representation of the human). In this case, one can standardize the utilities for each individual based on the observed variance for that human. As we can see, the relativity of utility can be both powerful (enabling us to create flexible models and efficiently optimize them) and limiting (requiring us to perform mitigations when translating models across datasets or individuals. Still, we find the notion of utility necessary to model human preferences as it provides a quantitative value we can use to model human decisions.

As a concrete model of meal recommendation in health coaching, let us suppose that we have three choices:

1. A burrito with rice, beans, and cheese.
2. French fries covered in mayonnaise.
3. A rice bowl with beans and chicken.

If we design z_i to be 1D, for example:

1. $z_1 = 1$ for the burrito since this is a somewhat balanced meal that may help prolong the lifespan, which a user prefers.
2. $z_2 = -1$ since this is unhealthy due to being deep fried, including saturated fats, and potentially reducing lifespan.
3. $z_3 = 1$ since a rice bowl is another healthy meal.

After observing the choices of a user who likes to eat healthily, we might learn that $\beta = 1$ is the best parameter for this model, and maybe we assume that $\epsilon \sim \mathcal{N}(0, 1)$. Then, this model implies that $U_1 = 1 \cdot 1 + 0.03 = 1.03$, $U_2 = 1 \cdot -1 + (-0.07) = -1.07$, $U_3 = 1 \cdot 1 + (0.02) = 1.02$, which means that the user, for whom $\beta = 1$ is the learned parameter, they would prefer the first meal, with the third meal as a close second option.

If we design z_i to be 3D, to indicate the carbohydrate, protein, and fat content of each meal, then for example:

1. $z_1 = (1, 1, 0.1)$ for the burrito
2. $z_2 = (1, 0, 1)$ for the fries
3. $z_3 = (1, 1, 0.2)$ for the rice bowl.

After observing the choices of a user who likes to eat healthy, we might learn that $\beta = (1, 1, -1)$ is the best parameter for this model, and maybe we assume that $\epsilon \sim \mathcal{N}(0, 1)$. Then, this model implies that $U_1 = (1, 1, 0.1) \cdot (1, 1, -1) + 0.01 = 1.91$, $U_2 = (1, 0, 1) \cdot (1, 1, -1) + 0.03 = 0.03$, $U_3 = (1, 1, 0.2) \cdot (1, 1, -1) - 0.07 = 1.73$, which means that the user prefers meals 1 and 3, which again have the best utility, but in this multi-dimensional representation of z_i, we start understanding how the two preferred meals are related (low fat and high protein).

2.3 Models of Individual Choices

After exploring motivations for preference learning and the framework we use to characterize human preferences to enable modeling, we now expand on the common probabilistic methods used to model human preference tasks. We will instantiate these models for our real-world health coaching application throughout as a pedagogical example. Specifically, we can define the following domain for meal choices: $z_i, \beta \in \mathbb{Z}^3$, where z_i defines the representation of a meal option with the three dimensions representing the carbohydrate, protein, and lipid macronutrient content of the meal, respectively, all measured in grams. β is a parameter of the model. This simple representation will allow us to consider how different probabilistic frameworks for human preferences can model a user's meal preferences. The information representation we instantiate here can accommodate scalar and high-dimensional vectors. While we use a mixture of integer and real-valued vectors in this simple example, we refer the reader to code in the practicum section for an example where vectors are all real-valued. If we let $z = [20, 15, 3]$ and $\beta = [0.2, 1, -3]$. This corresponds to a meal with 20g carbohydrates, 15g protein, and 3g lipids. In the following sections, we discover how to learn the parameter β and how to predict y for this meal, which indicates whether the user chooses it or refuses it.

2.3.1 Data Collection

Pairwise Sampling

In pairwise sampling, participants compare two options simultaneously to determine which is preferred. The goal is to understand relative preferences between pairs of items. This method is frequently used in preference and choice studies to gather detailed preference data. Two key models used in pairwise sampling are the Thurstonian and Bradley-Terry models (Cattelan 2012). The Thurstonian model assumes each item i has a true score u_i following a normal distribution. The difference $d_{ij} = u_i - u_j$ is also normally distributed. The probability that item i is preferred over item j is given by $P(i \succ j) = \Phi\left(\frac{u_i - u_j}{\sqrt{2\sigma^2}}\right)$, where Φ is the cumulative normal distribution function. The denominator $\sqrt{2\sigma^2}$ is the standard deviation of the difference $d_{ij} = u_i - u_j$ when u_i and u_j are normally distributed with variance σ^2(Cattelan 2012). The Bradley-Terry model defines the probability of preference based on latent scores β_i and β_j. The probability that item i is preferred over item j is $P(i \succ j) = \frac{e^{\beta_i}}{e^{\beta_i} + e^{\beta_j}}$. This model is used to estimate relative strengths or preferences based on latent scores. (Cattelan 2012).

Rank-Order Sampling

Rank-order sampling methods enable analysis of human preferences by asking participants to rank a set of items from most to least preferred. This approach is widely used in voting systems, market research, and psychological studies to understand the overall preference ordering among a set of items. Rank-order sampling offers comprehensive preference data, capturing detailed information about the relative ranking of multiple items. This richness makes them suitable for various applications, including market research, voting systems, sports competitions, and recommender systems. However, these models can be more complex and time-consuming

for participants compared to pairwise comparisons, and they impose a higher cognitive load, especially with large sets of items. Additionally, participants may show inconsistencies when ranking many items (Ragain and Ugander 2019).

Rating-Scale Sampling

Rating-scale sampling is a method in which participants rate items on a numerical scale to measure the intensity of preference or attitude towards items. These models are commonly used in surveys, product reviews, and psychological assessments to gather detailed information on how participants feel about various subjects. The Likert scale is a widely used rating-scale model. In this approach, participants rate items on a fixed-point scale, typically ranging from 1 to 5 or 1 to 7, to measure levels of agreement or satisfaction. For instance, a Likert scale might ask participants to rate their agreement with statements such as "Strongly Disagree" to "Strongly Agree" (Harpe 2015). This method is prevalent in survey research, customer satisfaction studies, and attitude measurement. Another key model is the continuous rating scale, where participants mark a point on a continuous line to indicate their preference or attitude. This provides a more nuanced measure compared to discrete scales. For example, participants might indicate their satisfaction on a line ranging from "Very Unsatisfied" to "Very Satisfied" (Harpe 2015). This model is used in detailed feedback mechanisms, user experience studies, and fine-grained preference measurements.

Rating-scale sampling offers several advantages. They are simple for participants to understand and use, provide rich data on the intensity of preferences, and are flexible enough for various types of measurements (e.g., agreement, satisfaction). Moreover, the data collected can be easily analyzed using standard statistical methods (Harpe 2015).

Applications include data collection on opinions, attitudes, and behaviors; in product reviews to measure customer satisfaction and product quality; in psychological assessments to evaluate mental states, personality traits, and attitudes; and in user experience studies to understand user satisfaction and usability of products (Harpe 2015). However, rating-scale sampling methods also have limitations. Ratings can be influenced by personal biases and interpretations of scales, leading to subjectivity. There is a central tendency bias, where participants may avoid extreme ratings, resulting in a clustering of responses around the middle. Different participants might interpret scale points differently, and fixed-point scales may not capture the full nuance of participants' preferences or attitudes (Harpe 2015).

Best-Worst Scaling

Best-Worst Scaling (BWS) is a powerful method for understanding preferences and the relative importance of different items. In BWS, participants are presented with a set of items and are asked to identify the most and least preferred options. This method helps to gather detailed preference data, providing more nuanced insights than traditional ranking or rating systems. The primary objective of BWS is to discern the relative importance or preference of items within a set, making it widely applicable in various fields such as market research, health economics, and social sciences (Campbell and Erdem 2015).

A key method within BWS is MaxDiff Analysis, which involves presenting participants with sets of items and asking them to select the best and worst options. This approach yields richer data by identifying extremes in preferences, offering a clearer picture of the relative importance of each item. For instance, in a product development context, MaxDiff Analysis can help identify the most and least important features according to consumer preferences (Campbell and Erdem 2015).

The advantages of Best-Worst Scaling are significant. It provides rich data on the relative importance of items, helps clarify preferences, reduces biases found in traditional rating scales, and results in utility scores that are easy to interpret. BWS is particularly useful in market research for understanding consumer preferences, in health economics for evaluating patient treatment preferences, in social sciences for studying the importance of social issues, and in product development for identifying key features driving consumer choices (Campbell and Erdem 2015).

However, BWS also has limitations, including increased complexity and cognitive load for participants compared to simpler rating scales, potential scale interpretation differences among participants, and design challenges to avoid biases. Additionally, differences in how participants interpret the scale can introduce variability, and the design of BWS studies requires careful consideration to avoid biases, such as the order effect or the context in which items are presented.

Multiple-Choice Sampling

Multiple-choice sampling models are widely used in various fields such as voting systems, surveys, and market research to understand the preferred choice among a set of alternatives. These models involve participants selecting one option from a set of alternatives, providing insights into the most favored options.

Multiple-choice sampling methods offer several advantages. They are simple for participants to understand and reflect on realistic decision-making scenarios where individuals choose one option from many. These models are versatile and can be applied in various applications, from voting to market research, providing clear preferences directly from the participants' choices. It is particularly useful in complex choice scenarios such as mode of transportation, where choices are not independent (Bolt and Wollack 2009).

However, multiple-choice sampling also has limitations. It often relies on simplistic assumptions such as the independence of irrelevant alternatives (IIA), which may not always hold true. Additionally, these models can place a cognitive load on participants, especially if the number of choices is large, leading to decision fatigue. This method may also fail to capture the variation in preferences among different individuals, as it typically records only the most preferred choice without accounting for the relative importance of other options.

2.3.2 Data Interpretation

Binary Choice Model

is centered around one specific user option. The model predicts, for that option, after observing user choices in the past, whether that option will be chosen or not. Specifically, if we are looking at a certain choice, we use binary variable $y \in \{0, 1\}$ to represent whether that choice will be picked or not by the user in the next phase of selection. Since $\mathbb{P}(y = 0) = 1 - \mathbb{P}(y = 1)$, we only need to model $\mathbb{P}(y = 1)$ which we will denote as P.

We can use a linear model represented by the parameter β we have already defined. Since utility is a stochastic function of the choice attributes, we will represent our utility as $U = \beta^\top z + \epsilon$. We can formally model y as a function of the utility of the positive choice: $y = \mathbb{I}[U > 0]$.

We explore two cases based on the choice of distribution for the unobserved random variable ϵ. If $\epsilon \sim$ Logistic, then $\mathbb{P}(\epsilon < a) = \frac{1}{1+\exp^{-a}}$. The probability P can be modeled as:

$$P = \mathbb{P}(U > 0) = \mathbb{P}(\beta^\top z + \epsilon > 0) = \mathbb{P}(\epsilon > -\beta^\top z) = 1 - \mathbb{P}(\epsilon < -\beta^\top z) = 1 - \frac{1}{1 + \exp^{\beta^\top z}}$$
$$= \frac{1 + \exp^{\beta^\top z}}{1 + \exp^{\beta^\top z}} - \frac{1}{1 + \exp^{\beta^\top z}} = \frac{\exp^{\beta^\top z}}{1 + \exp^{\beta^\top z}} = \frac{1}{1 + \exp^{-\beta^\top z}}$$

In the health coaching example, using this logistic model, we can compute the probability that an individual would choose this meal over no meal: $P = \frac{1}{1+\exp^{-(4+15-9)}} = 0.99995$. Therefore, the model predicts a high probability that the user would choose the meal over the no-meal option.

On the other hand, if $\epsilon \sim \mathcal{N}(0, 1)$, then $\mathbb{P}(\epsilon < a) = \Phi(a)$, where $\Phi(a)$ is the cumulative distribution function of the standard normal distribution. The probability P is modeled as:

$$P = \mathbb{P}(U > 0) = \mathbb{P}(\beta^\top z + \epsilon > 0) = \mathbb{P}(\epsilon > -\beta^\top z) = \mathbb{P}(\epsilon < \beta^\top z) = \Phi(\beta^\top z)$$

In the same health coaching example, we can compute the probability that an individual would choose this meal over no meal: $\Phi(4 + 15 - 9) = 1$. This model also predicts that the user will most likely take the meal!

Bradley-Terry Model

The Bradley-Terry (BT) model introduces a framework to model the utility of choice *over all others* (a multipronged prediction of overall choices, not just a binary prediction over one choice), given their attribute vectors (Bradley and Terry 1952b). Given information about all available operations, this is a general yet powerful method for modeling human preferences. The core idea in this model is to compare utilities of all items at once to model the probability of a user's actions and, therefore, their preferences. In the BT model, we have a discrete set of J choices $i \in \{1, 2, \dots, J\}$, each with an attribute representation $z_i \in \mathbb{Z}^n$ (where n is

the dimensionality of the representation). Each choice can also have its unique random noise variable representing the unobserved factor, although we can also choose to have all choices' unobserved factors follow the same distribution (e.g. independent and identically distributed, or iid).

We keep the assumption from previous sections that the utility U_i of choice i is also a linear stochastic function where the noise is sampled from the specified distribution: $U_i = \beta^\top z_i + \epsilon_i$. The noise is represented as an extreme value distribution, although we can choose alternatives such as a multivariate Gaussian distribution: $\epsilon \sim \mathcal{N}(0, \Sigma)$. If Σ is not a diagonal matrix, we effectively model correlations in the noise across choices, enabling us to avoid the iid assumption if necessary. In the case of the extreme value distribution, we model the probability of a user preferring choice i, which we denote as P_i as $P_i = \exp(\beta^\top z_i)/Z$ where $Z = \sum_{j=1}^{J} \exp(\beta^\top z_j)$.

We revisit the health coaching example. Denote two choices, where $z_1 = [20, 15, 3]$ is the choice from the previous example. Still, we now have a second choice $z_2 = [60, 20, 7]$ (which seems to be a very carbohydrate-heavy meal and potentially a larger meal overall). We will also assume we choose an extreme value distribution to model the unobserved factors, which are sampled i.i.d. Then, we have $\beta^\top z_1 = 10$ and $\beta^\top z_2 = 11 \Rightarrow P_1 = \frac{1}{1+\exp(1)} = 0.2689$. Since there are only two choices, the probabilities P_1 and P_2 must sum to 1. Therefore, we can calculate P_2 as $P_2 = 1 - P_1 = 1 - 0.2689 \approx 0.7311$. Our model predicts that choice 2 is more favorable between these two options.

Ordered Preferences Model

In all previous examples, we have assumed that we have no information on any explicit ordering of the available options a human can choose from: all choices were treated as independent by the model. The model aims to capture how an individual chooses between them. However, in many cases, we may introduce an inductive bias based on information about the options. For example, in a study for stated preferences, a user may be able to choose from intricately dependent options such as very poor, poor, fair, good, and great. In this case, it can be useful to include this bias in our model to represent a human's decision-making process better. For such cases, instead of comparing choices against alternatives, we can focus on a single example and use additional parameters to define classification criteria based on the utility determined by the model. Formally, let us suppose we have a single example with attributes z_i, and wish to know which of J predefined options an individual will choose from. We can define $J - 1$ parameters, which act as thresholds on the utility computed by $U_i = H(z_i)$ to classify the predicted choice between these options. For example, if there are 3 predefined options, we can define parameters $a, b \in \mathbb{R}$ such that

$$y_i = \begin{cases} 1 & U < a \\ 2 & a \leq U < b \\ 3 & \text{else} \end{cases}$$

1. Logistic Distribution

From a probabilistic perspective, we can use our cumulative distributions as before to model the probability that a person will choose a given option. Continuing with our linear utility function $U_i = \beta^\top z_i + \epsilon_i$, we can start with the setting that we assume unobserved factors follow a logistic distribution and focus on the first case:

$$\mathbb{P}(y_i = 1) = \mathbb{P}(U < a) = \mathbb{P}(\beta^\top z + \epsilon < a) = \mathbb{P}(\epsilon < a - \beta^\top z) = \frac{1}{1 + \exp(\beta^\top z - a)}$$

Extending this method to the second case, where we estimate the probability of the utility falling within a specific interval:

$$\begin{aligned}\mathbb{P}(y_i = 2) = \mathbb{P}(a \leq U < b) = \mathbb{P}(a - \beta^\top z \leq \epsilon < b - \beta^\top z) &= \frac{1}{1 + \exp(\beta^\top z - b)} - (1 - \mathbb{P}(\epsilon < a - \beta^\top z)) \\ = \frac{1}{1 + \exp(\beta^\top z - b)} - (1 - \frac{1}{1 + \exp(\beta^\top z - a)}) &= \frac{1}{1 + \exp(\beta^\top z - b)} - \frac{1}{1 + \exp(a - \beta^\top z)})\end{aligned}$$

The final case follows the form of the inverse of the first case:

$$\mathbb{P}(y_i = 3) = \mathbb{P}(U > b) = \mathbb{P}(\beta^\top z + \epsilon > b) = \mathbb{P}(\epsilon > b - \beta^\top z) = 1 - \mathbb{P}(\epsilon < b - \beta^\top z) = \frac{1}{1 + \exp(\beta^\top z - b)}$$

2. Normal Distribution

In the case of modeling unobserved factors with a standard normal distribution, we have:

$$\begin{aligned}\mathbb{P}(y_i = 1) &= \mathbb{P}(U < a) = \mathbb{P}(\beta^\top z + \epsilon < a) = \mathbb{P}(\epsilon < a - \beta^\top z) = \Phi(a - \beta^\top z) \\ \mathbb{P}(y_i = 2) &= \mathbb{P}(a \leq U < b) = \mathbb{P}(a - \beta^\top z \leq \epsilon < b - \beta^\top z) = \Phi(b - \beta^\top z) - \Phi(a - \beta^\top z) \\ \mathbb{P}(y_i = 3) &= \mathbb{P}(U > b) = 1 - \Phi(b - \beta^\top z)\end{aligned}$$

In our health coaching example, the derivation above yields three exact expressions for computing the probability of choosing each of our meals. Each computation involves the normal cumulative distribution function as seen for the binary choice model with standard normal for ϵ after parameters a and b are learned Section 2.4.

Plackett-Luce Model

In other cases, we may need an even more general framework combining elements of the BT model and ordered preferences. Specifically, we can model an open-ended ranking of the available options in a similar probabilistic framework. To do so, we can leverage the Plackett-Luce (PL) Model, in which we jointly model the full sequence of choice ordering. (Plackett 1975)

The general form models the joint distribution as the product of conditional probabilities, where each is conditioned on the preceding ranking terms. Given an ordering of J choices $\{Y_1, Y_2, \dots, Y_J\}$ where Y_1 is the first selection, Y_2 is the second, and so on, we decompose the joint probability into its respective conditionals. To compute the conditional probabilities, we can use the same method as the BT model, using a softmax to produce valid conditional distributions for each element of the sequence:

$$\mathbb{P}(Y_1, Y_2, \dots, Y_J) = \mathbb{P}(Y_1) \cdot \mathbb{P}(Y_2|Y_1) \cdot \dots \cdot \mathbb{P}(Y_J|Y_1, Y_2, \dots Y_{J-1}) = \prod_{i=1}^{J} \frac{\exp(\beta^\top z_i)}{\sum_{j \geq i} \exp(\beta^\top z_j)}$$

An interesting property of the PL Model is that in the naive case of only ordering a single choice, it is equivalent to the pairwise preference formulation of the BT model.

Exercise (Health coaching example): In our application, if we have 3 choices (burrito (B), fries (F), rice bowl (R)), we can let Y_1, Y_2, Y_3 be variables to which we assign meals in a one-to-one manner to establish a ranking.

1. One of the possible ranking assignments is $Y_1 = B, Y_2 = F, Y_3 = R$. How many assignments are there in all, and what are they explicitly?
2. What would one expect the sign to be, out of $\{\leq, \geq, =\}$ in the following expression? (Hint: healthier meals should be placed earlier in the ranking.)

$$\mathbb{P}(Y_1 = F, Y_2 = R, Y_3 = B) \ \_\_ \ \mathbb{P}(Y_1 = R, Y_2 = B, Y_3 = F)$$

Ideal Point Model

An observation one can make is that we have strictly used linear functions to represent the utility. However, in the case of vector representations of choice attributes and the individual, one can exploit vector geometry to compute this utility value. The Ideal Point Model does this by using distance functions to compute utility for individual-choice pairs (Huber 1976). Formally, with our vector representation z_i of choice i and a vector $\mathbf{v}_n$ representing an individual n, we can use a distance function to model a stochastic utility function, keeping the notion of unobserved factors following a specified distribution: $U_{n,i} = \texttt{dist}(z_i, \mathbf{v}_n) + \epsilon_{n,i}$. We continue with our framework of a human's preference following the choice corresponding to the maximum utility: $y_{n,i} = \mathbb{I}[U_{n,i} > U_{n,j} \ \forall i \neq j]$. The intuition supporting this type of model is that vectors exist in a shared n-dimensional space, and as such we can use geometry to match choices whose representations are closest to that of a given individual.

An observation with this model type is that it can often result in faster learning compared to non-geometric approaches (Jamieson and Nowak 2011; Tatli, Nowak, and Vinayak 2022). However, it carries the added burden of having to specify a distance metric. Certain distance metrics, such as Euclidian distance or inner product, can easily be biased by the scale of vectors. A distance measure such as cosine similarity, which compensates for scale by normalizing the inner product of two vectors by the product of their magnitudes, can mitigate this bias yet may

discard valuable information encoded by the length of the vectors. Beyond the distance metric alone, this model places a strong inductive bias that the individual and choice representations all share a common embedding space. In some contexts, this can be a robust bias to add to the model (Greiner 2005), but it is a key factor one must take into account before employing such a model, and is a key design choice for modeling.

Health coaching example: vector representations may indeed be useful as an individual's representation can capture the macronutrient proportions and volumes they wish to consume, enabling a distance metric such as inner product to be a powerful tool. This model also starts capturing user properties (e.g. a user may be more into working out, another into lowering anxiety and another into gaining weight) and implicitly the commonalities between user characteristics start being captured, akin to a recommendation system (Roy and Dutta 2022). However, in other domains and formulations, where perhaps user profiles are not as explicit, this may certainly hinder performance and make learning human preferences difficult.

2.4 Parameter Learning

With an understanding of the various techniques we can use to model human preferences, we can now create robust models which utilize context attributes about the options an individual has in front of them and model their choices. However, these models on their own are powerless; their parameters are initialized randomly and we must fit the models to the actual human choice data!

Each of the models we have studied contain distinct parameters which aim to capture human preferences; for example β is a parameter vector containing variables which represent a linear function to compute utility given a choice's attributes. We can also choose to represent stochastic utility functions or embedding functions for Ideal Point Models as neural networks. But how can we compute the optimal values of these parameters?

In this section, we give the reader an overview of the different methods available to tune human preference model parameters using given data. We refer the reader to (Casella and Berger 1990; Bock et al. 2015) for first-principle derivations of these methods and a deeper dive into their theoretical properties (convergence, generalization, data-hungriness, etc.).

A common and powerful approach for computing the parameters of a model is maximum likelihood estimation (Casella and Berger 1990; Bock et al. 2015). The likelihood of a model is the probability of the observed data given the model parameters; intuitively we wish to maximize this likelihood, as that would mean that our model associates observed human preferences in the data with high probability. We can formally define the likelihood for a model with parameters β and a given data point (z_i, y_i) as:

$$\mathcal{L}(z_i, y_i; \beta) = \mathbb{P}(y = y_i | z_i; \beta)$$

Assuming our data is independent and identically distributed (iid), the likelihood over the entire dataset is the joint probability of all observed data as defined by the model:

$$\mathcal{L}(z, Y; \beta) = \prod_{i=1}^{J} \mathbb{P}(y = y_i | z_i; \beta)$$

In our very first example of binary choice with logistic noise, this was simply the model's probability of the observed preference value:

$$\mathcal{L}(z_i, y_i; \beta) = \frac{1}{1 + \exp^{-\beta^\top z}}$$

In the same case with noise following a standard normal distribution, this took the form:

$$\mathcal{L}(z_i, y_i; \beta) = \Phi(\beta^\top z)$$

Fortunately, in these cases, there are straightforward methods for parameter estimation: logistic regression and probit regression (binary or multinomial, depending on the model), respectively. We can use ordinal regression to estimate the model's parameters for our ordered preference model.

Generally, the objective function commonly found in parameter learning can be optimized with stochastic gradient descent (SGD) (Ruder 2016). We can define an objective function as the likelihood to maximize this objective. Since SGD minimizes a given objective, we must negate the likelihood, which ensures that a converged solution maximizes the likelihood. SGD operates by computing the gradient of the objective with respect to the parameters of the model, which provides a signal of the direction in which the parameters must move to *maximize* the objective. Then, SGD makes an update step by subtracting this gradient from the parameters (most often with a scale factor called a *learning rate*), to move the parameters in a direction which *minimizes* the objective. When the objective is the negative likelihood (or sometimes negative log-likelihood for convenience or tractability), the result is an increase in the overall likelihood.

In the case of logistic and Gaussian models, SGD may yield a challenging optimization problem as its stochasticity can lead to noisy updates, for example, if certain examples or batches of examples are biased. Mitigations include batched SGD, in which multiple samples are randomly sampled from the dataset at each iteration, learning rates, which reduce the impact of noisy gradient updates, and momentum and higher-order optimizers which reduce noise by using movering averages of gradients or provide better estimates of the best direction in which to update the gradients. Some models, such as those that use neural networks, may, in fact, be intractable to estimate without a method such as SGD (or its momentum-based derivatives). For example, neural networks with many layers, non-linearities, and parameters can only be efficiently computed with gradient-based methods.

2.4.1 Reward Learning with Large Language Models

Taking a step away from explicitly modeling human bias and preference, we consider applying a deep learning approach to state-of-the-art language models. We begin by introducing the concepts of *foundation models* and *alignment*. A foundation model (Bommasani et al. 2021) in machine learning typically refers to a large and pre-trained neural network model that serves as the basis for various downstream tasks. In natural language processing, models like GPT-3, Llama, and BERT are considered foundation models. They are pre-trained on a massive corpus of text data, learning to understand language and context, and are capable of various language-related tasks such as text classification, language generation, and question answering. Foundation models are important because they alleviate the need to train massive neural networks from scratch, a compute and data expensive endeavor. However, a raw foundation model, trained on a pretraining objective such as a language modeling objective, is not useful on its own. It must be aligned to respond correctly based on human preferences.

In short, alignment for foundation models is the process by which model behavior is aligned with human values, ethics, and societal norms. Large Language Models (LLMs) are a foundation model for natural language processing. They are trained using a next-word prediction objective, allowing them to generate coherent language. A simple way to align a Large Language Model is to train it to follow instructions in a supervised way, using instruction-response pairs curated by hand. However, this limits the upper limit of LLM performance to the performance of the annotators' writing abilities. This type of annotation is also expensive.

An alternative, more promising approach is to train LLMs using reinforcement learning, potentially enabling them to surpass human-level performance. The main challenge with this method lies in defining an explicit reward function for generating free-form text. To address this, a reward model (RM) can be trained based on human preferences, providing a mechanism to score the quality of the generated text. This approach, known as Reinforcement Learning from Human Feedback (RLHF), leverages human feedback to guide model training, allowing LLMs to better align with human expectations while continuously improving performance.

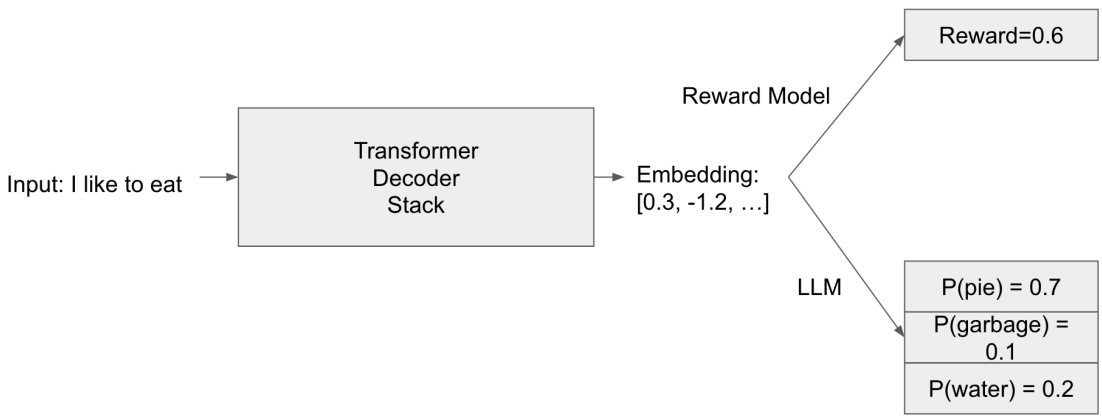


Figure 2.1
Overall architecture of a reward model based on LLM

The Llama2 reward model (Touvron et al. 2023) is initialized from the pretrained Llama2 LLM. In the LLM, the last layer is a mapping $L : \mathbb{R}^D \to \mathbb{R}^V$, where D is the embedding dimension from the transformer decoder stack and V is the vocabulary size. To get the RM, we replace that last layer with a randomly initialized scalar head that maps $L : \mathbb{R}^D \to \mathbb{R}^1$. It's important to initialize the RM from the LLM it's meant to evaluate. This is because:

1. The RM will have the same "knowledge" as the LLM. This is particularly useful if evaluating things like "does the LLM know when it doesn't know?". However, in cases where the RM is simply evaluating helpfulness or factuality, it may be useful to have the RM know more.
2. The RM is on distribution for the LLM - it is initialized in a way where it semantically understands the LLM's outputs.

An RM is trained with paired preferences, following the format:

$$\langle prompt\_history, response\_accepted, response\_rejected \rangle$$

Prompt_history is a multiturn history of user prompts and model generations, response_accepted is the preferred final model generation by an annotator, and response_rejected is the unpreferred response. The RM is trained with a binary ranking loss with an optional margin term m(r), shown in equation (7). There is also often a small regularization term added to center the score distribution on 0.

$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r) - m(r)))$$

The margin term increases the distance in scores specifically for preference pairs annotators rate as easier to separate.

Table 2.2
~~Two variants of preference rating based margin with different magnitude.~~

	Significantly Better	Better	Slightly Better	Negligibly Better / Unsure
Margin Small	1	2/3	1/3	0
Margin Large	3	2	1	0

It may seem confusing how the margins were chosen. It's primarily because the sigmoid function, which is used to normalize the raw reward model score, flattens out beyond the range of $[-4, 4]$. Thus, the maximum possible margin is eight.

When training or using a reward model, watching for the following is important:

1. **LLM Distribution Shift**: With each finetune of the LLM, the RM should be updated through a collection of fresh human preferences using generations from the new LLM. This ensures that the RM stays aligned with the current distribution of the LLM and avoids drifting off-distribution.

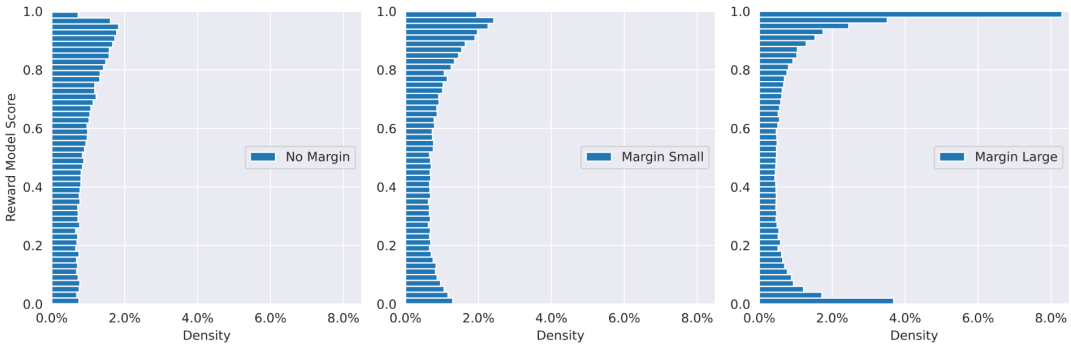


Figure 2.2
Reward model score distribution shift caused by incorporating preference rating based margin in ranking loss. With the margin term, we observe a binary split pattern in reward distribution, especially for a larger margin.

2. **RM and LLM are coupled**: An RM is generally optimized to distinguish human preferences more efficiently within the specific distribution of the LLM to be optimized. However, this specialization poses a challenge: such an RM will underperform when dealing with generations not aligned with this specific LLM distribution, such as generations from a completely different LLM.
3. **Training Sensitivities of RMs**: Training RMs can be unstable and prone to overfitting, especially with multiple training epochs. It's generally advisable to limit the number of epochs during RM training to avoid this issue.

The industry has centered around optimizing for two primary qualities in LLMs: helpfulness and harmlessness (safety). There are also other axes such as factuality, reasoning, tool use, code, multilingual, and more, but these are out of scope for us. In the Llama2 paper, preference data was collected from humans for each quality, with separate guidelines. This presents a challenge for co-optimizing the final LLM towards both goals.

Two main approaches can be taken for Reinforcement Learning from Human Feedback (RLHF) in this context:

1. Train a unified reward model that integrates both datasets.
2. Train two separate reward models, one for each quality, and optimize the LLM toward both.

Option 1 is difficult because of the tension between helpfulness and harmlessness. They trade off against each other, confusing an RM trained on both. The chosen solution was option 2, where two RMs are used to train the LLM in a piecewise fashion. The helpfulness RM is used as the primary optimization term, while the harmlessness RM acts as a penalty term, driving the behavior of the LLM away from unsafe territory only when the LLM veers beyond a certain threshold. This is formalized as follows, where R_s, R_h, and R_c are the safety, helpfulness, and

combined reward, respectively. g and p are the model generation and the user prompt:

$$R_c(g \mid p) = \begin{cases} R_s(g \mid p) & \text{if is\_safety}(p) \text{ or } R_s(g \mid p) < 0.15 \\ R_h(g \mid p) & \text{otherwise} \end{cases}$$

There are several open issues with reward models alluded to in the paper. For example, how best to collect human feedback? Training annotators and making sure they do the correct thing is hard. What should the guidelines be? Another question is whether RMs can be made robust to adversarial prompts. Last but not least, do RMs have well-calibrated scores? This matters for RLHF - pure preference accuracy isn't enough.

2.4.2 Reward Learning in Robotics

To help set up our basic reward learning problem, consider a user and a robot. The user's preferences or goals can be represented by an internal reward function, R(ξ), which the robot needs to learn. Since the reward function isn't explicit, there are a variety of ways that the robot can learn this reward function, which we will discuss in the next section. An example method of learning a reward function from human data is using pairwise comparison. Consider the robot example from section one, but now, the robot shows the human two possible trajectories ξ_A and ξ_B as depicted in the diagram below.

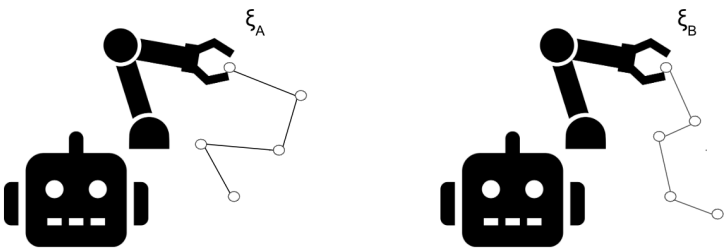


Figure 2.3
Two different trajectories taken by a robot to prompt user ranking.

The user is show both the trajectories above and asked to rank which one is better. Based on iterations of multiple trajectories and ranking, the robot is able to learn the user's internal reward function. There quite a lot of ways that models can learn a reward function from human data. Here's a list (Myers et al. 2021) of some of them:

1. Pairwise comparison: This is the method that we saw illustrated in the previous example. The robot is able to learn based on a comparison ranking provided by the user.
2. Expert demonstrations: Experts perform the task and the robot learns the optimal reward function from these demonstrations.
3. Sub-optimal demonstrations: The robot is provided with demonstrations that are not quite as good as the expert demonstrations but it is still able to learn a noisy reward function from the demonstrations.

4. Physical Corrections: While the robot is performing the task, at each point in its trajectory (or at an arbitrary point in its trajectory) its arm is corrected to a more suitable position. Based on these corrections, the robot is able to learn the reward function.
5. Ranking: This method is similar to pairwise comparison but involves more trajectories than 2. All the trajectories may have subtle differences from each other, but these differences help provide insight to the model.
6. Trajectory Assessment: Given a single trajectory, the user rates how close it is to optimal, typically using a ranking scale.

   Each of these methods allows the robot to refine its understanding of the user's reward function, but their effectiveness can vary depending on the application. For instance, expert demonstrations tend to produce more reliable results but may not always be feasible in everyday tasks. Pairwise comparison and ranking methods offer more flexibility but might require a higher number of iterations.

2.4.3 Reward Learning with Meta Learning

Learning a reward function from human preferences is an intricate and complicated task. At its core, this task is about designing algorithms that can capture what humans value based on their elicited preferences. However, due to the nuanced and multifaceted nature of human desires, learning reward functions from human can be a difficult task. Therefore, meta-learning rewards may be considered to facilitate the reward learning processes. Meta-learning, often referred to as "learning to learn," aims to design models that can adapt to new tasks with minimal additional efforts. We discuss paper (Hejna III and Sadigh 2023) in Section 2.4.3.1 showing how meta-learning can be leveraged for few-shot preference learning, where a system can quickly adapt to a new task after only a few queries to pairwise preferences from human.

Moving beyond the concept of learning from pairwise preferences, in Section 2.4.3.2 we discuss a different approach where meta-learning intersects with both demonstrations and rewards (Zhou et al. 2019). This paper considers the use of both demonstrations and rewards elicited from human that guide the learning process.

In the regular learning setting, a model is fitted to a dataset with certain learning algorithm. The learning algorithm, for example, can be the minimization of a loss function. To formulate the "regular" learning procedure, let's denote the training dataset as D, and the test dataset as S. Given a model parameterized by θ; training loss function $L(\theta, D)$; and test loss function $L(\theta, S)$, we can formulate a process of "regular" machine learning process as

$$\theta^\star = \arg\min_\theta \quad L(\theta, D).$$

Note that the minimization of the training loss function is essentially *one* possible learning algorithm. For example, instead of minimizing the loss function, one may do gradient descent with model regularization on the loss function, where the final solution may not be the one

that actually minimizes the loss function. As a result, we may want to be more general and more abstract for the moment, and denote the learning algorithm as $\mathcal{A}$. Thus, we can write

$$\theta^\star = \mathcal{A}(D),$$

i.e., the learning algorithm $\mathcal{A}$ takes in a training dataset and outputs a model parameter $\theta^\star$. Then, the performance of the model is evaluated by the test loss $L(\mathcal{A}(D), S)$. As we can see, in the regime of "regular" learning, the learning algorithm $\mathcal{A}$ is pre-defined and fixed.

Meta-learning, or learning-to-learn, essentially asks the question of whether one can *learn* the learning algorithm $\mathcal{A}$ from prior tasks, such that the modal can adapt to a new task more quickly/proficiently. For example, different human languages share similar ideas, and therefore a human expert who has learned many languages should be able to learn a new language easier than an average person. In other words, the human expert should have learned how to learn new languages more quickly based on their past experiences on learning languages.

To mathematically formulate meta-learning, we consider a family of learning algorithms $\mathcal{A}_\omega$ parameterized by ω. The "prior" tasks are represented by a set of meta-training datasets $\{(D_i, S_i)\}_{i=1}^N$ consists of N pairs of training dataset D_i and test dataset S_i. As we noted before, a learning algorithm $\mathcal{A}_\omega$ takes in a training dataset, and outputs a model, i.e.,

$$\forall i: \quad \theta_i^\star = \mathcal{A}_\omega(D_i).$$

Therefore, the **meta-learning objective** is

$$\min_\omega \quad \sum_i L(\mathcal{A}_\omega(D_i), S_i).$$

The above optimization problem gives a solution $\omega^\star$ which we use as the meta-parameter. Then, when a new task comes with a new training dataset D_{new}, we can simply apply $\theta_{new}^\star = \mathcal{A}_{\omega^\star}(D_{new})$ to obtain the adapted model $\theta_{new}^\star$. Note that we usually assume the meta-training datasets D_i, S_i and the new dataset D_{new} share the same underlying structure, or they come from the same distribution of datasets.

One of the most popular meta-learning method is Model-Agnosic Meta-Learning (MAML) (Finn, Abbeel, and Levine 2017). In MAML, the meta-parameter ω shares the same space as the model parameter θ. At its core, in MAML the learning algorithm is defined to be

$$\mathcal{A}_\omega(D_i) = \omega - \alpha \nabla_\omega L(\omega, D_i),$$

where α is the step size. As we can see, in fact ω is defined as the initialization of fine-tuning θ. With a good ω learned, the model can adapt to a new task very quickly. In general, meta-learning can be summarized as follows: Given data from prior tasks, learn to solve a new task more quickly/proficiently. Given the general nature of meta-learning, one may be curious about whether preference learning can be benefited from meta-learning, which we discuss in the following section.

2.4.3.1 *Few-Shot Preference Learning for Reinforcement Learning*

Reinforcement learning (RL) in robotics often stumbles when it comes to devising reward functions aligning with human intentions. Preference-based RL algorithms aim to solve this by learning from human feedback, but this often demands a *highly impractical number of queries* or leads to oversimplified reward functions that don't hold up in real-world tasks.

To address the impractical requirement of human queries, as we discussed in the previous section, one may apply meta-learning so that the RL agent can adapt to new tasks with fewer human queries. (Hejna III and Sadigh 2023) proposes to pre-training models on previous tasks with the meta-learning method MAML (Finn, Abbeel, and Levine 2017), and then the meta-trained model can adapt to new tasks with fewer queries.

We consider Reinforcement Learning (RL) settings where a state is denoted as $s \in S$, and action is denoted as $a \in A$, for state space S and action space A. The reward function $r : S \times A \to \mathbb{R}$ is unknown and need to be learned from eliciting human preferences. There are multiple tasks, where each task has its own reward function and transition probabilities. The reward model is parameterized by ψ. We denote $\hat{r}_\psi(s, a)$ to be a learned estimate of an unknown ground-truth reward function $r(s, a)$, parameterized by ψ. Accordingly, a reward model determines a RL policy ϕ by maximizing the accumulated rewards. The preferences is learned via pairwise comparison of trajectory segments

$$\sigma = (s_t, a_t, s_{t+1}, a_{t+1}, ..., s_{t+k-1}, s_{t+k-1})$$

of k states and actions.

For each pre-training task, there is a dataset D consists of labeled queries (σ_1, σ_2, y) where $y \in \{0, 1\}$ is the label representing which trajectory is preferred. Therefore, a loss function $L(\psi, D)$ captures how well the reward model characterizes the preferences in dataset D. In (Hejna III and Sadigh 2023) they the preference predictor over segments using the Bradley-Terry model of paired comparisons (Bradley and Terry 1952a), i.e.,

$$P[\sigma_1 \succ \sigma_2] = \frac{\exp \sum_t \hat{r}_\psi(s_t^1, a_t^1)}{\exp \sum_t \hat{r}_\psi(s_t^1, a_t^1) + \exp \sum_t \hat{r}_\psi(s_t^2, a_t^2)}.$$

Then, the loss function is essentially a binary cross-entropy which the reward model ψ aims to minimize, i.e.,

$$L(\psi, D) = -\mathbb{E}_{(\sigma^1, \sigma^2, y) \sim D} \left[y(1) \log(P[\sigma_1 \succ \sigma_2]) + y(2) \log(1 - P[\sigma_1 \succ \sigma_2]) \right].$$

Method Component 1: Pre-Training with Meta Learning

To efficiently approximate the reward function r_{new} for a new task with minimal queries, as described in (Hejna III and Sadigh 2023), we aim to utilize a pre-trained reward function $\hat{r}_\psi$ that can be quickly fine-tuned using just a few preference comparisons. By pre-training on data from prior tasks, we can leverage the common structure across tasks to speed up the adaptation

process. Although any meta-learning method is compatible, (Hejna III and Sadigh 2023) opt for Model Agnostic Meta-Learning (MAML) due to its simplicity. Therefore, the pre-training update for the reward model ψ is

$$\psi \leftarrow \psi - \beta \nabla_{\psi} \sum_{i=1}^{N} L(\psi - \alpha \nabla_{\psi} L(\psi, D_i), D_i),$$

where α, β are the inner and outer learning rate, respectively. We note that data $\{D_i\}_i$ of labeled preferences queries for prior tasks can come from offline datasets, simulated policies, or actual humans.

Method Component 2: Few-Shot Adaptation

With the aforementioned pre-training with meta learning, the meta-learned reward model can then be used for few-shot preference based RL during an online adaptation phase. The core procedure of the few-shot adaption is descibed as below

1. Given a pre-trained reward model ψ
2. For time step $t = 1, 2, ...$
    1. Find pairs of trajectories (σ_1, σ_2) with preference uncertainty based on ψ.
    2. Query human preference y and forms a new dataset D_{new}
    3. Update the reward model by $\psi' \leftarrow \psi - \alpha \nabla_{\psi} L(\psi, D_{new})$
    4. Update the policy with the new reward model ψ'

As mentioned in (Hejna III and Sadigh 2023), uncertain queries are selected using the disagreement of an ensemble of reward functions over the preference predictors. Specifically, comparisons that maximize $\texttt{std}(P[\sigma_1 \succ \sigma_2])$ are selected each time feedback is collected.

The whole pipeline of the method is outlined in Figure 2.4.

We present one set of experiment from the paper, as it illustrates the effectiveness of the proposed method in a straightforward way. The experiment test the propoesed method on the Meta-World benchmark (Yu et al. 2020). Three baselines are compared with the proposed method:

1. SAC: The Soft-Actor Critic RL algorithm trained from ground truth rewards. This represents the standard best possible method given the ground-truth reward.
2. PEBBLE: The PEBBLE algorithm (Lee, Smith, and Abbeel 2021). It does not use information from pripor tasks.
3. Init: This method initialize the reward model with the pretained weights from meta learning. However, instead of adapting the reward model to the new task, it performs standard updates as in PEBBLE.

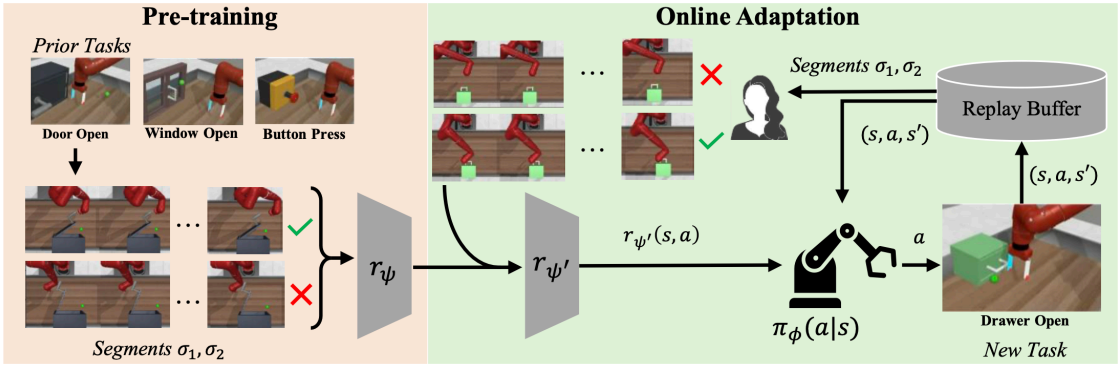


Figure 2.4
An overview of the proposed method in (Hejna III and Sadigh 2023). **Pre-training (left)**: In the pre-training phase, trajectory segment comparisons are generated using data from previously learned tasks. Then, they are used to train a reward model. **Online-Adaptation (Right)**: After pre-training the reward model, it is adapted to new data from human feedback. The adapted reward model is then used to train a policy for a new task in a closed loop manner.

The results are shown in Figure 2.5, where we can see that the proposed methord outperforms all of the baselines.

This paper (Hejna III and Sadigh 2023) shows that meta reward learning indeed reduce the number of queries of human preferences. However, as mentioned in the paper, there are still some drawbacks, as shown in the following.

Many of the queries the model pick for human preference elicitation are actually almost identical to human. After all, the model would pick the most uncertain pair of trajectories for human preference queries, and similar trajectories are for sure having high uncertainty in their preference. This suggest the need of new ways for designing the query selection strategy.

Moreover, despite the improved query complexity, it still needs an impractical amount of queries. As shown in Figure 2.5, the "sweep into" task still needs 2500 human queries for it to work properly, which is still not ideal for what we want them to be.

In addition, it is mentioned in the paper that the proposed method may be even worse than training from scratch, if the new task is too out-of-distribution. Certainly, since meta-learning assumes in-distribution tasks, we cannot expect the proposed method to be good for out-of-distribution task. It is thus an interesting future direction to investigate whether one can design a method that automatically balance between using the prior information or training from scratch.

2.4.3.2 Watch Try Learn

Watch, Try, Learn: Meta-Learning from Demonstrations and Rewards (Zhou et al. 2019) asks the question "How can we efficiently learn both from expert demonstrations and from trials where we only get **binary** feedback from a human". Why do we care about this question? In the context of robotics, a very compelling answer is the *cost of data-collection*. In a hypothetical world in which we have a vast number of **expert demonstrations** of robots accomplishing a

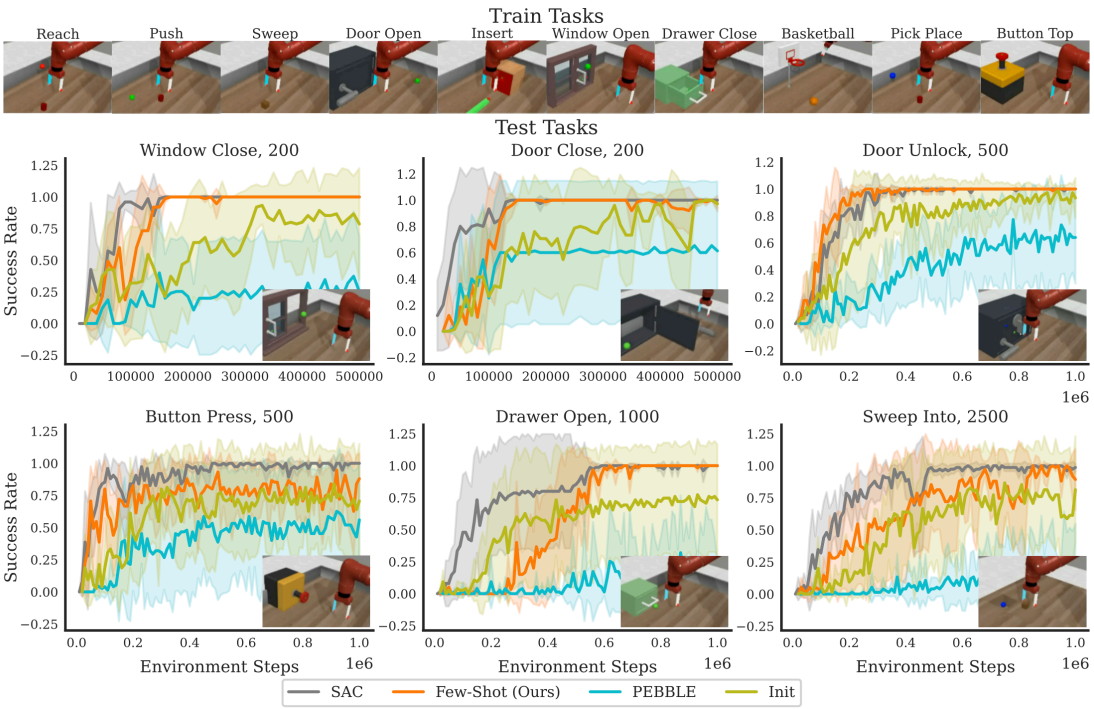


Figure 2.5
Results on MetaWorld tasks. The title of each subplot indicates the task and number of artificial feedback queries used in training. Results for each method are shown across five seeds.

large number of diverse tasks, we don't necessarily need to worry about learning from trials or from humans. We could simply learn a very capable imitation agent to perform any task. Natural Language Processing could be seen as living in this world, because internet-scale data is available. **Robots, however, are expensive**, so people generally don't have access to them, and therefore cannot use them to produce information to imitate. Similarly, **human time is expensive**, so even for large organizations that do have access to a lot of robots, it's still hard to collect a lot of expert demonstrations.

The largest available collection of robotics datasets today is Open X-Embodiment ((Padalkar et al. 2023)), which consists of around 1M episodes from more than 300 different scenes. Even such large datastes are not enough to learn generally-capable robotic policies from imitation learning alone.

Main insight: binary feedback is much cheaper to obtain than expert demonstrations! Instead of hiring people to act as robot operators to tell the robot exactly what to do, if there was a way of having many robots trying things in parallel, we can have humans watch videos of what the robots did and then give a success classification of whether the robot accomplished the goal. This is a much cheaper form of human supervision because the human labels don't necessarily need to be given in real time, so one human labeler can label many trajectories in parallel, and the human doesn't need to be a skilled robot operator.

Concretely, this paper seeks to learn new tasks with the following general problem setting:

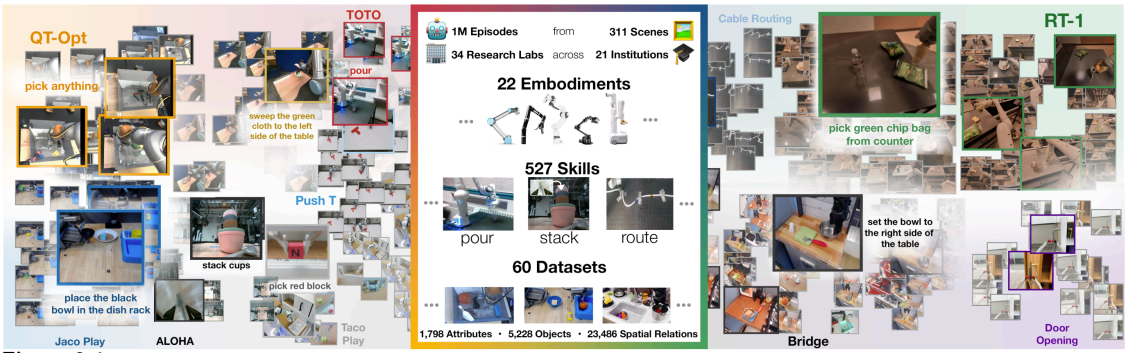


Figure 2.6
Visualization of the Open X-Embodiment dataset collection. Even this large-scale dataset for robot learning is not yet enough to learn generally-capable robotic policies.

1. We only get 1 expert demonstration of the target task
2. After seeing the expert demonstration, we have robots try to solve the task 1 or more times.
3. The user (or some pre-defined reward function) annotates each trial as success/failure.
4. The agent learns from both the demos and the annotated trials to perform well on the target task.

Note that this work falls under the **meta-learning** umbrella, because we are learning an algorithm for quickly learning new tasks given new observations (demos, trials, and success labels.)

The **main contribution** of this paper is a meta-learning algorithm for incorporating demonstrations and binary feedback from trials to solve new tasks.

Meta-Learning deals with efficient learning of new tasks. In the context of robotics or reinforcement learning in general, **how do we define tasks?** We will use the Markov decision process (**MDP**) formalism. A task T_i is described with the tuple $\{S, A, r_i, P_i\}$.

1. S represents the *state-space* of the task, or all possible states the agent could find itself in. This work uses image-observations, so S is the space of all possible RGB images.
2. A is the action space, meaning the set of all possible actions the agent could take. In robotics there are many ways of representing action spaces, and this work considers end-effector positions, rotations, and opening.
3. r_i is the reward function for the task, with function signature $r_i : S \times A \to \mathbb{R}$. This work assumes all reward functions are binary.
4. P_i is the transition dynamics function. It's a function that maps state-action pairs to probability distributions over next states.

Notice that S and A are shared across tasks. Transition dynamics functions are normally also shared between tasks because they represent the laws of physics. However, this work considers environments with different objects, so they don't share the dynamics function. Given this definition for tasks, they assume that the tasks from the data that they get come from some unknown task-generating distribution $p(T)$.

Let's give a more precise definition of the problem statement considered by **Watch, Try, Learn**. As the paper name suggests, there are 3 phases for the problem statement.

Watch: During the *watch* phase, we give the agent K demonstrations of the target tasks. This paper considers the case where K always equals 1, and all demonstrations are successful. That is, each demonstration consists of a trajectory $\{(s_0, a_0), ..., (s_H, a_H)\}$ where H is the task horizon, and the final state is always successful, that is $r_i(s_H, a_H) = 1, r_i(s_j, a_j) = 0$ for every $j \neq H$.

Importantly, these demonstrations alone might not be sufficient for **full task specification**. As an example, consider a demonstration in which an apple is moved to the right, next to a pan. Seeing this demonstration alone, the task could be always moving the apple to the right, or it could be always moving the apple next to the pan, irrespective of where the pan is. The expected output after the Watch phase is a policy capable of gathering information about a task, given demonstrations.

Try: In the Try phase, we use the agent learned during the Watch phase to attempt the task for L trials. As specified earlier, this paper considers the casae where L always equals 1. After the agent completes the trials, humans (or pre-programmed reward functions) provide one binary reward for each trial, indicating whether the trial was successful. The expected output of this phase is L trajectories and corresponding feedback that hopefully *disambiguate* the task.

Learn: After completing the trials, the agent must learn from both the original expert demonstrations and the trials, and become capable of solving the target task.

Given Data: To train agents that can Watch, Try, and Learn, we are given a dataset of expert demonstrations containing multiple demos for each task, and the dataset contains hundreds of tasks. Importantly, **no online interaction** is needed for training, and this method trains only with **supervised learning** and no reinforcement learning.

This section describes exactly how this paper trains an agent from the given expert demonstrations, and how to incorporate the trials and human feedback into the loop.

Training to Watch: We now describe the algorithm to obtain an agent conditioned on the given expert demonstration. In particular, what we want to obtain out of the Watch phase is a policy conditioned on a set of expert demonstrations. Formally, we want to obtain $\pi_\theta^{\text{watch}}(a|s, \{d_{i,k}\})$.

The way we can obtain this policy is through **meta-imitation learning**. Given the demonstrations $\{\mathbf{d}_{i,k}\}$ for task i, we sample another *different* demonstration coming from the same task $\mathbf{d}_i^{\text{test}}$. The key insight here is that $\mathbf{d}_i^{\text{test}}$ is an example of **optimal behavior** given the demonstrations. Therefore, to obtain $\pi_\theta^{\text{watch}}(a|s, \{d_{i,k}\})$, we simply regress the policy to imitate actions taken on $\mathbf{d}_i^{\text{test}}$. Concretely, we train policy parameters θ to minimize the following loss:

$$\mathcal{L}^{\text{watch}}(\theta, \mathcal{D}_i^*) = \mathbb{E}_{\{d_{i,k}\}\sim\mathcal{D}_i^*}\mathbb{E}_{\{d_{i,k}^{\text{test}}\}\sim\mathcal{D}_i^*\{d_{i,k}\}}\mathbb{E}_{(s_t,a_t)\sim d_i^{\text{test}}}\big[-\log\pi_\theta^{\text{watch}}(a_t|s_t,\{d_{i,k}\})\big]$$

This corresponds to doing imitation learning by minimizing the negative log-likelihood of the test trajectory actions, conditioning the policy on the entire demo set. However, how is the conditioning on the demo set achieved?

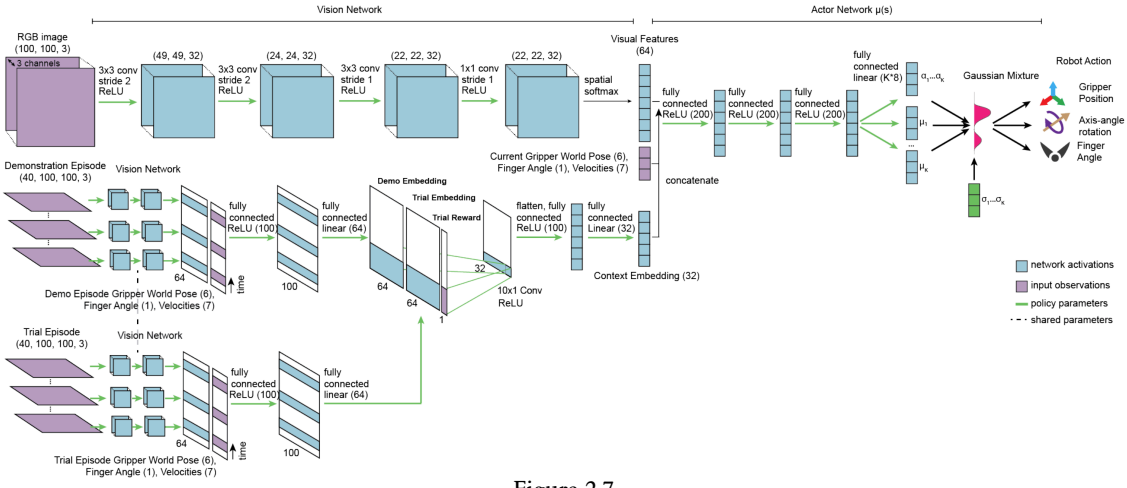


Figure 2.7
Vision-based policy architecture that conditions on a set of demonstrations.

Figure 2.7 visualizes how Watch Try Learn deals with conditioning on demonstrations. In addition to using features obtained from the images of the current state, the architecture uses features from frames sampled (in order) from the demonstration episodes, which are concatenated together.

Trying: On the **Try** phase, when the agent is given a set of demonstrations $\{\mathbf{d}_{i,k}\}$, we deploy the policy $\pi_\theta^{\text{watch}}(a|s,\{\mathbf{d}_{i,k}\})$ to collect L trials. There is no training involved in the Try phase, we simply condition the policy on the given demonstrations

Training to Learn: During the Watch phase the objective was to train a policy conditioned on demonstrations $\pi_\theta^{\text{watch}}(a|s,\{\mathbf{d}_{i,k}\})$. The authors of Watch, Try, Learn use a similar strategy as the Watch phase for the Learn phase. We now want to train a policy that is conditioned on the demonstrations, as well as the trials and binary feedback. That is, we want to learn $\pi_\phi^{\text{watch}}(a|s,\{\mathbf{d}_{i,k}\},\{\quad_{i,l}\})$. To train the policy, we again use meta-imitation learning where we additionally sample yet another trajectory from the same task. Concretely, we train policy parameters ϕ to minimize the following loss:

$$\mathcal{L}^{\text{learn}}(\phi, \mathcal{D}_i, \mathcal{D}_i^*) = \mathbb{E}_{(\{d_{i,k}\},\{\quad_{i,l}\})\sim\mathcal{D}_i}\mathbb{E}_{\{d_{i,k}^{\text{test}}\}\sim\mathcal{D}_i^*\{d_{i,k}\}}\mathbb{E}_{(s_t,a_t)\sim d_i^{\text{test}}}\big[-\log\pi_\theta^{\text{learn}}(a_t|s_t,\{d_{i,k}\},\{\tau_{i,l}\})\big]$$

The conditioning on both the demo episodes and the trial episodes is achieved in the exact same way as in the Watch phase, and is visualized in Figure 2.7. The architecture is simply adjusted to be able to take in more images fro mthe trial episodes.

In this section, we describe the evaluation suite for the paper, including the simulation benchmark used, the baselines considered, and the results.

Gripper environment setup:

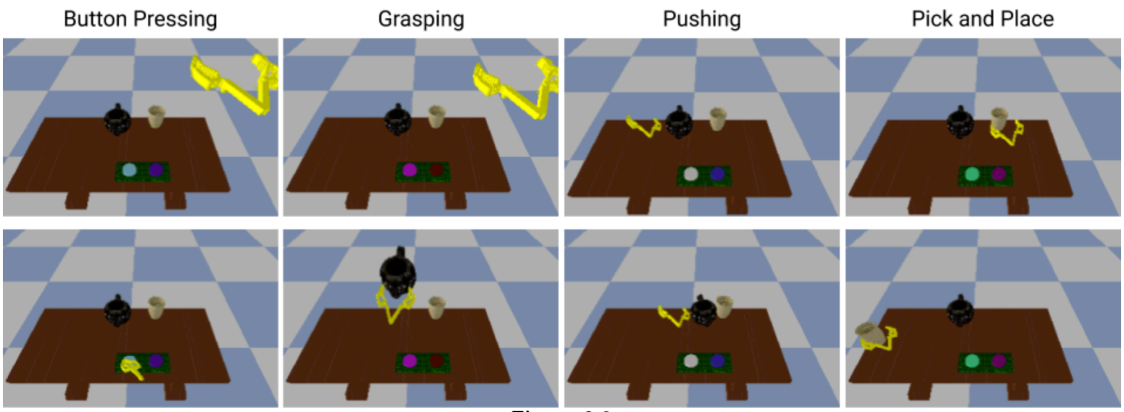


Figure 2.8
Visualization of different tasks from the simulated benchmark for Watch Try Learn.

Figure 2.8 illustrates the different task families considered in the simulated Gripper environment. Button Pressing, Grasping, Pushing, and Pick and Place. For each task family, the environment supports hundreds of different tasks by changing the objects in the scene and the objectives (e.g. which object to pick and where to place). For each task in each task family, a handful of expert demonstrations are given in a demonstrations dataset. As mentioned previously, the environment gives the agent image observations, and take in actions as end-effector (gripper) positions, angles, and opening.

Baselines: The following three baselines are considered:

1. **Behavior Cloning**: simple imitation learning based on maximum log-likelihood training using data from all tasks.
2. **Meta-imitation learning**: This baseline corresponds to simply running the policy from the Watch step, without using any trial data. That is, we only condition on the set of expert demonstrations, but no online trials.
3. **Behavior Cloning + SAC**: Pre-train a policy with Behavior Cloning on all data, and follow that with Reinforcement Learning fine-tuning for the specific target task, using the maximum-entropy algorithm SAC ((Haarnoja et al. 2018)).

Table 2.3
Average success rates over all tasks.

METHOD	SUCCESS RATE
BC	$.09 \pm .01$
MIL	$.30 \pm .02$
WTL, 1 TRIAL (OURS)	$.42 \pm .02$
RL FINE-TUNING WITH SAC	
BC + SAC, 1500 TRIALS	$.11 \pm .07$
BC + SAC, 2000 TRIALS	$.29 \pm .10$
BC + SAC, 2500 TRIALS	$.39 \pm .11$

METHOD	SUCCESS RATE

Figure 2.9 shows average success rates for Watch Try Learn compared to baselines. Watch Try Learn significantly outperforms baselines on every task family. In particular, it is far superior to Behavior Cloning, which is a very weak baseline, and it significantly surpasses Meta-Imitation Learning on 3 out of 4 task families. Table 2.3 includes comparison with BC fine-tuned with Reinforcement Learning. Even after 2500 online trials, SAC is not able to obtain the success rate that Watch Try Learn achieves after only 1 trial. Overall, Watch Try Learn exhibits very significant performance gains over prior methods.

2.4.4 Direct Preference Optimization

A modern method for estimating the parameters of a human preference model is direct preference optimization (Rafailov et al. 2023), which is used in the context of aligning language models to human preferences. A recent approach (Christiano et al. 2023) first trains a reward model that captures human preferences and then uses proximal policy optimization to train a language model-based policy to reflect those learned preferences. Direct Preference Optimization (DPO), on the other hand, removes the need for a reward model by directly using the model likelihood of two outcomes (a preferred or highly-ranked sequence and an unpreferred or low-ranked sequence) to capture the preference represented in the data. DPO provides a simpler framework than its reinforcement learning approach and results in comparable performance with improved stability. Furthermore, it obviates the need to train a reward model, instead using a language model policy and human preference dataset to align the policy directly to human preferences.

2.4.5 Model Design Consideration

When designing models and learning their parameters, one must account for important trade-offs when designing and optimizing a model to learn human preferences.

Bias vs. Variance Trade-off. In modeling human preferences, we aim to ensure that predicted utilities accurately reflect overall human preferences. One key challenge is managing the bias and variance trade-off.

Bias refers to assumptions made during model design and training that can skew predictions. For example, in Ideal Point Models, we make the assumption that the representations we use for individuals and choices are aligned in the embedding space, and that this representation is sufficient to capture human preferences using distance metrics. However, there are myriad cases in which this may break down, for example if the two sets of vectors follow different distributions each with their own unique biases. If the representations do not come from the same domain, one may have little visibility into how a distance metric computes the final utility value for a choice for a given individual. Some ways to mitigate bias in human preference models include increasing the number of parameters in a model (allowing for better learning

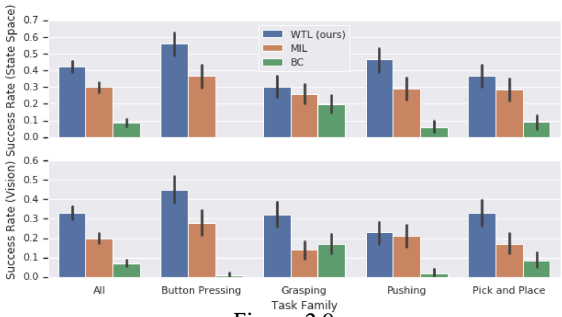


Figure 2.9
Results for Watch Try Learn on the gripper control environment, and comparisons with baselines.

of patterns in the data) or removing inductive biases based on our assumptions of the underlying data.

On the other hand, variance refers to the model's sensitivity to small changes in the input, which leads to significant changes in the outp ut. This phenomenon is often termed 'overfitting' or 'overparameterization.' This behavior can occur in models that have many parameters, and learn correlations in the data that do not contribute to learning human preferences, but are artifacts of noise in the dataset that one should ultimately ignore. One can address variance in models by reducing the number of parameters or incorporating biases in the model based on factors we can assume about the data.

Model Scope. One important consideration unique to human preference models is that we wish to model individual preferences, and we may choose to do so at arbitrary granularity. For example, we can fit models to a specific individual or even multiple models for an individual, each for different purposes or contexts. On the other end of the spectrum, we may create a model to capture human preferences across large populations or the world.

Individual models may certainly prove to be more powerful, as they do not need to generalize across multiple individuals and can dedicate all of their parameters to learning the preferences of a single user. In the context of human behavior, this can be a significant advantage as any two individuals can be arbitrarily different or even opposite in their preferences. On the other hand, models fit only one person can tremendously overfit to the training distribution and capture noise in the data, which is not truly representative of human preferences.

On the end of the spectrum, models fit to the entire world may be inadequate to model human preferences for arbitrary individuals, especially those whose data it has not been fit to. As such, models may underfit the given training distribution. These models aim to generalize to many people but may fail to capture the nuances of individual preferences, especially for those whose data is not represented in the training set. As a result, they may not perform well for arbitrary individuals within the target population

Choosing the appropriate scope for a model is crucial. ne must balance the trade-off between overfitting to noise in highly granular models and underfitting in broader models that may not capture individual nuances.

2.5 Multimodal Preferences

One of the core assumptions about learning a reward function is that it is unimodal, meaning that it consists of data from one person with a certain set of preferences or a group of people with similar preferences. However, the model of unimodality often oversimplifies human preferences and their often conflicting nature. To accurately capture all the nuances of human preference, we examine a multi-modal distribution with some baseline assumptions. Consider a scenario where we, as regular drivers, make a left-hand turn at an intersection (Myers et al. 2021). What would we do if we saw a car speeding down the road approaching us? The figure below describes some options. Following a timid driving pattern, some vehicles would stop to let the other car go, preventing a collision. Other vehicles would be more aggressive and try to make the turn before colliding with the oncoming vehicle. Given the data of one of these driving patterns, our model (our autonomous vehicle) can make an appropriate decision. However, what if our model was given data from both aggressive and timid drivers, and we don't know which data corresponds to which type of driver? If we applied standard learning based on comparison techniques, we see, as illustrated by the figure below, that the car would have an accident trying to find a policy close enough to both driving patterns.

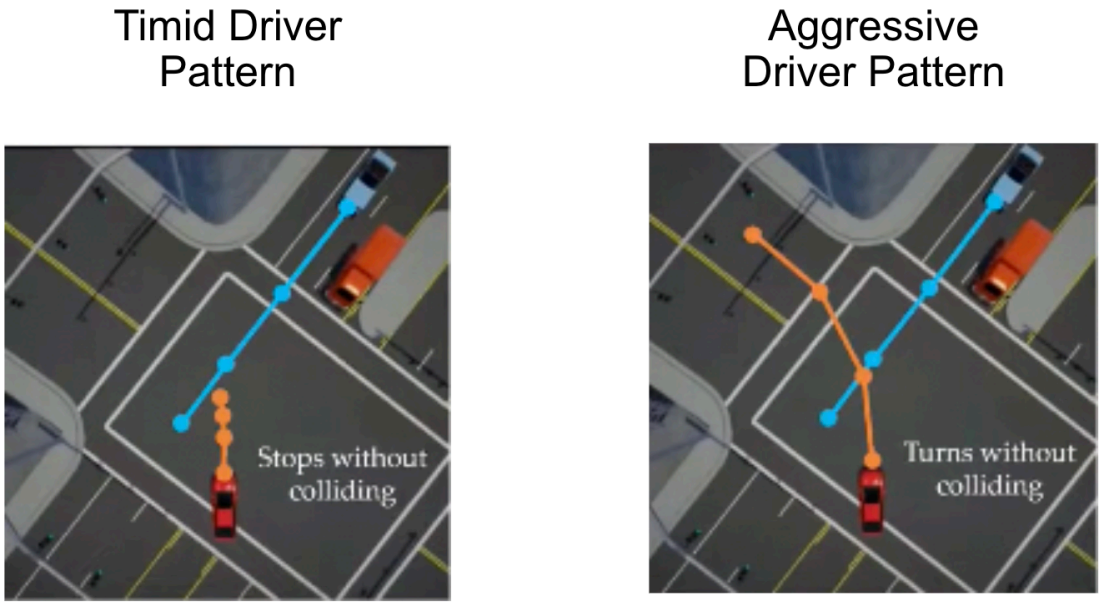


Figure 2.10
(Myers et al. 2021) shows the possibilities of 2 different driving patterns when a car is taking a left-hand turn at an intersection and sees another car approaching head-on.

As illustrated by the driving example, we see that multi-modality for our reward function is extremely important and, in some cases, if it is not considered, can lead to fatal decisions (Myers et al. 2021). But why can't we label the groups, which would be the timid and aggressive drivers in the driving case, and then learn separate reward functions for each driver? The first problem with this approach is that it is inefficient and time-consuming to separate the data into groups because we would have to cluster and label the data. Secondly, it would not be

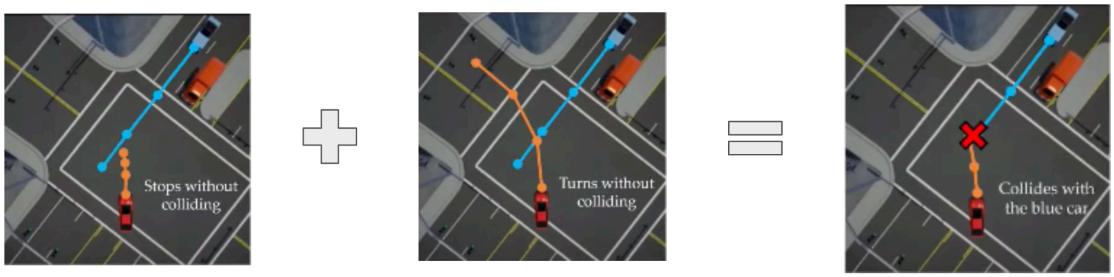


Figure 2.11
The figure (Myers et al. 2021) depicts the resultant collision when we try to find a policy close enough to both the driving patterns.

accurate just to split the data because a more timid driver can be aggressive when they are in a hurry.

To formulate this problem of learning reward functions and mixing coefficients from ranking queries in a fully observable deterministic dynamical system, we begin by describing the system as a trajectory $\xi = (s_0, a_0, ..., s_T, a_T)$, where the sequence of states and actions represents the system's evolution over time. Assume there are M different reward functions, each representing an expert's preferences. Using the linearity assumption in reward learning, we model each expert's reward function as a linear combination of features in a known, fixed feature space $\phi(\xi)$. The reward for the m-th expert is given by:

$$R_m(\xi) = \omega_m^T \phi(\xi),$$

where ω_m is a vector of parameters corresponding to the m-th expert's preferences. There exists an unknown distribution over the reward parameters and we can represent this distribution with mixing coefficients α_m such that $\sum_M^{m=1} \alpha_m = 1$. Our goal is to learn reward functions and mixing coefficients using ranking queries.

To define our problem, let's consider a robot who performs the following trajectories and asks a user to rank all the trajectories.

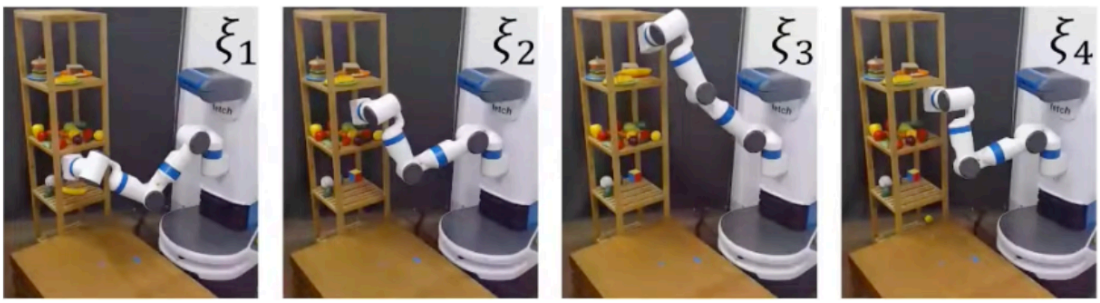


Figure 2.12
The figure (Myers et al. 2022) depicts a few different trajectories for an example multi-modal ranking scenario.

The robot will be given back a set of trajectory rankings, coming from M humans and the objective is to learn the underlying reward function. We can represent the response of the

ranking query as $x = (\xi_{a_1}, \ldots, \xi_{a_K})$ where a_1 is the index of the expert's top choice, a_2 is the index of the expert's second choice, ... and so on. With the response x, we generate a probability distribution with the softmax rule (Myers et al. 2022): $Pr(x_1 = \xi_{a_1} | R = R_m) = \frac{e^R_m(\xi_{a_1})}{\sum_{j=1}^K e^R_m(\xi_{a_j})}$. where $R_m(\xi_{a_i})$ denotes the reward assigned by the m-th expert to trajectory ξ_{a_i}. Then, we randomly sample our probability distribution to pick our top choice. From the remaining trajectories, we noisily choose from our distribution to rank our second-best option. We repeat this process until we have ranked all our trajectories. This follows what is known as the Plackett-Luce Ranking Model.

Given knowledge of the true reward function weights ω_m and mixing coefficients α_m, we have the following joint mass over observations x from a query Q: $Pr(x \mid Q) = \sum_{m=1}^M \alpha_m \prod_{i=1}^K \frac{e^{\omega_m^T \Phi(\xi_{a_i})}}{\sum_{j=i}^K e^{\omega_m^T \Phi(\xi_{a_j})}}$.

With the above formulation of the joint mass distribution over observation and queries, we can now formulate an objective. Specifically, it is to present users with the best set of queries that learn reward weights, ω, and mixing coefficient, α, based upon user rankings of preferred query responses. By learning these parameters, we can have an accurate estimation of the joint mass distribution of the observations.

To learn these parameters, we use a Bayesian learning framework. The goal will be to learn the reward weights, ω_m, and all mixing coefficients α_m. Thus, define the parameters to be $\theta = \{\omega, \alpha\}$. We start by simplifying the posterior over the parameters.

$$\begin{aligned}
\Pr(\Theta | Q^{(1)}, x^{(1)}, Q^{(2)}, x^{(2)}, \ldots) &\propto \Pr(\Theta) \Pr(Q^{(1)} | x^{(1)}, Q^{(2)}, x^{(2)}, \ldots | \Theta) \\
&= \Pr(\Theta) \prod_t \Pr(x^{(t)} | Q^{(t)}, \Theta, Q^{(1)}, x^{(1)}, \ldots, Q^{(t-1)}, x^{(t-1)}) \\
&\propto \Pr(\Theta) \prod_t \Pr(x^{(t)} | \Theta, Q^{(t)})
\end{aligned}$$

Note that the first proportionality term is directly from Bayes rule (removing normalization constant). The first equation comes directly from the assumption that the queries at timestamp t are conditionally independent of the parameters given previous queries & rankings. This assumption is reasonable because the previous queries & rankings ideally give all the information to inform the choice of the next set of. The last proportionality term comes from the assumption that the ranked queries are conditionally independent given the parameters

The prior distribution is dependent on use case. For example, in the user studies conducted by the authors to verify this method, they use a standard Gaussian for the reward weights and the mixing coefficients to be uniform on a $M - 1$ simplex to ensure that they add up to 1. Then we can use maximum likelihood estimation to compute the parameters with the simplified posterior.

2.6 Social Choices

Game theory provides a mathematical framework for analyzing strategic interactions among rational agents. These models help in understanding and predicting human behavior by considering multiple criteria and the associated trade-offs. They enhance the understanding of preferences across multiple criteria and allow for richer and more accurate feedback through structured comparisons. Game-theory framings capture the complexity of preferences and interactions in decision-making processes (Bhatia et al. 2020).

The most popular form of preference elicitation involves pairwise comparisons. Users are asked to choose between two options, such as product A or product B. This method is used in various applications like search engines, recommender systems, and interactive robotics. Key concepts include the Von Neumann Winner and the Blackwell Winner. The Von Neumann Winner refers to a distribution over objects that beats or ties every other object in the collection under the expected utility assumption. The Blackwell Winner generalizes the Von Neumann Winner for multi-criteria problems using a target set for acceptable payoff vectors (Bhatia et al. 2020).

Game-theory framings provide a framework for preference learning along multiple criteria. These models use tools from vector-valued payoffs in game theory, with Blackwell's approach being a key concept. This approach allows for a more comprehensive understanding of preferences by considering multiple criteria simultaneously (Bhatia et al. 2020).

In game-theory framings, pairwise preferences are modeled as random variables. Comparisons between objects along different criteria are captured in a preference tensor P. This tensor models the probability that one object is preferred over another along a specific criterion, allowing for a detailed understanding of preferences across multiple dimensions (Bhatia et al. 2020).

The preference tensor P captures object comparisons along different criteria. It is defined as:

$$P(i_1, i_2; j) = P(i_1 \succ i_2 \text{ along criterion } j)$$

where $P(i_2, i_1; j) = 1 - P(i_1, i_2; j)$. These values are aggregated to form an overall preference matrix P_{ov} (Bhatia et al. 2020).

The Blackwell Winner is defined using a target set S of acceptable score vectors. The goal is to find a distribution π^* such that $P(\pi^*, \pi) \in S$ for all π. This method minimizes the maximum distance to the target set, providing a robust solution to multi-criteria preference problems (Bhatia et al. 2020).

The optimization problem for finding the Blackwell Winner is defined as:

$$\pi(P, S, \|\cdot\|) = \arg\min_{\pi \in \Delta_d} \left[\max_{\pi' \in \Delta_d} \rho(P(\pi, \pi'), S) \right]$$

where $\rho(u, v) = \|u - v\|$. This measures the distance to the target set, ensuring that the selected distribution is as close as possible to the ideal preference vector (Bhatia et al. 2020).

2.7 Exercises

Question 1: Choice Modeling (15 points)

In Chapter 2, we discussed discrete choice modeling in the context of utility being a linear function. Suppose we are deciding between N choices and that the utility of each choice is given by $U_i = \beta_i \mathbf{x} + \epsilon_i$ for $i = 1, 2, \cdots, N$. We view $\mathbf{x}$ as the data point that is being conditioned on for deciding which choice to select, and β_i as the weights driving the linear utility model. The noise ϵ_i is i.i.d. sampled from a type of extreme value distribution called the *Gumbel* distribution. The standard Gumbel distribution is given by the density function $f(x) = e^{-(x+e^{-x})}$ and cumulative distribution function $F(x) = e^{-e^{-x}}$. Fix i. Our objective is to calculate $\Pr(U_i \text{ has max utility})$.

(a) **(Written, 2 points).** To start, set $U_i = t$ and compute $\Pr(U_j < t)$ for $j \neq i$ in terms of F. Use this probability to derive an integral for $\Pr(U_i \text{ has max utility})$ over t in terms of f and F.

Example of solution environment.

(b) **(Written, 4 points).** Compute the integral derived in part (a) with the appropriate u-substitution. Show your work. You should arrive at multi-class logistic regression in the end!

Next, you will implement logistic regression to predict preferred prompt completions. We will use the preference dataset from RewardBench[3]. Notice the provided `data/chosen_embeddings.pt` and `data/rejected_embeddings.pt` files. These files were constructed by feeding the prompt alongside the chosen/rejected responses through Llama3-8B-Instruct and selecting the last token's final hidden embedding. Let e_1 and e_2 be two hidden embeddings with $e_1 \succ e_2$. We assume weights w exist for which the Bradley-Terry reward of an embedding e can be modeled as $r = w \cdot e$. In this setting, the probability of $e_1 \succ e_2$ is

$$\frac{e^{w \cdot e_1}}{e^{w \cdot e_1} + e^{w \cdot e_2}} = \frac{1}{1 + e^{w \cdot (e_2 - e_1)}} = \sigma(w \cdot (e_1 - e_2)).$$

Hence, we can view maximum likelihood across the preference dataset with this model as logistic regression on $e_1 - e_2$ without a bias term and all labels being 1.

In biasless logistic regression, we are given a dataset X with N rows of datapoints and D features per datapoint. The weights of the model are parametrized by θ, a D-dimensional column vector. Given binary labels y of shape N by 1, the binary cross-entropy loss is

$$J(\theta) = -\frac{1}{N}(y^T \log(\sigma(X\theta)) + (1 - y)^T \log(1 - \sigma(X\theta)))$$

where σ is the sigmoid function and is applied element-wise along with log. The gradient of loss is

$$\nabla_\theta J(\theta) = \frac{1}{N} X^T (\sigma(X\theta) - y).$$

[3] https://huggingface.co/datasets/allenai/reward-bench

1. **(Coding, 3 points).** Open the file `logistic_regression/logistic_regression.py`. Implement the function `train` in the biasless case.
2. **(Coding, 2 points).** Implement the function `predict_probs`.
3. **(Written, 4 points).** Open the notebook `rewardbench_preferences.ipynb` and run all the cells. Make sure to tune the `learning_rate` and `num_iterations`. Report your final expected accuracy on the training and validation sets. How close are the two expected accuracies? You should be able to achieve $\approx 90\%$ expected accuracy on validation. You may add loss reporting to the `train` function to verify your model is improving over time.

```python
from sklearn.model_selection import train_test_split
import torch

class LogisticRegression:
    def __init__(self):
        self.weights = None  # Initialized during training

    def train(self, X, y, learning_rate, num_iterations):
        """
        Train the logistic regression model using gradient descent (no bias).
        Each gradient update should be with respect to the entire dataset X.

        Parameters:
        - X (torch.Tensor): Training data of shape (n_samples, n_features).
        - y (torch.Tensor): Target labels of shape (n_samples,).
        """
        n_samples, n_features = X.shape

        # Initialize weights without the bias term
        self.weights = torch.zeros(n_features)

        for i in range(num_iterations):
            # YOUR CODE HERE (~4-5 lines)
                pass
            # END OF YOUR CODE

    def predict_probs(self, X):
        """
        Predict probabilities for samples in X (no bias).

        Parameters:
        - X (torch.Tensor): Input data of shape (n_samples, n_features).

        Returns:
        - y_probs (torch.Tensor): Predicted probabilities.
        """
```

```
        y_probs = None

        # YOUR CODE HERE (~2-3 lines)
        pass
        # END OF YOUR CODE

        return y_probs


if __name__ == "__main__":
    # %%
    # Load in Llama3 embeddings of prompt + completions on RewardBench
    chosen_embeddings = torch.load('data/chosen_embeddings.pt')
    rejected_embeddings = torch.load('data/rejected_embeddings.pt')

    # Subtract the embeddings according to the Bradley-Terry reward model setup presented in the problem
    X = (chosen_embeddings - rejected_embeddings).to(torch.float)
    y = torch.ones(X.shape[0])

    # Split dataset 80/20 into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    print("Training set size:", X_train.shape)
    print("Validation set size:", X_val.shape)

    model = LogisticRegression()

    # Tune the learning_rate and num_iterations until you achieve expected validation accuracy of at least 90%
    learning_rate = None
    num_iterations = None

    model.train(X_train, y_train, learning_rate=learning_rate, num_iterations=num_iterations)

    y_train_probs = model.predict_probs(X_train)
    print(f"Expected Train Accuracy: {y_train_probs.mean()}")

    y_val_probs = model.predict_probs(X_val)
    print(f"Expected Validation Accuracy: {y_val_probs.mean()}") # Should reach at least 90%
```

Question 2: Revealed and Stated Preferences (20 points)

Alice and Bob are running for president. For R voters, we have access to their revealed candidate preferences through some means (e.g., social media, blogs, event history). Assume there

is an underlying probability z of voting for Alice among the population that is unknown. The aim of this question is to estimate z through *maximum likelihood estimation* by also incorporating stated preferences. In this scenario, we collect stated preferences through surveys. When surveyed, voters tend to be more likely to vote for Alice with probability $\frac{z+1}{2}$ for reasons of "political correctness."

(a) **(Written, 5 points)**. Suppose there are R_A revealed preferences for Alice, R_B revealed preferences for Bob, S_A stated preferences for Alice, and S_B stated preferences for Bob. Note $R = R_A + R_B$. Compute the log-likelihood of observing such preferences in terms of z, R_A, R_B, S_A, S_B.

(b) **(Coding, 1 point)**. Implement the short function `stated_prob` in the file `voting/simulation.py`.

(c) **(Coding, 5 points)**. Implement the class `VotingSimulation`.

(d) **(Coding, 7 points)**. Implement your derived expression from part (a) in the `log_likelihoods` function.

(e) **(Written, 2 points)**. Finally, implement the `average_mae_mle` method that will allow us to visualize the mean absolute error (MAE) of our maximum likelihood estimate $\hat{z}$ (i.e., $|\hat{z} - z|$) as the number of voters surveyed increases. Open `voting/visualize_sim.ipynb` and run the cells to get a plot of MAE vs. voters surveyed averaged across 100 simulations. Attach the plot to this question and briefly explain what you notice.

```python
import torch
import random
import matplotlib.pyplot as plt
from tqdm import tqdm
random.seed(42)
torch.manual_seed(42)

def stated_prob(z_values):
    """
    Computes the probability of stated preferences based on z values.

    Args:
        z_values (torch.Tensor): The z value(s), where z represents the true probability of voting for Alice.

    Returns:
        torch.Tensor: Probability for stated preferences, derived from z values.
    """
    # YOUR CODE HERE (~1 line)
    # END OF YOUR CODE

class VotingSimulation:
```

```
    """
    A class to simulate the voting process where revealed and stated preferences are generated.

    Attributes:
        R (int): Number of revealed preferences.
        z (float): The true probability of voting for Alice.
        revealed_preferences (torch.Tensor): Simulated revealed preferences of R voters using Bernoulli distribution.
                                             Takes on 1 for Alice, and 0 for Bob.
        stated_preferences (torch.Tensor): Simulated stated preferences, initialized as an empty tensor.
                                           Takes on 1 for Alice, and 0 for Bob.
    """
    def __init__(self, R, z):
        self.R = R
        self.z = z
        self.revealed_preferences = None # YOUR CODE HERE (~1 line)
        self.stated_preferences = torch.tensor([])

    def add_survey(self):
        """
        Simulates an additional stated preference based on stated_prob and adds it to the list.
        This updates the self.stated_preferences tensor by concatenating on a new simulated survey result.
        """
        # YOUR CODE HERE (~3 lines)
        # END OF YOUR CODE

def log_likelihoods(revealed_preferences, stated_preferences, z_values):
    """
    Computes the log likelihoods across both revealed and stated preferences.
    Use your answer in part (a) to help.

    Args:
        revealed_preferences (torch.Tensor): Tensor containing revealed preferences (0 or 1).
        stated_preferences (torch.Tensor): Tensor containing stated preferences (0 or 1).
        z_values (torch.Tensor): Tensor of underlying z values to calculate likelihood for.

    Returns:
        torch.Tensor: Log likelihood for each z value.
    """
    # YOUR CODE HERE (~10-16 lines)
    pass
```

```
    # END OF YOUR CODE

def average_mae_mle(R, z, survey_count, num_sims, z_sweep):
    """
    Runs multiple simulations to compute the average mean absolute error (MAE) of
    Maximum Likelihood Estimation (MLE)
    for z after increasing number of surveys.

    Args:
        R (int): Number of revealed preferences.
        z (float): The true probability of voting for Alice.
        survey_count (int): Number of additional surveys to perform.
        num_sims (int): Number of simulation runs to average over.
        z_sweep (torch.Tensor): Range of z values to consider for maximum likelihood
    estimation.

    Returns:
        torch.Tensor: Tensor of mean absolute errors averaged over simulations.
                      Should have shape (survey_count, )
    """
    all_errors = []
    for _ in tqdm(range(num_sims)):
        errors = []
        vote_simulator = VotingSimulation(R=R, z=z)

        for _ in range(survey_count):
            revealed_preferences = vote_simulator.revealed_preferences
            stated_preferences = vote_simulator.stated_preferences

            # YOUR CODE HERE (~6-8 lines)
            pass # Compute log_likelihoods across z_sweep. Argmax to find MLE for z.
                 # Append the absolute error to errors and add a survey to the
                  simulator.
            # END OF YOUR CODE

        errors_tensor = torch.stack(errors)
        all_errors.append(errors_tensor)

    # Calculate the average error across simulations
    mean_errors = torch.stack(all_errors).mean(dim=0)
    return mean_errors

if __name__ == "__main__":
    # DO NOT CHANGE!
    max_surveys = 2000
    z = 0.5
    R = 10
    num_sims = 100
```

```
    z_sweep = torch.linspace(0.01, 0.99, 981)

    # Compute and plot the errors. Attach this plot to part (d).
    mean_errors = average_mae_mle(R, z, max_surveys, num_sims, z_sweep)
    plt.plot(mean_errors)

    plt.xlabel('Surveys Conducted')
    plt.ylabel('Average Error')
    plt.title(f'MLE MAE Error (z={z}, {num_sims} simulations)')
    plt.show()
```

Question 3: Probabilistic Multi-modal Preferences (25 points)

Suppose you are part of the ML team on the movie streaming site CardinalStreams. After taking CS329H, you collect a movie preferences dataset with 30000 examples of the form $(m_1, m_2, \text{user id})$ where m_1 and m_2 are movies with $m_1 \succ m_2$. The preferences come from 600 distinct users with 50 examples per user. Each movie has a 10-dimensional feature vector m, and each user has a 10-dimensional weight vector u. Given movie features m_1, m_2 and user weights u, the user's preference between the movies is given by a Bradley-Terry reward model, i.e.,

$$P(m_1 \succ m_2) = \frac{e^{u \cdot m_1}}{e^{u \cdot m_1} + e^{u \cdot m_2}} = \frac{1}{1 + e^{u \cdot (m_2 - m_1)}} = \sigma(u \cdot (m_1 - m_2)).$$

You realize that trying to estimate the weights for each user with only 50 examples will not work due to the lack of data. Instead, you choose to drop the user IDs column and shuffle the dataset in order to take a *multi-modal preferences* approach. For simplicity, you assume a model where a proportion p of the users have weights w_1 and the other $1 - p$ have weights w_2. In this setting, each user belongs to one of two groups: users with weights w_1 are part of Group 1, and users with weights w_2 are part of Group 2.

(a) **(Written, 3 points).** For a datapoint (m_1, m_2) with label $m_1 \succ m_2$, compute the data likelihood $P(m_1 \succ m_2 | p, w_1, w_2)$ assuming p, w_1, w_2 are given.

(b) **(Written, 3 points).** As a follow up, use the likelihood to simplify the posterior distribution of p, w_1, w_2 after updating on (m_1, m_2) leaving terms for the priors unchanged.

(c) **(Written, 4 points).** Assume priors $p \sim B(1, 1)$, $w_1 \sim \mathcal{N}(0, \mathbf{I})$, and $w_2 \sim \mathcal{N}(0, \mathbf{I})$ where B represents the Beta distribution and $\mathcal{N}$ represents the normal distribution. You will notice that the posterior from part (b) has no simple closed-form. As a result, we must resort to *Markov Chain Monte Carlo (MCMC)* approaches to sample from the posterior. These approaches allow sampling from highly complex distributions by constructing a Markov chain $\{x_t\}_{t=1}^{\infty}$ so that $\lim_{t \to \infty} x_t$ act as desired samples from the target distribution. You can think of a Markov chain as a sequence with the special property that x_{t+1} only depends on x_t for all $t \geq 1$.

The most basic version of MCMC is known as Metropolis-Hastings. Assume π is the target distribution we wish to sample from where $\pi(z)$ represents the probability density at point z. Metropolis-Hastings constructs the approximating Markov chain x_t as follows: a proposal P for x_{t+1} is made via sampling from a chosen distribution $Q(\cdot|x_t)$ (e.g., adding Gaussian noise). The acceptance probability of the proposal is given by

$$A = \min\left(1, \frac{\pi(P)Q(x_t|P)}{\pi(x_t)Q(P|x_t)}\right).$$

That is,

$$x_{t+1} = \begin{cases} P & \text{with probability } A, \\ x_t & \text{with probability } 1 - A. \end{cases}$$

To extract our samples from π, we run the Markov chain for N timesteps and disregard the first $T < N$ timesteps in what is called the *burn-in or mixing time* (i.e., our final samples are $x_{T+1}, x_{T+2}, \cdots, x_N$). The mixing time is needed to ensure that the Markov chain elements are representative of the distribution π – initial elements of the chain will not be a good approximation of π and depend more on the choice of initialization x_1.

To build some intuition, suppose we have a biased coin that turns heads with probability p_{heads}. We observe 12 coin flips to have 9 heads and 3 tails. If our prior for p_{heads} was $B(1,1)$, then our posterior will be $B(1+9, 1+3) = B(10,4)$. The Bayesian update is given by

$$\begin{aligned} P(p_{\text{heads}}|9 \text{ heads}, 3 \text{ tails}) &= \frac{P(9 \text{ heads}, 3 \text{ tails}|p_{\text{heads}})B(1,1)(p_{\text{heads}})}{\int_0^1 P(9 \text{ heads}, 3 \text{ tails}|p_{\text{heads}})B(1,1)(p_{\text{heads}})dp_{\text{heads}}} \\ &= \frac{P(9 \text{ heads}, 3 \text{ tails}|p_{\text{heads}})}{\int_0^1 P(9 \text{ heads}, 3 \text{ tails}|p_{\text{heads}})dp_{\text{heads}}}. \end{aligned}$$

Find the acceptance probablity A in the setting of the biased coin assuming the proposal distribution $Q(\cdot|x_t) = x_t + N(0, \sigma)$ for given σ. Notice that this choice of Q is symmetric, i.e., $Q(x_t|P) = Q(P|x_t)$. In addition, you will realize that is unnecessary to compute the normalizing constant of the Bayesian update (i.e., the integral in the denominator) which is why MCMC is commonly used to sample from posteriors!

(d) **(Written + Coding, 6 points).** Implement Metropolis-Hastings to sample from the posterior distribution of the biased coin in `multimodal_preferences/biased_coin.py`. Attach a histogram of your MCMC samples overlayed on top of the true posterior $B(10,4)$ by running `python biased_coin.py`.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta

```

```python
def likelihood(p: float) -> float:
    """
    Computes the likelihood of 9 heads and 3 tails assuming p_heads is p.

    Args:
    p (float): A value between 0 and 1 representing the probability of heads.

    Returns:
    float: The likelihood value at p_heads=p. Return 0 if p is outside the range [0, 1].
    """
    # YOUR CODE HERE (~1-3 lines)
    pass
    # END OF YOUR CODE


def propose(x_current: float, sigma: float) -> float:
    """
    Proposes a new sample from the proposal distribution Q.
    Here, Q is a normal distribution centered at x_current with standard deviation sigma.

    Args:
    x_current (float): The current value in the Markov chain.
    sigma (float): Standard deviation of the normal proposal distribution.

    Returns:
    float: The proposed new sample.
    """
    # YOUR CODE HERE (~1-3 lines)
    pass
    # END OF YOUR CODE


def acceptance_probability(x_current: float, x_proposed: float) -> float:
    """
    Computes the acceptance probability A for the proposed sample.
    Since the proposal distribution is symmetric, Q cancels out.

    Args:
    x_current (float): The current value in the Markov chain.
    x_proposed (float): The proposed new value.

    Returns:
    float: The acceptance probability
    """
    # YOUR CODE HERE (~4-6 lines)
    pass
```

```
    # END OF YOUR CODE


def metropolis_hastings(N: int, T: int, x_init: float, sigma: float) -> np.ndarray:
    """
    Runs the Metropolis-Hastings algorithm to sample from a posterior distribution.

    Args:
    N (int): Total number of iterations.
    T (int): Burn-in period (number of initial samples to discard).
    x_init (float): Initial value of the chain.
    sigma (float): Standard deviation of the proposal distribution.

    Returns:
    list: Samples collected after the burn-in period.
    """
    samples = []
    x_current = x_init

    for t in range(N):
        # YOUR CODE HERE (~7-10 lines)
        # Use the propose and acceptance_probability functions to get x_{t+1} and
        ↪  store it in samples after the burn-in period T
        pass
        # END OF YOUR CODE

    return samples


def plot_results(samples: np.ndarray) -> None:
    """
    Plots the histogram of MCMC samples along with the true Beta(10, 4) PDF.

    Args:
    samples (np.ndarray): Array of samples collected from the Metropolis-Hastings
 ↪  algorithm.

    Returns:
    None
    """
    # Histogram of the samples from the Metropolis-Hastings algorithm
    plt.hist(samples, bins=50, density=True, alpha=0.5, label="MCMC Samples")

    # True Beta(10, 4) distribution for comparison
    p = np.linspace(0, 1, 1000)
    beta_pdf = beta.pdf(p, 10, 4)
    plt.plot(p, beta_pdf, "r-", label="Beta(10, 4) PDF")

```

```
    plt.xlabel("p_heads")
    plt.ylabel("Density")
    plt.title("Metropolis-Hastings Sampling of Biased Coin Posterior")
    plt.legend()
    plt.show()


if __name__ == "__main__":
    # MCMC Parameters (DO NOT CHANGE!)
    N = 50000  # Total number of iterations
    T = 10000  # Burn-in period to discard
    x_init = 0.5  # Initial guess for p_heads
    sigma = 0.1  # Standard deviation of the proposal distribution

    # Run Metropolis-Hastings and plot the results
    samples = metropolis_hastings(N, T, x_init, sigma)
    plot_results(samples)
```

(e) **(Coding, 9 points).** Implement Metropolis-Hastings in the movie setting inside `multimodal_preferences/movie_metropolis.py`. The movie dataset we use for grading will not be provided. However, randomly constructed datasets can be used to test your implementation by running `python movie_metropolis.py`. You should be able to achieve a 90% success rate with most `fraction_accepted` values above 0.1. Success is measured by thresholded closeness of predicted parameters to true parameters. You may notice occasional failures that occur due to lack of convergence which we will account for in grading.

```
import torch
import torch.distributions as dist
import math
from tqdm import tqdm
from typing import Tuple

def make_data(
    true_p: torch.Tensor, true_weights_1: torch.Tensor, true_weights_2: torch.Tensor,
 ↪  num_movies: int, feature_dim: int
) -> Tuple[torch.Tensor, torch.Tensor]:
    """
    Generates a synthetic movie dataset according to the CardinalStreams model.

    Args:
        true_p (torch.Tensor): Probability of coming from Group 1.
        true_weights_1 (torch.Tensor): Weights for Group 1.
        true_weights_2 (torch.Tensor): Weights for Group 2.

    Returns:
```

```
        Tuple[torch.Tensor, torch.Tensor]: A tuple containing the dataset and labels.
    """
    # Create movie features
    first_movie_features = torch.randn((num_movies, feature_dim))
    second_movie_features = torch.randn((num_movies, feature_dim))

    # Only care about difference of features for Bradley-Terry
    dataset = first_movie_features - second_movie_features

    # Get probabilities that first movie is preferred assuming Group 1 or Group 2
    weight_1_probs = torch.sigmoid(dataset @ true_weights_1)
    weight_2_probs = torch.sigmoid(dataset @ true_weights_2)

    # Probability that first movie is preferred overall can be viewed as sum of conditioning on Group 1 and Group 2
    first_movie_preferred_probs = (
        true_p * weight_1_probs + (1 - true_p) * weight_2_probs
    )
    labels = dist.Bernoulli(first_movie_preferred_probs).sample()
    return dataset, labels


def compute_likelihoods(
    dataset: torch.Tensor,
    labels: torch.Tensor,
    p: torch.Tensor,
    w_1: torch.Tensor,
    w_2: torch.Tensor,
) -> torch.Tensor:
    """
    Computes the likelihood of each datapoint. Use your calculation from part (a) to help.

    Args:
        dataset (torch.Tensor): The dataset of differences between movie features.
        labels (torch.Tensor): The labels where 1 indicates the first movie is preferred, and 0 indicates preference of the second movie.
        p (torch.Tensor): The probability of coming from Group 1.
        w_1 (torch.Tensor): Weights for Group 1.
        w_2 (torch.Tensor): Weights for Group 2.

    Returns:
        torch.Tensor: The likelihoods for each datapoint. Should have shape (dataset.shape[0], )
    """
    # YOUR CODE HERE (~6-8 lines)
    pass
    # END OF YOUR CODE
```

```python

def compute_prior_density(
    p: torch.Tensor, w_1: torch.Tensor, w_2: torch.Tensor
) -> torch.Tensor:
    """
    Computes the prior density of the parameters.

    Args:
        p (torch.Tensor): The probability of preferring model 1.
        w_1 (torch.Tensor): Weights for model 1.
        w_2 (torch.Tensor): Weights for model 2.

    Returns:
        torch.Tensor: The prior densities of p, w_1, and w_2.
    """
    # Adjusts p to stay in the range [0.3, 0.7] to prevent multiple equilibria issues
     ↪  at p=0 and p=1
    p_prob = torch.tensor([2.5]) if 0.3 <= p <= 0.7 else torch.tensor([0.0])

    def normal_pdf(x: torch.Tensor) -> torch.Tensor:
        """Computes the PDF of the standard normal distribution at x."""
        return (1.0 / torch.sqrt(torch.tensor(2 * math.pi))) * torch.exp(-0.5 * x**2)

    weights_1_prob = normal_pdf(w_1)
    weights_2_prob = normal_pdf(w_2)

    # Concatenate the densities
    concatenated_prob = torch.cat([p_prob, weights_1_prob, weights_2_prob])
    return concatenated_prob


def metropolis_hastings(
    dataset: torch.Tensor,
    labels: torch.Tensor,
    sigma: float = 0.01,
    num_iters: int = 30000,
    burn_in: int = 20000,
) -> Tuple[torch.Tensor, torch.Tensor, torch.Tensor, float]:
    """
    Performs the Metropolis-Hastings algorithm to sample from the posterior
 ↪  distribution.
    DO NOT CHANGE THE DEFAULT VALUES!

    Args:
        dataset (torch.Tensor): The dataset of differences between movie features.
        labels (torch.Tensor): The labels indicating which movie is preferred.
        sigma (float, optional): Standard deviation for proposal distribution.
            Defaults to 0.01.
```

```
        num_iters (int, optional): Total number of iterations. Defaults to 30000.
        burn_in (int, optional): Number of iterations to discard as burn-in.
            Defaults to 20000.

    Returns:
        Tuple[torch.Tensor, torch.Tensor, torch.Tensor, float]: Samples of p,
        w_1, w_2, and the fraction of accepted proposals.
    """
    feature_dim = dataset.shape[1]

    # Initialize random starting parameters by sampling priors
    curr_p = 0.3 + 0.4 * torch.rand(1)
    curr_w_1 = torch.randn(feature_dim)
    curr_w_2 = torch.randn(feature_dim)

    # Keep track of samples and total number of accepted proposals
    p_samples = []
    w_1_samples = []
    w_2_samples = []
    accept_count = 0

    for T in tqdm(range(num_iters)):
        # YOUR CODE HERE (~3 lines)
        pass # Sample proposals for p, w_1, w_2
        # END OF YOUR CODE

        # YOUR CODE HERE (~4-6 lines)
        pass # Compute likehoods and prior densities on both the proposed and current
         ↪  samples
        # END OF YOUR CODE

        # YOUR CODE HERE (~2-4 lines)
        pass # Obtain the ratios of the likelihoods and prior densities between the
         ↪  proposed and current samples
        # END OF YOUR CODE

        # YOUR CODE HERE (~1-2 lines)
        pass # Multiply all ratios (both likelihoods and prior densities) and use this
         ↪  to calculate the acceptance probability of the proposal
        # END OF YOUR CODE

        # YOUR CODE HERE (~4-6 lines)
        pass # Sample randomness to determine whether the proposal should be accepted
         ↪  to update curr_p, curr_w_1, curr_w_2, and accept_count
        # END OF YOUR CODE

        # YOUR CODE HERE (~4-6 lines)
```

```
        pass # Update p_samples, w_1_samples, w_2_samples if we have passed the burn in period T
        # END OF YOUR CODE

    fraction_accepted = accept_count / num_iters
    print(f"Fraction of accepted proposals: {fraction_accepted}")
    return (
        torch.stack(p_samples),
        torch.stack(w_1_samples),
        torch.stack(w_2_samples),
        fraction_accepted,
    )


def evaluate_metropolis(num_sims: int, num_movies: int, feature_dim: int) -> None:
    """
    Runs the Metropolis-Hastings algorithm N times and compare estimated parameters with true parameters to obtain success rate. You should attain a success rate of around 90%.

    Note that there are two successful equilibria to converge to. They are true_weights_1 and true_weights_2 with probabilities
    p and 1-p in addition to true_weights_2 and true_weights_1 with probabilities 1-p and p. This is why even though it may appear your
    predicted parameters don't match the true parameters, they are in fact equivalent.

    Args:
        num_sims (int): Number of simulations to run.

    Returns:
        None
    """

    success_count = 0
    for _ in range(num_sims):
        # Sample random ground truth parameters
        true_p = 0.3 + 0.4 * torch.rand(1)
        true_weights_1 = torch.randn(feature_dim)
        true_weights_2 = torch.randn(feature_dim)

        print("\n---- MCMC Simulation ----")
        print("True parameters:", true_p, true_weights_1, true_weights_2)

        dataset, labels = make_data(true_p, true_weights_1, true_weights_2, num_movies, feature_dim)
        p_samples, w_1_samples, w_2_samples, _ = metropolis_hastings(dataset, labels)

```

```
        p_pred = p_samples.mean(dim=0)
        w_1_pred = w_1_samples.mean(dim=0)
        w_2_pred = w_2_samples.mean(dim=0)

        print("Predicted parameters:", p_pred, w_1_pred, w_2_pred)

        # Do casework on two equilibria cases to check for success
        p_diff_case_1 = torch.abs(p_pred - true_p)
        p_diff_case_2 = torch.abs(p_pred - (1 - true_p))

        w_1_diff_case_1 = torch.max(torch.abs(w_1_pred - true_weights_1))
        w_1_diff_case_2 = torch.max(torch.abs(w_1_pred - true_weights_2))

        w_2_diff_case_1 = torch.max(torch.abs(w_2_pred - true_weights_2))
        w_2_diff_case_2 = torch.max(torch.abs(w_2_pred - true_weights_1))

        pass_case_1 = (
            p_diff_case_1 < 0.1 and w_1_diff_case_1 < 0.5 and w_2_diff_case_1 < 0.5
        )
        pass_case_2 = (
            p_diff_case_2 < 0.1 and w_1_diff_case_2 < 0.5 and w_2_diff_case_2 < 0.5
        )
        passes = pass_case_1 or pass_case_2

        print(f'Result: {"Success" if passes else "FAILED"}')
        if passes:
            success_count += 1
    print(f'Success rate: {success_count / num_sims}')


if __name__ == "__main__":
    evaluate_metropolis(num_sims=10, num_movies=30000, feature_dim=10)
```

Question 4: Direct Preference Optimization (40 points)

Note this question requires a GPU which is provided for free on Google Colab (T4 instance) or through the course cloud credits provided on Ed.

Direct Preference Optimization (DPO) allows for policy alignment on a preference dataset without the need to train a separate reward model. The preference dataset is constructed by sampling generations $(y_1, y_2) \sim \pi_{\text{ref}}(\cdot \mid x)$ where π_{ref} is the base policy to be aligned, and x comes from a set of previously collected prompts. The pairs of generations are then labeled by an annotator for which of the generations is preferred. Denote the preference dataset by $\mathcal{D} = \left\{\left(x^{(i)}, y_+^{(i)}, y_-^{(i)}\right)\right\}_{i=1}^{N}$, where y_+ and y_- are the preferred and non-preferred generations,

respectively. DPO aims to solve the following:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{(x,y_+,y_-)\sim\mathcal{D}} \left[-\log \sigma \left(\beta \log \left(\frac{\pi(y_+|x)}{\pi_{\text{ref}}(y_+|x)} \right) - \beta \log \left(\frac{\pi(y_-|x)}{\pi_{\text{ref}}(y_-|x)} \right) \right) \right]$$

where Π is the space of possible polices π can take on. π is typically parametrized.

(a) **(Written, 6 points).** Consider the setting where π_{ref} has no conditioning features and randomly outputs one of two possible values, **A** or **B** (also known as the "Bandit" setting). Suppose that $\pi_{\text{ref}}(\mathbf{A}) = p_0$ and $\pi_{\text{ref}}(\mathbf{B}) = 1-p_0$. Furthermore, assume that the preference dataset $\mathcal{D}$ is infinitely large, sampled from π_{ref}, and that the preferred response is selected through a Bradley-Terry reward model where **A** has reward score r_A and **B** has reward score r_B. Set $\Pi = \{\pi_p \mid 0 < p < 1\}$ where π_p is the policy defined by $\pi_p(\mathbf{A}) = p$ and $\pi_p(\mathbf{B}) = 1 - p$. The DPO objective is to compute:

$$\pi_{\hat{p}} = \arg\min_{\pi_p \in \Pi} f(p, p_0, \beta, r_A, r_B),$$

for a function f. Find f by explicitly computing the relevant expectation.

(b) **(Written, 8 points).** Assume that a solution to the optimization problem in part (a) exists. Find an expression for $\hat{p}$. (Hint: Make sure to know your sigmoid derivative properties! Everything should simplify nicely. You may use the *logit function*[4] denoted by σ^{-1} in your final expression.)

(c) **(Written, 3 points).** Show that $\lim_{\beta\to\infty} \hat{p} = p_0$. (Very) briefly explain why this makes sense intuitively based on the role of β in KL-constrained reward optimization (we suggest two sentences).

(d) **(Written, 3 points).** Assume $r_A = r_B$ and $\beta > 0$. Notice that $\hat{p} = p_0$. Briefly explain why this makes sense intuitively (we suggest two sentences).

Next, you will fine-tune the lightweight 2 billion parameter Gemma 2 model on the DPO objective. We will use the instruction fine-tuned variant of the model (i.e., designed for chat-based interactions).

1. **(Coding, 4 points).** Open the `dpo/dpo.ipynb` file of the PSET's codebase. Execute the first few cells of the notebook until you see the `sample_chat_tokens` and their IDs printed out. The next cell requires you to implement the `get_response_idxs` function in `dpo/dpo.py`.

   To implement it, you must find the indices of the first and last token of the model's response in `sample_chat_tokens`. In the notebook's example, this corresponds to the tokens "As" and ".".

[4] https://en.wikipedia.org/wiki/Logit

2. **(Coding, 4 points).** The following cell asks you to implement the `get_response_next_token_probs` function. The next token logits for each token of the chat prompt are provided. Pass them through the softmax function and appropriately index the next token IDs.

   ```
   <bos><start_of_turn>user
   Where are you?<end_of_turn>
   <start_of_turn>model
   I am here.<end_of_turn>
   ```

   In the example above, we look for the next-token probabilities of "I", "am", "here", and "." To do so, you must extract the logits for "\n", "I", "am", and "here" because the probability of generating a given token comes from the prediction of the token before. Use the return value of `get_response_idxs` as anchor points for indexing. Be careful of off-by-one indexing mistakes!

3. **(Coding, 6 points).** The training and reference LLM policies are loaded for you. We load the training policy in with LoRA for computational efficiency during fine-tuning in the next part. Implement `compute_dpo_objective` with the objective provided in the theory portion for your favorite positive value of β. Does β affect the loss printed out? Why or why not? You do not need to write why in your submission, but this line of thinking will help debug any issues with your DPO loss function.

4. **(Written + Coding, 6 points).** Finally, you will fine-tune the Gemma model on the DPO loss function with batch size (and dataset size) of 1 by implementing `finetune`. The prompt and completions are provided in the notebook. The optimizer, β, and the number of fine-tuning steps have also been provided. Make sure to use `torch.no_grad()` on the reference model to prevent unnecessary gradients!

   Report the proportion of "because of" occurences before and after fine-tuning. Additionally, include a plot of the DPO loss curve.

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, set_seed
from peft import LoraConfig, get_peft_model

set_seed(42) # DO NOT CHANGE THE SEED

def get_response_idxs(tokenizer, chat_token_ids):
    """
    Finds the start and end indices of the response in the tokenized chat.

    Args:
    tokenizer: The tokenizer object used to encode/decode text.
    chat_token_ids (list[int]): The token IDs representing the chat conversation.

    Returns:
```

```
    tuple: A tuple (response_start_idx, response_end_idx), both of which are nonnegative integers.
    """

    start_of_turn_id = tokenizer.convert_tokens_to_ids("<start_of_turn>")
    end_of_turn_id = tokenizer.convert_tokens_to_ids("<end_of_turn>")

    response_start_idx = None # Nonnegative integer
    response_end_idx = None # Nonnegative integer

    # YOUR CODE HERE (~3-5 lines)
    pass
    # END OF YOUR CODE

    return response_start_idx, response_end_idx

def get_response_next_token_probs(tokenizer, model, chat_token_ids):
    """
    Computes the next token probabilities for the response in a chat.

    Args:
    tokenizer: The tokenizer object used to encode/decode text.
    model: The language model used to generate the logits.
    chat_token_ids (list[int]): The token IDs representing the chat conversation.

    Returns:
    torch.Tensor: A 1D tensor containing the probabilities of the tokens in the response found by appropriately indexing
                  the next token probabilities of the preceding token.
    """

    response_start_idx, response_end_idx = get_response_idxs(tokenizer, chat_token_ids)
    chat_token_ids_tensor = torch.tensor([chat_token_ids]).to(model.device)
    logits = model(chat_token_ids_tensor).logits[0, :, :] # shape (len(chat_token_ids), vocabulary_size)

    next_token_probs = None # Should be a 1D-tensor

    # YOUR CODE HERE (~3-5 lines)
    pass
    # END OF YOUR CODE

    return next_token_probs

def compute_dpo_objective(preferred_train_probs, nonpreferred_train_probs, preferred_ref_probs, nonpreferred_ref_probs, beta):
    """
```

```
    Computes the Direct Preference Optimization (DPO) objective for training.

    Args:
    preferred_train_probs (torch.Tensor): Token probabilities for the preferred chat sequence from the training model.
    nonpreferred_train_probs (torch.Tensor): Token probabilities for the non-preferred chat sequence from the training model.
    preferred_ref_probs (torch.Tensor): Token probabilities for the preferred chat sequence from the reference model.
    nonpreferred_ref_probs (torch.Tensor): Token probabilities for the non-preferred chat sequence from the reference model.
    beta (float): Controls the KL strength of staying close to the reference model.

    Returns:
    torch.Tensor: The computed DPO objective, which is a float.
    """

    dpo_obj = None # Float value

    # YOUR CODE HERE (~4-6 lines)
    pass
    # END OF YOUR CODE

    return dpo_obj

def finetune(tokenizer, optimizer, train_model, ref_model, preferred_chat_ids, nonpreferred_chat_ids, num_gradient_steps, beta):
    """
    Fine-tunes the training model using DPO. Make sure to disable gradients on the reference model!

    Args:
    tokenizer: The tokenizer object used to encode/decode text.
    optimizer: The optimizer for updating the training model's parameters.
    train_model: The model being fine-tuned.
    ref_model: The reference model.
    preferred_chat_ids (list[int]): The token IDs representing the preferred chat sequence.
    nonpreferred_chat_ids (list[int]): The token IDs representing the non-preferred chat sequence.
    num_gradient_steps (int): The number of gradient updates to perform.
    beta (float): A parameter used in computing the DPO objective.

    Returns:
    None
    """

    print('Fine-tuning...')
```

```
    for i in range(num_gradient_steps):
        # YOUR CODE HERE (~9-12 lines)
        pass
        # END OF YOUR CODE
    print("Fine-tuning complete!")

# DO NOT CHANGE!
def sample_model(tokenizer, model, prompt, N=100):
    """
    Samples N different completions from the model based on the given prompt.

    Args:
    tokenizer: The tokenizer object used to encode/decode text.
    model: The language model used for generation.
    prompt (str): The input prompt for which completions will be generated.
    N (int): The number of completions to generate.

    Returns:
    list[str]: A list of N generated completions.
    """

    chat = [{"role": "user", "content": prompt}]
    chat_tokens = tokenizer.apply_chat_template(chat, tokenize=True,
    ↪ add_generation_prompt=True)

    # Generate N different responses
    outputs = model.generate(
        torch.tensor([chat_tokens], device=model.device),
        num_return_sequences=N,
        max_new_tokens=32,
        temperature=0.15,
        top_k=50,
        top_p=0.95,
        do_sample=True
    )

    def extract_response(decoded_text):
        return decoded_text.rsplit('model\n', 1)[-1][:-2]

    responses = [extract_response(tokenizer.decode(output, skip_special_tokens=True))
    ↪ for output in outputs]
    return responses

# DO NOT CHANGE!
def fraction_responses_with_because_of(responses):
    """
    Calculates the fraction of responses that start with a specific match string.

```

```
    Args:
    responses (list[str]): A list of model-generated responses.

    Returns:
    float: The fraction of responses that start with the phrase "The sky appears blue because of".
    """

    match_str = "The sky appears blue because of"
    match_count = 0

    for response in responses:
        if response.startswith(match_str):
            match_count += 1

    return match_count / len(responses)


if __name__ == '__main__':
    model = AutoModelForCausalLM.from_pretrained(
        "google/gemma-2-2b-it",
        torch_dtype=torch.bfloat16,
        device_map='auto'
    )
    tokenizer = AutoTokenizer.from_pretrained("google/gemma-2-2b-it")

    sample_prompt = "How is it going?"
    sample_completion = "As an AI, I don't have feelings or experiences like humans do, so I don't have a \"going\" in the same way."

    sample_chat = [
        {"role": "user", "content": sample_prompt},
        {"role": "assistant", "content": sample_completion}
    ]

    sample_chat_tokens = tokenizer.apply_chat_template(sample_chat, tokenize=False, add_generation_prompt=False)
    sample_chat_token_ids = tokenizer.apply_chat_template(sample_chat, tokenize=True, add_generation_prompt=False)

    print("Chat tokens:")
    print(sample_chat_tokens)

    print("Chat token IDs:")
    print(sample_chat_token_ids)

    response_start_idx, response_end_idx = get_response_idxs(tokenizer, sample_chat_token_ids)
```

```
    print(f"Response tokens index in sample_chat_tokens range from {response_start_idx} to {response_end_idx}.")

    first_response_token_id = sample_chat_token_ids[response_start_idx]
    last_response_token_id = sample_chat_token_ids[response_end_idx]
    print(f'First response token is "{tokenizer.decode(first_response_token_id)}" with ID {first_response_token_id}')
    print(f'Last response token is "{tokenizer.decode(last_response_token_id)}" with ID {last_response_token_id}')

    # Make sure your code passes this test!
    assert tokenizer.decode(first_response_token_id) == "As" and tokenizer.decode(last_response_token_id) == "."

    with torch.no_grad():
        next_token_probs = get_response_next_token_probs(tokenizer, model, sample_chat_token_ids)
    print(f'Next token probabilities: {next_token_probs}')

    # Make sure your code passes this test!
    assert next_token_probs.mean() > 0.7

    train_model = AutoModelForCausalLM.from_pretrained(
        "google/gemma-2-2b-it",
        torch_dtype=torch.bfloat16,
        device_map='auto'
    )
    lora_config = LoraConfig()
    train_model = get_peft_model(train_model, lora_config)
    train_model.train()

    ref_model = model
    ref_model.train()
    print('Loaded models!')

    # The model's response to the prompt usually includes the words "due to" - we want to change that to "because of" using DPO!
    prompt = "Explain why the sky is blue in one sentence."
    preferred_completion = "The sky appears blue because of"
    nonpreferred_completion = "The sky appears blue due to"

    preferred_chat = [
        {"role": "user", "content": prompt},
        {"role": "assistant", "content": preferred_completion}
    ]

    nonpreferred_chat = [
        {"role": "user", "content": prompt},
```

```
        {"role": "assistant", "content": nonpreferred_completion}
    ]

    preferred_chat_ids = tokenizer.apply_chat_template(preferred_chat, tokenize=True, add_generation_prompt=False)
    nonpreferred_chat_ids = tokenizer.apply_chat_template(nonpreferred_chat, tokenize=True, add_generation_prompt=False)

    preferred_train_probs = get_response_next_token_probs(tokenizer, train_model, preferred_chat_ids)
    nonpreferred_train_probs = get_response_next_token_probs(tokenizer, train_model, nonpreferred_chat_ids)

    # Gradients are not needed for the reference model since we will not be optimizing with respect to it
    with torch.no_grad():
        preferred_ref_probs = get_response_next_token_probs(tokenizer, ref_model, preferred_chat_ids)
        nonpreferred_ref_probs = get_response_next_token_probs(tokenizer, ref_model, nonpreferred_chat_ids)

    your_favorite_beta = 1.0 # Feel free to play with beta here. Does anything change?
    dpo_obj = compute_dpo_objective(preferred_train_probs, nonpreferred_train_probs, preferred_ref_probs, nonpreferred_ref_probs, beta=your_favorite_beta)
    print(dpo_obj)

    prior_responses = sample_model(tokenizer, train_model, prompt)
    print('Sampled responses before fine-tuning:\n' + '\n'.join(prior_responses[:10]))
    print(f'Fraction responses with because of: {fraction_responses_with_because_of(prior_responses)}') # should start close to 0

    # DO NOT CHANGE THESE VALUES
    num_gradient_steps = 150
    learning_rate = 2e-6
    beta = 1
    optimizer = torch.optim.Adam(train_model.parameters(), lr=learning_rate)

    finetune(tokenizer, optimizer, train_model, ref_model, preferred_chat_ids, nonpreferred_chat_ids, num_gradient_steps, beta)

    # Save GPU memory
    del ref_model
    del model

    post_tuning_responses = sample_model(tokenizer, train_model, prompt)
    print('Sampled responses after fine-tuning:\n' + '\n'.join(post_tuning_responses[:10]))
```

```
print(f'Fraction responses with because of:
↪ {fraction_responses_with_because_of(post_tuning_responses)}') # should be more
↪ than half
```

References

Bhatia, Kush, Ashwin Pananjady, Peter L. Bartlett, Anca D. Dragan, and Martin J. Wainwright. 2020. "Preference Learning Along Multiple Criteria: A Game-Theoretic Perspective." *Neural Information Processing Systems* 34 (1): 1–12.

Bobu, Andreea, Dexter R. R. Scobee, Jaime F. Fisac, S. Shankar Sastry, and Anca D. Dragan. 2020. "LESS Is More: Rethinking Probabilistic Models of Human Behavior." https://doi.org/10.1145/3319502.3374811.

Bock, Hans Georg, Thomas Carraro, Willi Jäger, Stefan Körkel, Rolf Rannacher, and Johannes P. Schlöder. 2015. *Model Based Parameter Estimation: Theory and Applications*. Springer. https://api.semanticscholar.org/CorpusID:60333071.

Bolt, Daniel M., and James A. Wollack. 2009. "Application of a Multidimensional Nested Logit Model to Multiple-Choice Test Items." *Journal of Educational Measurement* 46 (3): 181–98. https://doi.org/10.1111/j.1745-3984.2009.00081.x.

Bommasani, Rishi, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, et al. 2021. "On the Opportunities and Risks of Foundation Models."

Bradley, Ralph Allan, and Milton E Terry. 1952a. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

Bradley, Ralph Allan, and Milton E. Terry. 1952b. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45. http://www.jstor.org/stable/2334029.

Campbell, Danny, and Seda Erdem. 2015. "Position Bias in Best-Worst Scaling Surveys: A Case Study on Trust in Institutions." *American Journal of Agricultural Economics* 97 (2): 526–45. https://doi.org/10.1093/ajae/aau112.

Casella, George, and Roger L. Berger. 1990. *Statistical Inference*. Springer. https://api.semanticscholar.org/CorpusID:125727004.

Cattelan, Manuela. 2012. "Models for Paired Comparison Data: A Review with Emphasis on Dependent Data." *Statistical Science* 27 (3): 412–33. https://doi.org/10.1214/12-STS396.

Christiano, Paul, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2023. "Deep Reinforcement Learning from Human Preferences." https://arxiv.org/abs/1706.03741.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. 2017. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In *International Conference on Machine Learning*, 1126–35. PMLR.

Greiner, James. 2005. "Ideal Points." Harvard IQSS Blog. https://blogs.iq.harvard.edu/ideal_points_1.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In *International Conference on Machine Learning*, 1861–70. PMLR.

Harpe, Spencer E. 2015. "How to Analyze Likert and Other Rating Scale Data." *Currents in Pharmacy Teaching and Learning* 7 (5): 836–50. http://dx.doi.org/10.1016/j.cptl.2015.08.001.

Hejna III, Donald Joseph, and Dorsa Sadigh. 2023. "Few-Shot Preference Learning for Human-in-the-Loop Rl." In *Conference on Robot Learning*, 2014–25. PMLR.

Huber, Joel. 1976. "Ideal Point Models of Preference." In *Advances in Consumer Research*, 03:138–42. Association for Consumer Research.

Jamieson, Kevin G, and Robert Nowak. 2011. "Active Ranking Using Pairwise Comparisons." In *Advances in Neural Information Processing Systems*, edited by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger. Vol. 24. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2011/file/6c14da109e294d1e8155be8aa4b1ce8e-Paper.pdf.

Keisler, H. Jerome, and Byung Soo Lee. 2003. "Common Assumption of Rationality." *Economic Theory Journal* 30 (2): 123–45.

Lee, Kimin, Laura Smith, and Pieter Abbeel. 2021. "Pebble: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-Training." *arXiv Preprint arXiv:2106.05091*.

Luce, R.Duncan. 1977. "The Choice Axiom After Twenty Years." *Journal of Mathematical Psychology* 15 (3): 215–33. https://doi.org/10.1016/0022-2496(77)90032-3.

Miljkovic, Dragan. 2005. "Rational Choice and Irrational Individuals or Simply an Irrational Theory: A Critical Review of the Hypothesis of Perfect Rationality." *The Journal of Socio-Economics* 34 (5): 621–34. https://doi.org/10.1016/j.socec.2003.12.031.

Myers, Vivek, Erdem Biyik, Nima Anari, and Dorsa Sadigh. 2022. "Learning Multimodal Rewards from Rankings." In *Conference on Robot Learning*, 342–52. PMLR.

Myers, Vivek, Erdem Bıyık, Nima Anari, and Dorsa Sadigh. 2021. "Learning Multimodal Rewards from Rankings." https://arxiv.org/abs/2109.12750.

Neumann, John Von, and Oskar Morgenstern. 1945. *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press.

Padalkar, Abhishek, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, et al. 2023. "Open x-Embodiment: Robotic Learning Datasets and RT-x Models." *arXiv Preprint arXiv:2310.08864*.

Plackett, R. L. 1975. "The Analysis of Permutations." *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 24 (2): 193–202. http://www.jstor.org/stable/2346567.

Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. "Improving Language Understanding by Generative Pre-Training." San Francisco, CA, USA.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." https://arxiv.org/abs/2305.18290.

Ragain, Stephen, and Johan Ugander. 2019. "Choosing to Rank." *arXiv Preprint arXiv:1809.05139*. https://arxiv.org/abs/1809.05139.

Roy, Deepjyoti, and Mala Dutta. 2022. "A Systematic Review and Research Perspective on Recommender Systems." *Journal of Big Data* 9:1–36. https://api.semanticscholar.org/CorpusID:248508374.

Ruder, Sebastian. 2016. "An Overview of Gradient Descent Optimization Algorithms." *ArXiv* abs/1609.04747. https://api.semanticscholar.org/CorpusID:17485266.

Simon, Herbert A. 1972. "Theories of Bounded Rationality." In *Decision and Organization*, edited by C. B. McGuire and Roy Radner, 161–76. North-Holland Publishing Company.

Tatli, Gokcan, Rob Nowak, and Ramya Korlakai Vinayak. 2022. "Learning Preference Distributions from Distance Measurements." In *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 1–8. https://doi.org/10.1109/Allerton49937.2022.9929404.

Touvron, Hugo et al. 2023. "Llama 2: Open Foundation and Fine-Tuned Chat Models." https://arxiv.org/abs/2307.09288.

Yu, Tianhe, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. 2020. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning." In *Conference on Robot Learning*, 1094–1100. PMLR.

Zhou, Allan, Eric Jang, Daniel Kappler, Alex Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn. 2019. "Watch, Try, Learn: Meta-Learning from Demonstrations and Rewards." In *International Conference on Learning Representations*.

3 Model-Based Preference Optimization

3.1 Active Preference Learning

3.1.1 Introduction to Active Learning

In real-world scenarios, data is often scarce, and acquiring labeled data can be expensive. Active learning is a machine learning paradigm that aims to reduce the amount of labeled data required to train a model to achieve high accuracy. Active learning algorithms iteratively select an input datapoint for an oracle (e.g., a human annotator) to label such that when the label is observed, the model improves the most. The goal of AL algorithms is to minimize the number of labels required to achieve a desired level of performance. This technique is particularly useful in situations where labeling data is expensive, time-consuming, or requires domain expertise.

There are two primary setups in active learning:

- **Pool-based**: The model selects samples from a large unlabeled pool of data. For example, a model for text classification selects the most uncertain texts from a large pool to ask a human annotator to label.
- **Stream-based**: The model receives samples sequentially (one sample at a time) and decides whether to label them. The data is gone if the decision maker decides not to label it. For example, a system monitoring sensor data decides on-the-fly whether new sensor readings are valuable enough to label.

A common AL process is shown in Figure 3.1:

- Current model trained on current dataset $\mathcal{D}$, potential points $\tilde{x}_1 \dots \tilde{x}_m$ are being investigated. AL will choose one of them to add to the dataset.
- Relative to the model, a proxy highlights the relative value of each point to model improvement $(v(\tilde{x}_1) \dots v(\tilde{x}_m))$. A naive proxy is the model's uncertainty about the point.
- The cycle repeats until we collect enough data or the model is good enough.

Active learning has been successfully applied to various domains to enhance real-world systems, including computer vision, natural language processing, and recommender systems. For example, active learning can improve the computer vision models used in autonomous vehicles (Jarl et al. 2021), here driving scenes can take infinitely many forms, making it impossible to gather an exhaustive dataset. Instead, probing a model to understand what type of data it would benefit from is more practical. In robotics, autonomous agents may query humans when unsure how to act or when facing new situations (Taylor, Berrueta, and Murphey 2021). In this field,

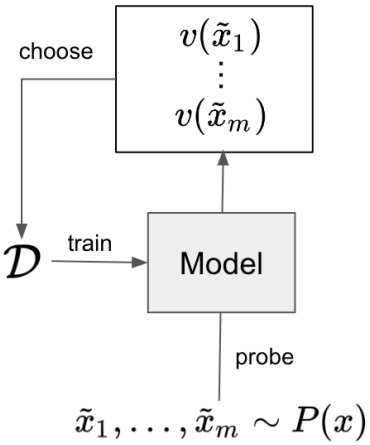


Figure 3.1
The current model is trained on the current data set $\mathcal{D}$. Potential points $\tilde{x}_1, \ldots, \tilde{x}_m$ are being investigated, and one of them will be chosen and added to the data set. A proxy highlights the relative value of each point in terms of improving the model, denoted by $v(\tilde{x}_1), \ldots, v(\tilde{x}_m)$. The point with the highest value is selected and added to $\mathcal{D}$. This cycle repeats until enough data has been collected or the model is good enough.

collecting data often incurs significant financial and time costs: the robot must act in real-time in the real world, and while parallelization is possible, being strategic about which examples to collect can best benefit the model. In meteorology, active learning can help decide where to place additional sensors for weather predictions (Singh, Nowak, and Ramanathan 2006). Sensor placement involves deploying teams to remote locations and expensive construction for an extra data point. Choosing these locations and allocating resources wisely is of interest to governments and businesses. Active learning could also be employed to select data for fine-tuning large language models (LLMs) for specific downstream tasks (Margatina et al. 2023). In this context, it might be difficult to fully describe an NLP task one might want an LLM to solve. Often, instead of defining a task via a dataset of examples, it may be easier for a human to interact with the LLM for a specific use case, identify gaps in the model, and address those using active learning.

3.1.2 Introduction to Active Preference Learning

Consider the scenario where a robot is being trained to assist individuals with feeding. How can such a robot be effectively taught to perform necessary tasks, such as determining the appropriate distance to reach, detecting the location of a person's mouth, and, most importantly, understanding human preferences? Typically, robots learn by observing human demonstrations, replicating the ways a person performs the task. However, this method poses significant challenges. Expert demonstrations are often limited, and training a supervised learning model would require vast amounts of demonstration data, which is difficult to obtain at scale. Moreover, demonstrations tend to be variable, reflecting the actions of individual humans, making the data collection process inconsistent. To address these limitations, alternative approaches

have been proposed, such as using pairwise comparisons, where humans evaluate two action trajectories to determine the superior one, or employing physical corrections, in which reward functions are learned through human–robot interactions, with humans guiding the robot's actions during the task.

Active learning algorithms can be employed in preference learning tasks, such as the previously mentioned example, where the objective is to develop a model that aligns with human preferences while minimizing the need for extensive labeled data or reducing the high cost of annotations. This chapter will explore the theoretical foundations of pairwise comparisons and active preference learning, along with extensions to these methods that address known limitations. Practical examples where these approaches prove beneficial will also be discussed. Additionally, we will examine the role of LLMs in assisting robots through corrective feedback and highlight the applications of these techniques.

3.1.3 Uncertainty Qualification

Problem Setup: In this section, we consider a binary classification problem. The model is trained on a small labeled dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where x_i represents the input data and y_i is the corresponding label. The model is uncertain about the class labels of some data points and can query an oracle to obtain the true labels of these data points. The goal is to minimize the number of queries to the oracle while maximizing the model's performance.

Uncertainty quantification (UQ) is a critical aspect of active learning that allows models to evaluate the informativeness of new data points. In machine learning (ML), two primary types of uncertainty are often considered: epistemic and aleatoric uncertainty. **Epistemic uncertainty**, or model uncertainty, arises from a lack of knowledge and can be reduced by acquiring more data. This type of uncertainty is especially significant when the model lacks confidence due to insufficient or incomplete information in its training set. On the other hand, **aleatoric uncertainty**, or data uncertainty, stems from the inherent randomness within the data itself. Unlike epistemic uncertainty, aleatoric uncertainty cannot be reduced, even with additional data, as it reflects noise or unpredictability in the real data-generating process. Several approaches exist to quantify uncertainty in active learning, each with its strengths and limitations.

Bayesian methods, such as Bayesian Neural Networks (BNNs) and Gaussian Processes (GPs), offer a principled way of estimating uncertainty by incorporating prior knowledge into the model. These approaches can generate meaningful uncertainty estimates that aid in choosing informative samples for labeling. However, they can become computationally prohibitive, especially for large and complex models, limiting their applicability in some practical scenarios.

Another common technique for uncertainty quantification is the use of **ensemble methods**, such as Random Forests or Gradient Boosting Machines. These methods involve training multiple models and combining their predictions to provide an estimate of uncertainty. Ensemble methods are relatively easy to implement and can give valuable insights into model uncertainty. However, they can be computationally expensive and may not always produce well-calibrated

uncertainty estimates. Moreover, they do not integrate prior knowledge, which can be a disadvantage in certain applications.

Conformal prediction methods also provide a framework for estimating uncertainty by offering a measure of confidence in predictions based on the conformity of a given instance with the training data. While these methods are useful in some contexts, this book focuses primarily on the Bayesian approach due to its theoretical robustness and capacity to quantify uncertainty in a more comprehensive manner.

3.1.4 Acquisition Function

Uncertainty quantification plays a vital role in **acquisition functions**, which are central to active learning strategies. These functions determine which samples are most valuable to label by evaluating their utility based on the model's current uncertainty estimates. Common acquisition functions include **uncertainty sampling** (Zhu et al. 2010), which selects samples the model is least confident about, **query-by-committee** (Beluch et al. 2018), which utilizes a set of models to choose the most uncertain samples, and **Bayesian Active Learning by Disagreement (BALD)** (Houlsby et al. 2011), which selects samples that maximize information gain by reducing model uncertainty. Through careful uncertainty quantification, acquisition functions guide the active learning process, improving the model's efficiency in learning from limited data. Other acquisition functions that can be employed include:

- **Active Thompson Sampling** (Bouneffouf et al. 2014): This method leverages the Thompson Sampling algorithm to select a posterior sample from the model's distribution and compute the expected utility of labeling using that sample. By doing so, the algorithm balances exploration and exploitation, leading to effective active learning.
- **Expected model change** (Cai, Zhang, and Zhou 2013): This approach focuses on labeling points that would have the most impact on changing the current model parameters.
- **Expected error reduction** (Mussmann et al. 2022): Points that would most effectively reduce the model's generalization error are labeled using this strategy.
- **Variance reduction** (Cohn, Ghahramani, and Jordan 1996): This approach labels points that would minimize output variance, which is one component of error. By selecting points that reduce variability in the model's predictions, it aims to improve overall performance.
- **User Centered Labeling Strategies** (Bernard et al. 2018): This approach involves actively involving the user in the labeling process by visualizing data through dimensionality reduction techniques. The user then provides labels for the compiled data based on their domain expertise and preferences. This strategy leverages user input to improve the quality and relevance of the labeled data.
- **Querying from diverse subspaces or partitions** (Ma et al. 2022): When using a forest of trees as the underlying model, the leaf nodes can represent overlapping partitions of the

feature space. This strategy selects instances from non-overlapping or minimally overlapping partitions for labeling.

- **Conformal prediction** (Makili, Sánchez, and Dormido-Canto 2012): This method predicts that a new data point will have a label similar to old data points in some specified way. The degree of similarity within the old examples is used to estimate the confidence in the prediction.
- **Mismatch-first farthest-traversal** (Zhao, Heittola, and Virtanen 2020): This strategy first prioritizes data points that are wrongly predicted by the current model compared to the nearest-neighbor prediction. The second criterion is the distance to previously selected data, with preference given to those that are farthest away. The goal is to optimize both the correction of mispredictions and the diversity of the selected data.

3.1.4.1 Uncertainty Sampling

Uncertainty sampling (Zhu et al. 2010) is a widely used acquisition function in active learning that selects data points for which the model exhibits the greatest uncertainty. This method aims to improve model performance by focusing labeling efforts on ambiguous samples, where additional information is likely to yield the greatest benefit. Let x represent the input, and $p(y|x)$ the probability distribution of the output y given x. Several acquisition strategies fall under uncertainty sampling, including **entropy sampling**, **margin sampling**, and **least confidence sampling**, each providing a unique measure of uncertainty.

- **Entropy sampling** measures uncertainty by calculating the entropy of the predicted probability distribution. The acquisition function is given by $\alpha(x) = -\sum_y p(y|x) \log p(y|x)$, with higher entropy values indicating higher uncertainty.
- **Margin sampling** focuses on the difference between the two highest predicted probabilities for a sample. The acquisition function is given by $\alpha(x) = p(y_1|x) - p(y_2|x)$, where y_1 and y_2 are two most likely classes. Smaller margins signify greater uncertainty.
- **Least confidence sampling** measures uncertainty by identifying the sample with the lowest predicted probability for its most likely class. The acquisition function is $\alpha(x) = 1 - p(y_{\text{max}}|x)$, where y_{max} is the class with the highest probability.

Example: Consider a binary classification problem with two classes y_1 and y_2. We have three samples x_1, x_2, x_3 and the corresponding predictive distributions are as follows:

$$\begin{aligned} p(y_1|x_1) &= 0.6, \quad p(y_2|x_1) = 0.4 \\ p(y_1|x_2) &= 0.3, \quad p(y_2|x_2) = 0.7 \\ p(y_1|x_3) &= 0.8, \quad p(y_2|x_3) = 0.2 \end{aligned} \tag{3.1}$$

- **Entropy Sampling**
  - $\alpha(x_1) = -0.6 \log(0.6) - 0.4 \log(0.4) = 0.29$

  - $\alpha(x_2) = -0.3\log(0.3) - 0.7\log(0.7) = 0.27$
  - $\alpha(x_3) = -0.8\log(0.8) - 0.2\log(0.2) = 0.22$

We would select x_1 for labeling as it has the highest entropy, indicating the model is most uncertain about its prediction at x_1.

- **Margin Sampling**
  - $\alpha(x_1) = 0.6 - 0.4 = 0.2$
  - $\alpha(x_2) = 0.7 - 0.3 = 0.4$
  - $\alpha(x_3) = 0.8 - 0.2 = 0.6$

We would select x_1 for labeling as it has the smallest margin, indicating the model is most uncertain about the prediction at x_1.

- **Least Confidence Sampling**
  - $\alpha(x_1) = 1 - 0.6 = 0.4$
  - $\alpha(x_2) = 1 - 0.7 = 0.3$
  - $\alpha(x_3) = 1 - 0.8 = 0.2$

We would select x_1 for labeling as it has the lowest confidence, indicating the model is most uncertain about the prediction at x_1.

In summary, uncertainty sampling methods, whether based on entropy, margin, or least confidence, help prioritize data points that the model struggles with the most. By focusing on these uncertain samples, the model can more efficiently improve its performance, making uncertainty sampling a key tool in active learning.

3.1.4.2 Query-by-Committee

Query-by-Committee (Beluch et al. 2018) is an active learning strategy where a committee of models selects samples for labeling based on the level of disagreement among the committee members. Several acquisition functions can be employed under this framework to quantify the disagreement:

- **Vote Entropy**: The vote entropy measures the uncertainty based on how often the committee members vote for each class. The acquisition function is defined as $\alpha(x) = \mathbb{H}\left[\frac{V(y)}{C}\right]$, where $V(y)$ is the number of votes for class y and C is the number of committee members.
- **Consensus Entropy**: This acquisition function measures the entropy of the average probability distribution across committee members. It is given by $\alpha(x) = \mathbb{H}[P_C(y|x)]$, where $P_C(y|x)$ is the average probability distribution for sample x across all committee members.

– **KL Divergence**: The KL divergence quantifies the disagreement by comparing the probability distribution of each committee member to the average distribution. The acquisition function is given by $\alpha(x) = \frac{1}{C}\sum_{c=1}^{C} D_{KL}[P_c(y|x)||P_C(y|x)]$, where $P_c(y|x)$ is the probability distribution of committee member c and $P_C(y|x)$ is the average distribution across the committee.

Example: Consider a binary classification problem with two classes y_1 and y_2. We have three committee members and three samples: x_1, x_2, and x_3. The predictive distributions for each committee member are given below:

x	$p_1(y_1\|\cdot)$	$p_1(y_2\|\cdot)$	$p_2(y_1\|\cdot)$	$p_2(y_2\|\cdot)$	$p_3(y_1\|\cdot)$	$p_3(y_2\|\cdot)$
x_1	0.6	0.4	0.7	0.3	0.3	0.7
x_2	0.3	0.7	0.4	0.6	0.4	0.6
x_3	0.8	0.2	0.9	0.1	0.7	0.3

Query-by-Committee: Vote Entropy

– For sample x_1, the votes for y_1 and y_2 are $V(y_1) = 2$ and $V(y_2) = 1$. The vote entropy is $\alpha(x_1) = -\frac{2}{3}\log(\frac{2}{3}) - \frac{1}{3}\log(\frac{1}{3}) = 0.28$.
– For sample x_2, the votes are $V(y_1) = 0$ and $V(y_2) = 3$, resulting in vote entropy $\alpha(x_2) = 0$.
– For sample x_3, the votes are $V(y_1) = 3$ and $V(y_2) = 0$, resulting in vote entropy $\alpha(x_3) = 0$.

Thus, sample x_1 would be selected for labeling as it has the highest vote entropy, indicating the greatest disagreement among the committee members.

Query-by-Committee: Consensus Entropy

The first step is to compute the consensus probability of each class for each sample:

– For x_1, $p_c(y_1|x_1) = \frac{0.6+0.7+0.3}{3} = 0.53$ and $p_c(y_2|x_1) = \frac{0.4+0.3+0.7}{3} = 0.47$.
– For x_2, $p_c(y_1|x_2) = \frac{0.3+0.4+0.4}{3} = 0.37$ and $p_c(y_2|x_2) = \frac{0.7+0.6+0.6}{3} = 0.63$.
– For x_3, $p_c(y_1|x_3) = \frac{0.8+0.9+0.7}{3} = 0.8$ and $p_c(y_2|x_3) = \frac{0.2+0.1+0.3}{3} = 0.2$.

Next, we compute the entropy of these consensus probabilities:

– For x_1, $\mathbb{H}[p_c(y|x_1)] = -0.53\log(0.53) - 0.47\log(0.47) = 0.30$.
– For x_2, $\mathbb{H}[p_c(y|x_2)] = -0.37\log(0.37) - 0.63\log(0.63) = 0.29$.
– For x_3, $\mathbb{H}[p_c(y|x_3)] = -0.8\log(0.8) - 0.2\log(0.2) = 0.22$.

Thus, x_1 would be selected for labeling as it has the highest consensus entropy, indicating the highest level of disagreement among the committee members.

3.1.4.3 *Bayesian Active Learning by Disagreement*

Bayesian Active Learning by Disagreement (BALD) (Houlsby et al. 2011) selects the samples for which the model expects to gain the most Shannon information when corresponding labels are observed:

$$\mathbb{I}(\theta; y|x, \mathcal{D}) = \mathbb{H}[p(y|x, \mathcal{D})] - \mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x, \theta, \mathcal{D})]] \tag{3.2}$$

where $\mathbb{H}[\cdot]$ denotes entropy. When there is significant disagreement among models, the predictive entropy (the first term) will be large, while the expected entropy (the second term) will be smaller. This difference represents the degree to which the models disagree. BALD selects points where this disagreement is maximized.

– To compute the first term, we can derive the following expression:

$$\begin{aligned} \mathbb{H}[p(y|x, \mathcal{D})] &= \mathbb{H}\left[\int_\theta p(y|x, \theta, \mathcal{D}) p(\theta|\mathcal{D}) d\theta\right] \\ &\approx \mathbb{H}\left[\frac{1}{N}\sum_{i=1}^{N} p(y|x, \theta_i, \mathcal{D})\right] \\ &= \mathbb{H}\left[\overline{p}(y|x, \mathcal{D})\right] \end{aligned} \tag{3.3}$$

– To compute the second term, we can derive the following expression:

$$\begin{aligned} \mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x, \theta, \mathcal{D})]] &= \mathbb{E}_{p(\theta|\mathcal{D})}\left[-\sum_y p(y|x, \theta, \mathcal{D}) \log p(y|x, \theta, \mathcal{D})\right] \\ &\approx -\frac{1}{N}\sum_{i=1}^{N}\left(\sum_y p(y|x, \theta_i, \mathcal{D}) \log p(y|x, \theta_i, \mathcal{D})\right) \end{aligned} \tag{3.4}$$

Example: Consider a binary classification problem with two classes, y_1 and y_2. We have two samples, x_1 and x_2, and the model's predictive distributions are as follows:

– **First-time inference** (with $\theta_1 \sim p(\theta|\mathcal{D})$):

$$p(y_1|x_1, \theta_1, \mathcal{D}) = 0.6, \quad p(y_2|x_1, \theta_1, \mathcal{D}) = 0.4 \tag{3.5}$$

$$p(y_1|x_2, \theta_1, \mathcal{D}) = 0.4, \quad p(y_2|x_2, \theta_1, \mathcal{D}) = 0.6 \tag{3.6}$$

– **Second-time inference** (with $\theta_2 \sim p(\theta|\mathcal{D})$):

$$p(y_1|x_1, \theta_2, \mathcal{D}) = 0.8, \quad p(y_2|x_1, \theta_2, \mathcal{D}) = 0.2 \tag{3.7}$$

$$p(y_1|x_2, \theta_2, \mathcal{D}) = 0.5, \quad p(y_2|x_2, \theta_2, \mathcal{D}) = 0.5 \tag{3.8}$$

Solution:

Step 1: Compute the entropy of the model's predictive distribution for each sample:

- $\overline{p}_\theta(y_1|x_1, \theta, \mathcal{D}) = 0.7$
- $\overline{p}_\theta(y_2|x_1, \theta, \mathcal{D}) = 0.3$
- $\overline{p}_\theta(y_1|x_2, \theta, \mathcal{D}) = 0.45$
- $\overline{p}_\theta(y_2|x_2, \theta, \mathcal{D}) = 0.55$

Now, we compute the entropy for each sample using the formula:

$$\mathbb{H}[p(y|x, \mathcal{D})] = -p(y_1|x, \mathcal{D}) \log(p(y_1|x, \mathcal{D})) - p(y_2|x, \mathcal{D}) \log(p(y_2|x, \mathcal{D})) \tag{3.9}$$

For x_1:

$$\mathbb{H}[p(y|x_1, \mathcal{D})] = -0.7 \log(0.7) - 0.3 \log(0.3) = 0.27 \tag{3.10}$$

For x_2:

$$\mathbb{H}[p(y|x_2, \mathcal{D})] = -0.45 \log(0.45) - 0.55 \log(0.55) = 0.30 \tag{3.11}$$

Step 2: Compute the expected entropy of the model's predictive distribution for each sample.

For x_1:

- $\mathbb{H}_{\theta_1}[p(y|x_1, \theta, \mathcal{D})] = -0.6 \log(0.6) - 0.4 \log(0.4) = 0.29$
- $\mathbb{H}_{\theta_2}[p(y|x_1, \theta, \mathcal{D})] = -0.8 \log(0.8) - 0.2 \log(0.2) = 0.22$

Average the results:

$$\mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x_1, \theta, \mathcal{D})]] \approx \frac{0.29 + 0.22}{2} = 0.255 \tag{3.12}$$

For x_2:

- $\mathbb{H}_{\theta_1}[p(y|x_2, \theta, \mathcal{D})] = -0.4 \log(0.4) - 0.6 \log(0.6) = 0.29$
- $\mathbb{H}_{\theta_2}[p(y|x_2, \theta, \mathcal{D})] = -0.5 \log(0.5) - 0.5 \log(0.5) = 0.30$

Average the results:

$$\mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x_2, \theta, \mathcal{D})]] \approx \frac{0.29 + 0.30}{2} = 0.295 \tag{3.13}$$

Step 3: Compute the BALD score for each sample.

The BALD score $\alpha(x)$ is the difference between the predictive entropy and the expected entropy:

For x_1:

$$\alpha(x_1) = \mathbb{H}[p(y|x_1, \mathcal{D})] - \mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x_1, \theta, \mathcal{D})]] = 0.27 - 0.255 = 0.015 \tag{3.14}$$

For x_2:

$$\alpha(x_2) = \mathbb{H}[p(y|x_2, \mathcal{D})] - \mathbb{E}_{p(\theta|\mathcal{D})}[\mathbb{H}[p(y|x_2, \theta, \mathcal{D})]] = 0.30 - 0.295 = 0.005 \tag{3.15}$$

We would select x_1 for labeling since it has the highest BALD score, indicating that labeling x_1 will provide the most information gain for the model.

3.1.5 Active Learning by Variance Reduction

Active Learning by Variance Reduction (Cohn, Ghahramani, and Jordan 1996) is an algorithm designed to select the next data point for labeling based on the anticipated reduction in the model's variance. The objective is to identify the point $\tilde{x} \sim p(x)$ that, when labeled ($y(\tilde{x})$), will most effectively decrease the model's variance. To quantify the expected error at a given input x, we can mathematically express it as follows:

$$\mathbb{E}_{\hat{y}\sim p(\hat{y}|\mathcal{D};x), y\sim p(y|x)}(\hat{y} - y)^2 \tag{3.16}$$

In Equation 3.16, $\hat{y}$ represents the model's prediction, while y denotes the true label corresponding to the input x. This formulation captures the average squared difference between the predicted and actual values, providing a measure of the model's accuracy. Utilizing concepts from bias-variance decomposition as outlined in the literature (Geman, Bienenstock, and Doursat 1992), we can expand the expected error term. The expansion is given by:

$$\begin{aligned}
\mathbb{E}_{\hat{y}\sim p(\hat{y}|\mathcal{D};x), y\sim p(y|x)}(\hat{y} - y)^2 &= \mathbb{E}_{\hat{y},y}[(\hat{y} - \mathbb{E}[y|x]) + (\mathbb{E}[y|x] - y)]^2 \\
&= \mathbb{E}_{\hat{y},y}[(y - \mathbb{E}[y|x])^2] \\
&+ 2\mathbb{E}_{\hat{y},y}[(\hat{y} - \mathbb{E}[y|x])(\mathbb{E}[y|x] - y)] \\
&+ \mathbb{E}_{\hat{y},y}(\hat{y} - \mathbb{E}[y|x])^2
\end{aligned} \tag{3.17}$$

In Equation 3.17, the first term represents the variance of the true label y, the second term evaluates to zero, and the third term accounts for the variance of the model's prediction $\hat{y}$. To clarify why the second term is zero, we note that:

$$\mathbb{E}_{\hat{y},y}[\mathbb{E}[y|x] - y] = 0 \tag{3.18}$$

This indicates that the expected deviation of the true label from its conditional mean is null, as $\mathbb{E}[y|x]$ is, by definition, the average of y given x. Focusing on the third term, we derive it as follows:

$$\begin{aligned}\mathbb{E}_{\hat{y},y}(\hat{y} - \mathbb{E}[y|x])^2 &= \mathbb{E}_{\hat{y},y}[(\hat{y} - \mathbb{E}_{\hat{y}}[\hat{y}] + \mathbb{E}_{\hat{y}}[\hat{y}] - \mathbb{E}[y|x])^2] \\ &= \mathbb{E}_{\hat{y},y}[(\hat{y} - \mathbb{E}_{\hat{y}}[\hat{y}])^2] + (\mathbb{E}_{\hat{y}}[\hat{y}] - \mathbb{E}[y|x])^2\end{aligned} \tag{3.19}$$

Here, $\mathbb{E}_{\hat{y}}[\hat{y}]$ represents the expected model prediction conditioned on the data $\mathcal{D}$ and input x. Combining the results of our analysis, we arrive at the total expected error as:

$$\mathbb{E}_y[(y - \mathbb{E}[y|x])^2] + (\mathbb{E}_{\hat{y}}[\hat{y} - \mathbb{E}[y|x]])^2 + \mathbb{E}_{\hat{y}}[(\hat{y} - \mathbb{E}_{\hat{y}}[\hat{y}])^2] \tag{3.20}$$

In this equation, the first term signifies the variance of the true label, which remains constant for a given x. The second term captures the bias of the model, reflecting how much the average model prediction deviates from the expected true label. The third term quantifies the model's uncertainty concerning the selected input x.

Referring to (Cohn, Ghahramani, and Jordan 1996), we can denote the uncertainty term as:

$$\sigma^2_{\hat{y}}(x|\mathcal{D}) = \mathbb{E}_{\hat{y}}[(\hat{y} - \mathbb{E}_{\hat{y}}[\hat{y}])^2] \tag{3.21}$$

This term explicitly represents the variance of the model predictions at the input x given the dataset $\mathcal{D}$. More explicitly, it can be expressed as:

$$\sigma^2_{\hat{y}}(x|\mathcal{D}) = \mathbb{E}_{\hat{y}\sim p(\hat{y}|\mathcal{D};x)}[(\hat{y} - \mathbb{E}_{\hat{y}\sim p(\hat{y}|\mathcal{D};x)}[\hat{y}])^2] \tag{3.22}$$

This formulation emphasizes the variability of the model's predictions around their mean, providing insights into the model's reliability in its estimations. The active learning by variance reduction algorithm can be summarized as follows:

1. **Sampling Candidates**: Sample candidate points $\tilde{x}_1, \ldots, \tilde{x}_m$ from $p(x)$.
2. **Compute Expected Variance Reduction**: For each candidate $\tilde{x}_i$, compute:

$$\mathbb{E}_{p(x)}[\sigma^2_{\hat{y}}(x|\tilde{\mathcal{D}})] \tag{3.23}$$

3. **Select the Best Candidate**: Choose the point that minimizes expected variance reduction:

$$\tilde{x}^* = \arg\min_{\tilde{x}_i} \mathbb{E}_{p(x)}[\sigma^2_{\hat{y}}(x|\tilde{\mathcal{D}})] \tag{3.24}$$

4. **Update Model**: Incorporate the newly labeled data and repeat the process.

While there is no general recipe for the number of iterations to perform, one could imagine relying on some empirical measure like a loss on left-out labelled data to gauge model improvement (as seen in Figure 3.4, Figure 3.5). Intuitively, the size of the data set and its relationship to the loss is intimately tied to the model complexity which impacts its data-thirstiness.

We note to the reader that $P(X = x)$ is a distribution with potentially-infinite support and the authors do not compute this integral exactly. Instead, the computational estimate of that integral consists of sampling several points $x \sim P(X = x)$ and averaging the quantity inside the integral over these points until convergence using Monte-Carlo sampling approaches (see (Ghojogh et al. 2020)).

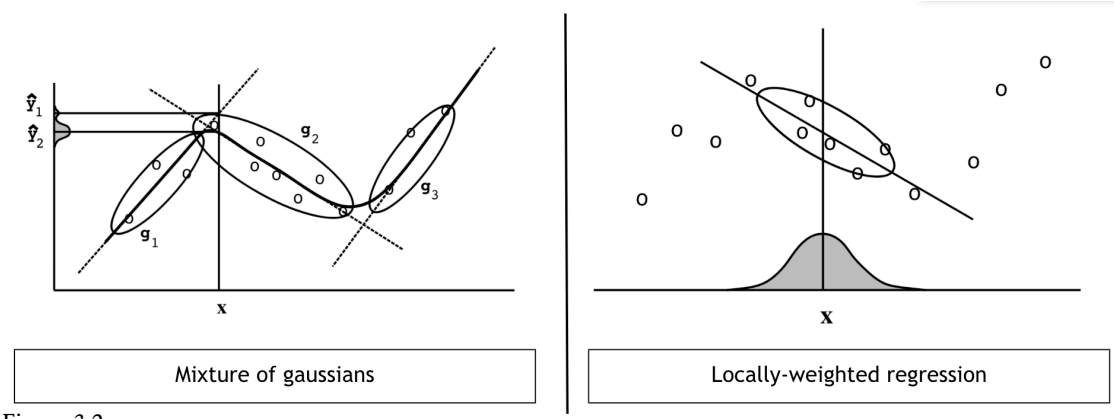


Figure 3.2
Two models were empirically explored. These two models lead to closed-form, accurately and efficiently-computed expected learner variance which can be plugged into the algorithm.

Arm2D (Figure 3.3) is a kinematics problem where learner has to predict the tip position of a robotic arm given a set of joint angles $_1$, $_2$. In this analysis, the two models seen in Figure 3.2, namely the Gaussian mixture model and locally-weighted regression (LOESS).

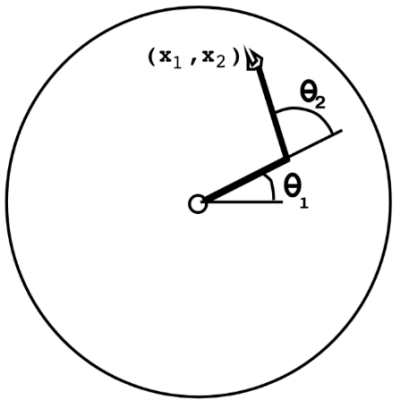


Figure 3.3
The arm kinematics problem. The learner attempts to predict tip position given a set of joint angles $_1$, $_2$

The results shown in Figure 3.4, Figure 3.5 are intriguing. As expected, the variance of the learner decreases because the authors selected points to minimize expected variance. Addition-

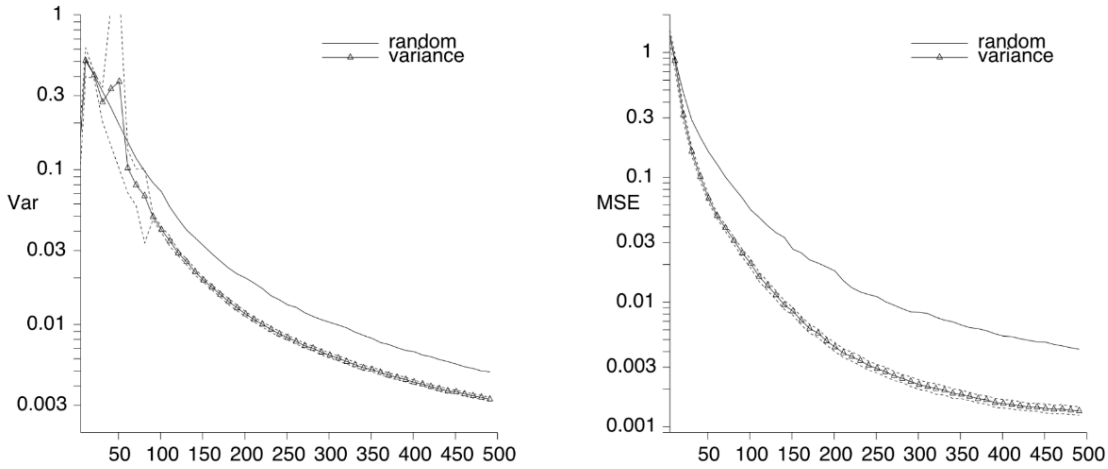


Figure 3.4
Arm2D domain. Dotted lines denote standard error for average of 10 runs, each started with one initial random example.

ally, we observe a related decrease in the mean square error (MSE) of both models as the dataset size increases. This is a notable outcome because the expected learner variance for these models can be computed accurately and efficiently relative to a new point. When integrated into the general active learning loop (Figure 3.1), this significantly enhances model performance.

In the case of the locally-weighted regression model (Figure 3.5), it is surprising that if points were chosen randomly, the MSE would be highly unstable, with sharp fluctuations. However, when active learning by variance reduction is applied, using expected learner variance as a proxy, the MSE decreases almost smoothly, aside from some initial instabilities.

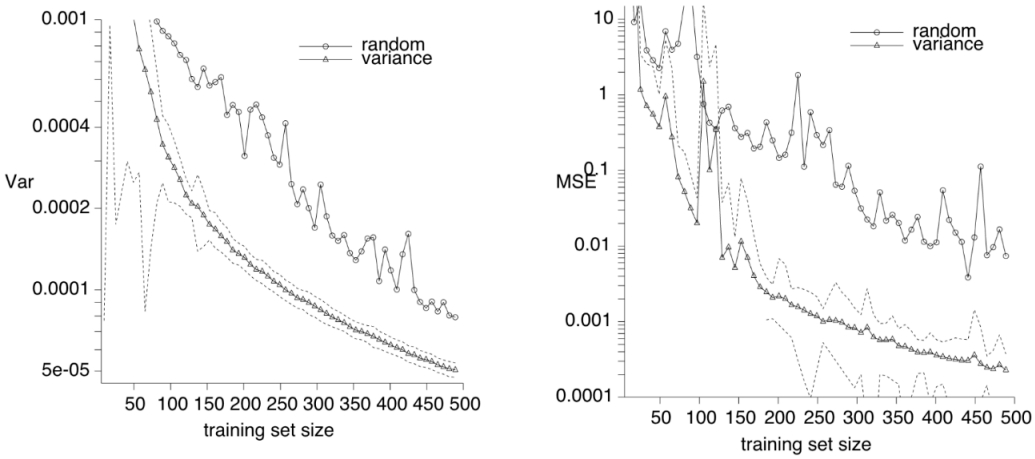


Figure 3.5
Variance and MSE learning curves for LOESS model trained on the Arm2D domain. Dotted lines denote standard error for average of 60 runs, each started with a single initial random example.

3.1.6 Active Learning in Ranking and Comparison

Many researchers have shown that making comparisons is easier and more convenient for users than assigning a specific score to each item. Individual comparisons yield a complete ranking over a set of n objects $\Theta = (\theta_1, \theta_2, \cdots, \theta_n)$. This ranking is defined as a mapping $\sigma : \{1, \cdots, n\} \rightarrow \{1, \cdots, n\}$ that orders the set of objects Θ. Specifically, for a single σ, $\sigma(\Theta) = \theta_{\sigma(1)} < \theta_{\sigma(2)} < \cdots < \theta_{\sigma(n-1)} < \theta_{\sigma(n)}$, where $\theta_i < \theta_j$ means that θ_i is rated lower than θ_j.

For any n elements to be ranked, there are $n!$ possible orderings that can result in the correct complete ranking. Given that a lower bound on sorting is $n \log n$, obtaining a guaranteed true rating over n objects requires $n \log n$ pairwise comparisons if those comparisons are chosen at random. This number can be quite high and costly in many applications, especially since most ranking information comes from humans. The more comparisons they have to make, the more money and time is spent. This process can also be inefficient, as some comparisons provide more value to the learning process than others, making some comparisons a waste. This inefficiency can be detrimental in fields like psychology and market research, where comparisons are heavily utilized, and a faster process could offer significant benefits.

The reason the lower bound on the number of comparisons is $n \log n$ is that it assumes no prior information about the underlying space and field, so comparisons are chosen at random. However, leveraging the structures within the comparison space can provide more information about which comparisons are most valuable. For example, (G. and Nowak 2011) discusses how eye doctors have a wide range of options when assigning prescriptions for glasses, yet patients do not see them making many comparisons before deciding on the best option. This is because eye doctors incorporate domain knowledge into the process and only ask clients for comparisons when necessary. Applying similar knowledge in the ranking field leads to an active learning approach that selects data based on the relevance of a comparison query toward finding the final $\sigma(\Theta)$.

3.1.6.1 Geometric Approach to Comparisons

In this section, we will review the paper (G. and Nowak 2011), which explores active learning within data that can be embedded in a multi-dimensional space. In this context, comparisons between two different objects divide the space into halves, with one object being superior in each half. By leveraging such spatial information, the paper develops a geometric approach to ranking and active learning. This spatial information serves as the domain knowledge that informs which comparisons to perform to achieve the ranking.

For this application, the following terms are defined:

1. R^d: The space in which objects can be embedded.
2. $\theta_1, ..., \theta_n$: The objects, now representing their locations in R^d.

3. For each ranking σ, there is a reference point $r_\sigma \in R^d$, such that if, according to ranking σ, $\theta_i < \theta_j$ (object i is worse than j), then $||\theta_i - r_\sigma|| < ||\theta_j - r_\sigma||$. In other words, object i is closer to the reference point r_σ of the ranking than object j.
4. $\Sigma_{n,d}$: The set of all possible rankings of the n objects that satisfy the embedding distances in the space R^d as defined above. Note that not all possible rankings will satisfy the embedding conditions, but multiple rankings might satisfy all those conditions.
5. For every ranking σ, there is $M_n(\sigma)$, the number of pairwise comparisons needed to identify the ranking. When comparisons are done at random, $E[M_n(\sigma)] = n \log n$. The paper (G. and Nowak 2011) examines this quantity to demonstrate that it can be reduced by incorporating spatial knowledge.
6. $q_{i,j}$: The query of comparison between objects i and j.

3.1.6.2 Embedding Space

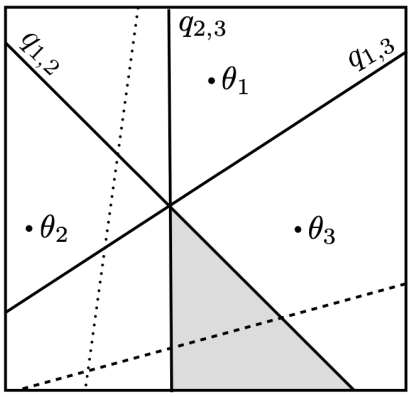


Figure 3.6
Objects $\theta_1, \theta_2, \theta_3$ and queries in R^2. The r_θ lies in the shaded region which represents $\Sigma_{n,2}$(consistent with the labels of $q_{1,2}, q_{1,3}, q_{2,3}$). The dotted (dashed) lines represent new queries whose labels are (are not) ambiguous.

To properly understand how to select the most valuable queries, it is essential to examine the space where the objects exist and how the queries divide that space to determine the proper rankings. For this example, in Figure 3.6, the paper (G. and Nowak 2011) operates in R^2 space with three objects: θ_1, θ_2, and θ_3. There are pairwise queries $q_{1,3}$, $q_{2,3}$, and $q_{1,2}$ between them, denoted by solid lines equidistant from the two objects they compare. These lines split the R^2 space into halves, with each half closer to one of the two objects. The paper colors the side of the worse object for each query in dark grey and takes the intersection of these halves, resulting in the dark grey region in the image. This region indicates $\Sigma_{n,2}$ since all points follow the embedding conditions. Specifically, for every point r in the dark grey area, $||\theta_3 - r|| < ||\theta_2 - r|| < ||\theta_1 - r||$, meaning $\theta_3 < \theta_2 < \theta_1$. Thus, every point r is one of the r_σ representing their respective rankings $\sigma \in \Sigma_{n,2}$. In other words, the paper aims to have the reference points and dark grey region closest to the worst object and furthest from the best object.

The authors also denote the label for each query $q_{i,j}$, such as label $y_{i,j} = 1\{q_{i,j}\}$ (for example, $y_{1,2} = 0, y_{3,2} = 1$). This allows for deciding how to label new queries represented by dashed and dotted lines, depending on which objects each query compares. Focusing on the dotted line, called $q_{i,4}$, where $i = 1, 2, 3$, and considering potential locations of θ_4, the line must be equidistant from one of the three objects in the picture and θ_4, meaning θ_4 can be placed in three different locations. If the query performed is $q_{2,4}$, then θ_4 will be closer to the dark grey area than θ_2, thus $y_{2,4} = 0$. However, if $q_{1,4}$ or $q_{3,4}$ are performed, θ_4 will be further from the dark grey area than θ_1 or θ_3, meaning $y_{1,4} = y_{3,4} = 1$. In this case, the labels are contradictory and depend on which object they are compared with, making such a query $q_{i,4}$ ambiguous.

In contrast, the authors analyze the dashed line, called $q_{i,5}$, where $i = 1, 2, 3$, and consider potential locations of θ_5. Since the line must be equidistant from one of the three objects in the picture and θ_5, it can be placed in three different locations. If one of the three potential queries is performed, θ_5 will be closer to the dark grey area than θ_1, θ_2, and θ_3, meaning $y_{1,5} = y_{2,5} = y_{3,5} = 0$. In this case, all labels are the same regardless of which object is used, meaning such a query will not be contradictory, as all agree on the label.

The goal is to perform as many ambiguous queries as possible and skip non-ambiguous queries to decrease the total $M_n(\sigma)$. Intuitively, if there is contradictory information about a query, it needs to be erformed so that a human can clarify its direction. Conversely, if all sources of information from the domain space agree on the query's label, that information can be used without asking a human, incorporating the knowledge of the embedding distances.

Lastly, to consider the general case of the R^d space, rather than discussing halves of the image, it is essential to discuss half-spaces. Similarly, consider the half-space that assigns a label of 1 to the query and the half-space assigning a label of 0. If both half-spaces exist, they have conflicting information on the query, making the query ambiguous. However, if one of the half-spaces does not exist, it means the other is the full space, representing consistency in the label assignment and a non-ambiguous query.

3.1.6.2.1 Algorithms for Ambiguous Query Selection

Algorithm 3.1 Query Selection Algorithm

input: n objects in $\mathbb{R}^d$
initialize: objects $\theta_1, \ldots, \theta_n$ in uniformly random order
for $j = 2, \ldots, n$ **do**
    for $i = 1, \ldots, j-1$ **do**
        if $q_{i,j}$ is ambiguous **then**
            request $q_{i,j}$'s label from reference
        else
            impute $q_{i,j}$'s label from previously labeled queries
        end if
    end for
end for
output: ranking of n objects

The standard algorithm in Algorithm 3.1 requests labels for $q_{i,j}$ if those queries are ambiguous; otherwise, it infers the information from prior comparisons and their labels.

It is important to demonstrate that the number of comparisons decreases. Specifically, (G. and Nowak 2011) shows that this algorithm has $E[M_n(\sigma)] = O(d \log n)$, where d is the dimension of the space and $d < n$, which improves on the $O(n \log n)$ baseline. The proof can be studied in detail in the paper itself, but at a high level, it starts by reasoning about the probability of a query being ambiguous and a comparison being requested from a human, thus representing $M_n = \Sigma_{k=1}^{n-1} \Sigma_{i=1}^{k} 1\{Requestq_{i,k+1}\}$. For that, the authors define $Q(i, j)$, which represents the number of different rankings that exist for i elements in j-dimensional space (e.g., $Q(1, d) = 1, Q(n, 0) = 1, Q(n, 1) = n!$). In that case, $|\Sigma_{n,d}| = Q(n, d)$. Further, using recurrence relations for $Q(i, j)$, the authors derive that $|\Sigma_{n,d}| = Q(n, d) = O(n^{2d})$, which is omitted here. Analogously, the authors define $P(i, j)$, which represents the number of rankings in $\Sigma_{n,d}$ that will still be possible with the addition of a new element $i + 1$ to the ranking objects. Referring back to Figure 3.6, $P(i, j)$ estimates how much of the dark grey area will still exist after making a query for $i + 1$. As indicated there, the dotted line ambiguous query did not change the dark grey a rea at all ($P(n, d) = Q(n, d)$), whereas the dashed non-ambiguous query would cut a piece from it ($P(n, d) < Q(n, d)$). Thus, $Requestq_{i,k+1} = P(k, d)/Q(k, d)$, so a higher value indicates more possible rankings and an ambiguous query that needs to be requested to obtain more useful information. With this in mind, the authors derive that $E[M_n(\sigma)] = O(d \log n)$, showing that fewer queries are needed for effective ranking.

The issue with this algorithm is that only one human provides the answers to the requested queries, which means it does not account for their biases. An alternative approach is a Robust Query Selection Algorithm (RQSA) (G. and Nowak 2011), which uses majority voting for every query to indicate the ground truth of the query's label. However, the authors consider that a group of people can still give incorrect or divided responses. If the votes for each answer are almost equal in number, the authors push that query to the end of the algorithm to see if it can become a non-ambiguous query with more information learned. If it does not, an odd number of voters is used to determine the final ranking.

3.1.6.2.2 Performance Analysis

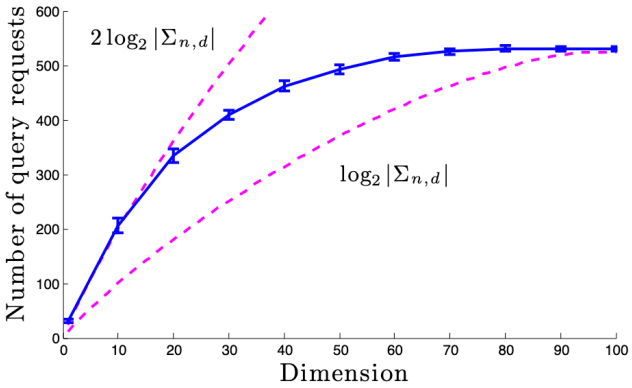


Figure 3.7
Mean and standard deviation of requested queries (solid) in the noiseless case for $n = 100$; $\log_2|\Sigma_{n,d}|$ is a lower bound (dashed).

Table 3.2
Statistics for the Robust Query Selection Algorithm (RQSA) (G. and Nowak 2011) discussed at the end of Section 3.1.6.2.2 and the baseline of conducting all comparisons. y serves as a noisy ground truth, $\tilde{y}$ is the result of all comparisons, and $\hat{y}$ is the output of ~~the RQSA.~~

Dimension		2	3
% of queries	mean	14.5	18.5
	std	5.3	6
Average error	$d(\bar{y}, y)$	0.23	0.21
	$d(\bar{y}, y)$	0.31	0.29

Figure 3.7 shows that the number of comparisons fits within the expected bounds, as $\log|\Sigma_{n,d}| = \log(n^d) = d\log n$. To derive that graph, authors (G. and Nowak 2011) sampled 100 random data points in a R^d space, where d took on 10 different values as indicated on the graph. Each dimension's experiments were repeated 25 times for consistency.

With regard to the accuracy and performance of the method, the authors did a ranking experiment on 100 different audio signals, results of which can be seen in Table 3.2. The ground truth labels came from humans, indicated by y in the table. That resulted in the existence of noise and potential errors in the ground truth, which could influence the performance of both the baseline algorithm that does all comparisons ($\tilde{y}$) and the Robust Query Selection Algorithm (RQSA) proposed in Section 3.1.6.2.2 ($\hat{y}$). As can be seen in both 2 and 3-dimensional spaces RQSA performed worse by 8% compared to the baseline, which indicates that active learning that uses the domain information can still be erroneous due to the inference of certain comparisons that sometimes may not be entirely correct. However, as can be seen by the upper part of Table 3.2, significantly less queries were requested compared to the baseline, which means that the approach can have a significant benefit at a cost of slight loss in accuracy.

3.1.6.3 User Information as Domain Knowledge for Active Learning

An alternative source of domain knowledge could be users themselves, who can indicate their uncertainty when it comes to comparing two objects. Prior studies have shown (Amershi et al. 2014) that when presented with only two options when selecting which object is better, but not being able to properly decide, users would get frustrated and tend to respond more faultyly, creating noise and incorrect responses in the data. Through feedback and other studies (Guillory and Bilmes 2011) it was determined that presenting users with an option of indifference between the two objects can remove those problems. Moreover, in connection to active learning, the authors show that such an option helps to select more informative queries since it provides more domain knowledge that can be used, resulting in a decrease in the number of queries required.

For this problem, the following terms are defined:

1. c - a cost function that represents user preferences, and the result the model has to determine at the end of training. The preferred items will have lower costs, and less preferred ones will have higher costs. The goal is to determine this function with the fewest possible number of queries using active learning.
2. H - a set of hypotheses over the possible cost functions, where for each $h \in H$ there is a cost function c_h associated with it.
3. h^* - a true hypothesis that the model needs to determine, which has cost c_{h^*} associated with it
4. $t(x, y)$ - a test performed to compare items x and y (the user is being asked to provide a response to which item is better). Those tests result in changes and adjustments to H as more information is learned.
5. $o(x, y)$ - observation or result of $t(x, y)$, where $o(x, y) \in \{x < y, x > y\}$
6. $S = \{(t_1, o_1), (t_2, o_2), ..., (t_m, o_m)\}$ - a sequence of m pairs of tests and observations
7. $w(H|S)$ - probability mass of all hypotheses that are still consistent with the observations (similar to the dark grey area from Figure 3.6 and $Q(i, j)$ discussed in Section 3.1.6.2.2. This means that if $h \in H$ is inconsistent with user responses received, it is removed from H.

With the key terms defined, let's consider the noiseless base setting where users only have two options for response. Those components will also later be translated to the setting with the third option so the true cost function can be determined there. $w(H|S)$ is the sum of the weights of all hypotheses that are still consistent with the evidence.

$$w(H|S) = \sum_{h \in H} w(h|S) \tag{3.25}$$

Each $w(h|S)$ is a probability of the evidence's existence given such hypothesis:

$$w(h|S) = p(S|h) \tag{3.26}$$

Such probability comes from the test-observation pairs since they compose the set S. Moreover, each test is independent of other tests, which gives:

$$p(S|h) = \prod_{(t,o)\in S} p((t,o)|h) \tag{3.27}$$

In the noiseless setting, users will select an option that minimizes their cost function (selecting more preferred items), mathematically defined as:

$$p((t, o = x)|h) = \begin{cases} 1 & c_h(x) < c_h(y) \\ 0 & else \end{cases} \tag{3.28}$$

6.3.3.1 User Noise Modeling

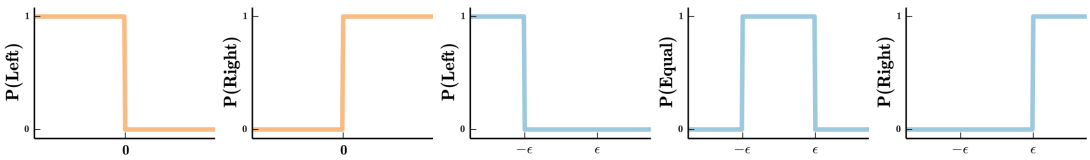
As has been discussed, users are not perfect evaluators and even get frustrated if unable to select the better option. Prior work (Amershi et al. 2014) has shown that treating users as perfect can lead to poor performance. That gave rise to accounting for noise in users' responses, but a majority of such work applies the same noise to all queries and all responses. While those led to great performance results (Guillory and Bilmes 2011), they don't accurately reflect the real world, which gave rise to the idea of creating query-based noise.

Effectively, for some of the queries it is important to incorporate the fact that the user is unsure and noisy, but for others, if the user is confident, noise in the response is not needed at all. For comparison-based learning, this means that the noise is related to the costs of the two items compared. Specifically for items x and y, if $c_{h^*}(x) \simeq c_{h^*}(y)$ then the items are hard to distinguish for the user, so here it is preferred to incorporate user uncertainty and noise. But if $c_{h^*}(x) >> c_{h^*}(y)$, the user will certainly select y and the other way around, which is where the noise is not needed.

Query-dependent noise is also supported in the psychology literature, which means that such an approach is more related to the real world. In particular, psychologists talk about the Luce-Sheppard Choice rule (Shepard 1957) when talking about comparisons. This rule previously gave rise to a logistic model based on the noise (Viappiani and Boutilier 2010) where the probability of observation for a given test is:

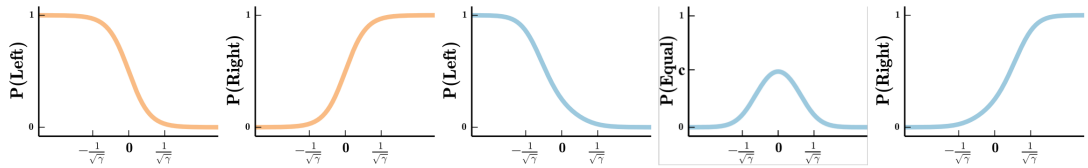
$$p((t, o = x)|h) \propto exp(-\gamma * c_h(x)) \tag{3.29}$$

Figure 3.8, Figure 3.9 demonstrate the difference between the noiseless setting and incorporating the Luce-Sheppard Choice rule. GBS is the baseline model with only 2 response options, and CLAUS is the model with the uncertainty option added. The figures show how incorporating such noise influences and smoothes the probability distribution of the user's response.



(a) GBS (Left) (b) GBS (Right) (c) CLAUS (Left) (d) CLAUS (Equal) (e) CLAUS (Right)

Figure 3.8
User response model in the noiseless setting



(a) GBS (Left) (b) GBS (Right) (c) CLAUS (Left) (d) CLAUS (Equal) (e) CLAUS (Right)

Figure 3.9
User response with Luce Sheppard noise model

6.3.3.2 User Uncertainty

We will now discuss the functionality of CLAUS, which is an algorithm designed by (Holladay et al. 2016) that allows users to select an uncertain response about the two options that they need to rank. The authors model such uncertainty as ϵ and it is associated with each c_h, so now every hypothesis h is defined over a pair of (c_h, ϵ_h). It is important to note that the goal is to still learn and maintain our objective on c, ϵ is only necessary to model the users' responses. The uncertainty relates to the cost function in the following way:

$$|c_h(x) - c_h(y)| < \epsilon_h \tag{3.30}$$

this means that the user is uncertain between items x and y and their cost difference is negligible such that the user is not able to select which item is better. This in turn gives more information about the real value of the two items, as a binary response would indicate the user's preference towards one item, which will not be real and will skew the cost functions.

This causes modifications of the problem set-up:

1. For test $t(x, y)$ the observation will be $o(x, y) \in \{x < y, x > y, \tilde{xy}\}$, where $\tilde{xy}$ is the uncertain response.
2. The probability distribution over the user's response (Equation 3.28) will now be defined as:

$$p((t, o = x)|h) = \begin{cases} 1 & c_h(x) < c_h(y) - \epsilon_h \\ 0 & else \end{cases} \tag{3.31}$$

$$p((t, o = \tilde{xy})|h) = \begin{cases} 1 & |c_h(x) - c_h(y)|^2 < \epsilon_h^2 \\ 0 & else \end{cases} \tag{3.32}$$

This means the user confidently selects x when it is better than y by more than ϵ, but if the squared difference of the cost functions of two items is negligible by ϵ user will choose the indifferent option.

3. Finally this also updates the noise model (Equation 3.29):

$$p((t, o = x)|h) \propto \exp(-\gamma * [c_h(x) - c_h(y)]) \tag{3.33}$$

$$p((t, o = \tilde{xy})|h) \propto exp(-1/\epsilon_h^2 * [c_h(x) - c_h(y)]^2) \tag{3.34}$$

6.3.3.3 Performance Analysis

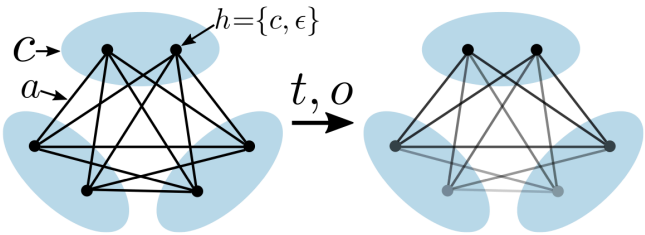


Figure 3.10
CLAUS using equivalence classes. Each cost function c corresponds to an equivalence class (blue ellipse). Hypotheses (black dots) are $\{c_h, \epsilon_h\}$ pairs. Hypotheses sharing a cost c are said to be inside the equivalence class of c. After performing a test and receiving an observation, the evidence results in downweighting connections among some of the hypotheses.

Before diving deeper into the comparisons of performance, it is important to indicate that rather than predicting a specific pair (c_h, ϵ_h), the algorithm focuses on predicting a group of pairs that are similar to one another, otherwise called equivalence class (Figure 3.10), which indicates not essentially different hypothesis for the cost function and uncertainty. That information is learned through each new test, as the algorithm updates the information about c and ϵ that distinguishes between the distinct h, finding the equivalence groups among them. Moreover, the authors tweaked the parameter responsible for the size of the equivalence class (how many hypotheses can be grouped together at a time).

Table 3.3
Performance of GBS and CLAUS with different labels for the uncertainty

Category	Accuracy	Query Count
GBS - About Equal	94.15 ± 0.52	36.02 ± 0.03
GBS - Not Sure	$\mathbf{94.66 \pm 0.55}$	35.95 ± 0.04
CLAUS - About Equal	91.56 ± 0.84	$\mathbf{25.93 \pm 0.41}$
CLAUS - Not Sure	90.86 ± 0.74	26.98 ± 0.47

The first performance evaluation is done on the number of queries and confirms that it decreases in Figure 3.11. The GBS model serves as the baseline, as it will do all of the comparison queries using the binary response options. The CLAUS model is measured over different

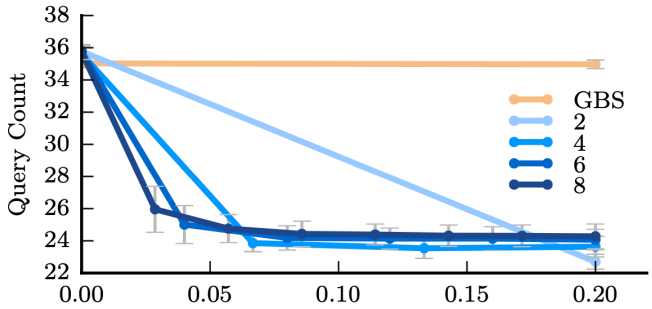


Figure 3.11
Performance of GBS and its variants

values of ϵ on the x-axis and over different sizes of the equivalence sets indicated by different shades of blue. Figure shows that all variants of CLAUS use approximately 10 fewer queries on average compared to GBS. Moreover, using bigger-sized equivalence classes can further decrease the number of needed queries. The most optimal $\epsilon \simeq 0.07$, after which higher ϵ does not provide any benefit.

Lastly, the authors considered the performance difference, which is indicated in Table 3.3. For that authors used two different labels for the uncertainty button in CLAUS, it was either labeled as "About Equal" or "Not Sure" as those can provoke different responses and feelings in users. Moreover, GBS and CLAUS-type responses were mixed in the same set of questions to the user, which splits the metrics for both in two as can be seen in Table 3.3. The performance of CLAUS is lower by 3% on average, indicating similar results to Section 3.1.6.3, showing that a smaller number of queries can still lead to a performance loss. However, the second column of Table 3.3 supports the information in Figure 3.11, as it also shows that 10 fewer queries were conducted on average.

3.1.7 Active Preference-Based Learning of Reward Functions

Active learning can be essential in learning within dynamic systems and environments. Say we have an agent in an environment, and we want it to conform to a certain behavior as set by a human. How exactly do we go about doing this? In a traditional RL setting, this is solved by a class of algorithms under Inverse Reinforcement Learning. Techniques such as VICE and GAIL attempt to learn a reward function that can distinguish between states visited by the agent and states desired to be visited as defined by a human. In effect, a human will demonstrate what it would like the agent to do in the environment, and from there, learning is done. However, what if humans do not precisely know how an agent should optimally behave in an environment but still have some opinion on what trajectories would be better than others? This is where a paper like Active Preference-Based Learning of Reward Functions comes into the picture. The paper aims to use human preferences to aid an agent's learning within a dynamic system.

A dynamic system contains human input, robotic input, and an environment state. The transitions between states is defined by f_{HR}, so that we have:

$$x^{t+1} = f_{HR}(x^t, u_R, u_H) \tag{3.35}$$

At a given time step t, we have x_t, u_R^t, and u_H^t. This can be encapsulated into a single d dimensional feature vector that the authors denote as ϕ. The paper then assumes that the underlying reward model we are trying to learn can be represented linearly. If we have our human reward preference function defined as r_H, this means we can write r_H as:

$$r_H(x^t, u_R^t, u_H^t) = w^\top \phi(x^t, u_R^t, u_H^t) \tag{3.36}$$

Because the reward function is linear, we can take the weight vector out of the summation if we want to calculate the reward over an entire trajectory:

$$\begin{aligned} R_H(x^0, u_R, u_H) &= \sum_{t=0}^{N} r_H(x^t, u^t, u_H^t) \\ \Phi &= \sum \phi(x^t, u_R^t, u_H^t) \\ R_H(traj) &= w \cdot \Phi(traj) \end{aligned} \tag{3.37}$$

3.1.7.1 Properties of W

First, the scale of w does not matter because we only care about the relative rewards produced with w (given two different trajectories, we want to answer the question of which trajectory a human would prefer, i.e. which one has a higher preference reward). This means we can constrain $||w|| <= 1$, so the initial prior is uniform over a unit ball. From here, we can determine a probabilistic expression to assess whether we should prefer trajectory A or B (because it can be noisy with human input). Let $I_t = +1$ if the human prefers trajectory A and let $I_t = -1$ if the human prefers trajectory B. We get the following for $p(I_t|w)$.

$$\begin{aligned} p(I_t = +1|w) &= \frac{exp(R_H(traj_A))}{exp(R_H(traj_A)) + exp(R_H(traj_B))} \\ p(I_t = -1|w) &= \frac{exp(R_H(traj_B))}{exp(R_H(traj_A)) + exp(R_H(traj_B))} \end{aligned} \tag{3.38}$$

We can re-write this expression to make it cleaner, using the following substitution:

$$\psi = \Phi(traj_a) - \Phi(traj_b) \tag{3.39}$$

$$f_\psi(w) = p(I_t|w) = \frac{1}{1 + exp(-I_t w^\top \psi)} \tag{3.40}$$

The idea now is that we can update $p(w)$ everytime we get a result from a human preference query using Bayes:

$$p(w|I_t) < -p(w) \cdot p(I_t|w) \tag{3.41}$$

We do not need to know $p(I_t)$ because we can use an algorithm like the Metropolis algorithm to actually sample.

3.1.7.2 Generating Queries

This is where the interesting part of the paper comes into play. How do we actually generate queries for the user to pick between? This paper synthetically generates queries through an optimization process and then presents them to a human to pick between. The idea is that we want to generate a query that maximizes the conditional entropy $H(I|w)$. There are a few ways to think about this – intuitively we want to pick a query that we are most uncertain about given our current weights (thus having the highest conditional entropy given the weights). The way the authors of the paper frame this originally in the paper is that "we want to find the next query such that it will help us remove as much volume (the integral of the unnormalized pdf over w) as possible from the space of possible rewards." Mathematically this can be written as:

$$max_{x^0,u_R,u_H^A,u_H^B} min\{E[1 - f_\psi(w)], E[1 - f_{-\psi}(w)]\} \tag{3.42}$$

But how exactly do we optimize this expression mathematically? After all, we need to use this expression to generate synthetic queries. The answer is to sample $w_1, ...w_m$ from $p(w)$. We can assume we are sampling points from a point cloud, thus approximating the distribution $p(w)$ as

$$p(w) = \frac{1}{M} \sum \delta(w_i). \tag{3.43}$$

We can now approximate the expectation expression like so:

$$E[1 - f_\psi(w)] = \frac{1}{M}(\sum 1 - f_\psi(w_i)) \tag{3.44}$$

and now we can optimize the expression to generate a synthetic query! Altogether, the algorithm looks like the following:

Algorithm 3.2 Preference-Based Learning of Reward Functions

input: features ϕ, horizon N, dynamics f, $iter$
initialize: $p(w) \sim Uniform(B)$, for a unit ball B
while $t < iter$ **do**
    $W \leftarrow M$ samples from $AdaptiveMetropolis(p(w))$
    $(x^0, u_R, u_H^A, u_H^B) \leftarrow SynthExps(W, f)$
    $I_t \leftarrow QueryHuman(x^0, u_R, u_H^A, u_H^B)$
    $\varphi = \Phi(x^0, u_R, u_H^A) - \Phi(x^0, u_R, u_H^B)$
    $f_\varphi(w) = \min(1, I_t \exp(w^\top \varphi))$
    $p(w) \leftarrow p(w) \cdot f_\varphi(w)$
    $t \leftarrow t + 1$
end while
output: distribution of $w : p(w)$

3.1.7.3 Batching Queries

The algorithm itself works well, however there ends up being a bottle neck that each query needs to be synthesized before being sent to the human – one at a time. In other words, the human gives their feedback, waits for a query to be synthesized, and then gives another data point of feedback. There is no room for parallelization and so the authors proposed a second algorithm in a separate paper that allows for the batching of queries. Simply put, we change the mathematical expression to the following:

$$max_{\xi_{ib+1_A}, \xi_{ib+1_B}, \ldots, \xi_{ib+b_A}, \xi_{ib+b_B}} H(I_{ib+1}, I_{ib+2}, .., I_{ib+b} | w) \tag{3.45}$$

Naively, we could consider optimizing this in the greedy fashion. This would mean just synthetically generating b independent queries. The obvious drawback of this method would be that the queries would likely be very similar to each other. The authors propose a few other heuristics that would help guide the algorithm away from generating very similar queries. As an example, the authors propose Medioid Selection where we have to cluster B greedy vectors into $b < B$ groups and pick one vector from each group (the medioid). The authors also propose two other methods rooted in providing different queries: boundary medioids selection and successive elimination. They are best visually depicted as:

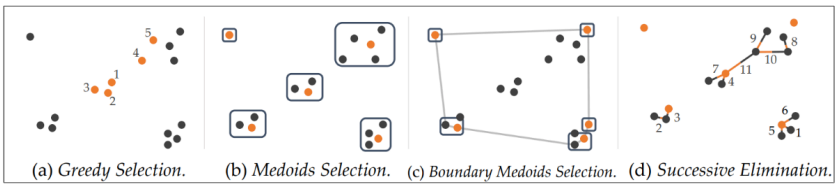


Figure 3.12
Different selection strategies

3.1.7.4 Results

The authors test both the non-batched and variety of batched learning algorithms on multiple environments:

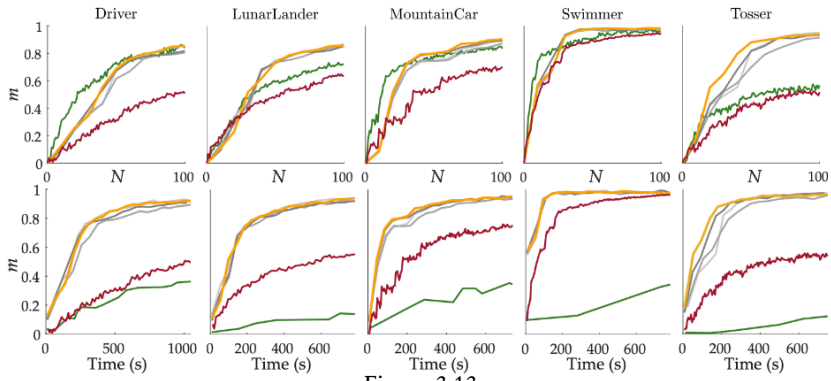


Figure 3.13
Comparison between batched and non-batched algorithms

What is interesting to note is that when graphed over N the non-batched active learning approach does in the same ball-park of performance as the batched approaches. However, if you graph it over time, we see that learning is a much slower process when not-batched.

3.1.8 Foundation Models for Robotics

Modern foundation models have been ubiquitous in discussions of powerful, general purpose AI systems that can accomplish myriad tasks across many disciplines such as programming, medicine, law, open question-answering and much more, with rapidly increasing capabilities (Bommasani et al. 2022). However, despite successes from large labs in controlled environments (Brohan et al. 2023) foundation models have not seen ubiquitous use in robotics due to shifting robot morphology, lack of data, and the sim to real gap in robotics (Walke et al. 2023). For this subsection we explore two promising approaches known as R3M and Voltron which are the first to leverage pre-training on vast amounts of data towards performance improvement on downstream robotic tasks despite the aforementioned issues (Nair et al. 2022; Karamcheti et al. 2023).

3.1.8.1 R3M: Universal Visual Representation for Robotics

R3M represents a significant advancement in the field of robotic manipulation and learning. This model diverges from traditional approaches that rely on training from scratch within the same domain on the same robot data as instead it leverags pretraining on large datasets, akin to the practices in computer vision and natural language processing (NLP) where models are trained on diverse, large-scale datasets to create reusable, general-purpose representations.

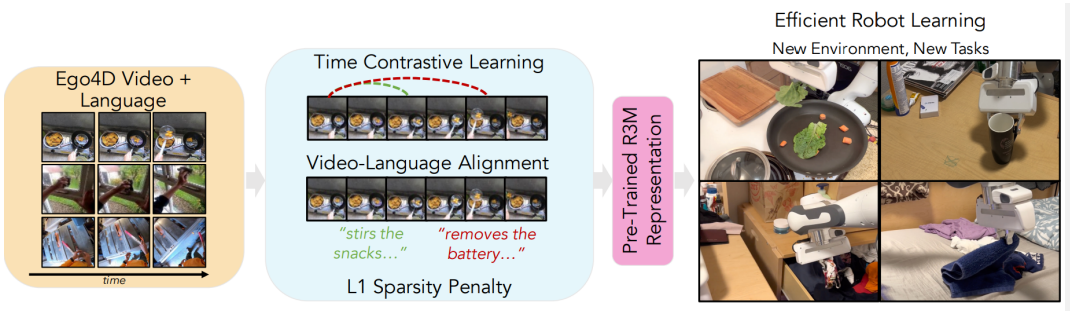


Figure 3.14
R3M pipeline

The core principle behind R3M is its training methodology. It is pre-trained on a wide array of human videos, encompassing various activities and interactions. This diverse dataset enables the model to capture a broad spectrum of physical interactions and dynamics, which are crucial for effective robotic manipulation known as EGO4D (Grauman et al. 2022). However, prior papers could not fit this dataset well, and R3M leveraged. The training utilizes a unique objective that combines time contrastive learning, video-language alignment, and a sparsity penalty. This objective ensures that R3M not only understands the temporal dynamics of scenes (i.e., how states transition over time) but also focuses on semantically relevant features, such as objects and their interrelations, while maintaining a compact and efficient representation.

What sets R3M apart in the realm of robotics is its efficiency and effectiveness in learning from a limited amount of data. The model demonstrates remarkable performance in learning tasks in the real world with minimal human supervision – typically less than 10 minutes. This is a stark contrast to traditional models that require extensive and often prohibitively large datasets for training. Furthermore, R3M's pre-trained nature allows for its application across a variety of tasks and environments without the need for retraining from scratch, making it a versatile tool in robotic manipulation. The empirical results from using R3M are compelling, leading to a 10% improvement over training from a pretrained image-net model, self-supervised approaches such as MoCo or even CLIP (Deng et al. 2009; He et al. 2020; Radford et al. 2021). Note however, that R3m does **not** use any language data which leaves quite a bit of supervision to be desired.

3.1.8.2 Voltron: Language Driven Representation Learning for Robotics

Building off the success of R3M, Voltron proposes a further extension of leveraging self-supervision and advancements in foundation models, and multi-modality. Voltron takes on an intuitive and simple dual use objective, where the trained model alternates between predicting the task in an image through natural language and classifying images based on a natural text label. This forces a nuanced understanding of both modalities (Radford et al. 2021).

Voltron's approach is distinguished by its versatility and depth of learning. It is adept at handling a wide range of robotic tasks, from low-level spatial feature recognition to high-level semantic understanding required in language-conditioned imitation and intent scoring. This

flexibility makes it suitable for various applications in robotic manipulation, from grasping objects based on descriptive language to performing complex sequences of actions in response to verbal instructions.

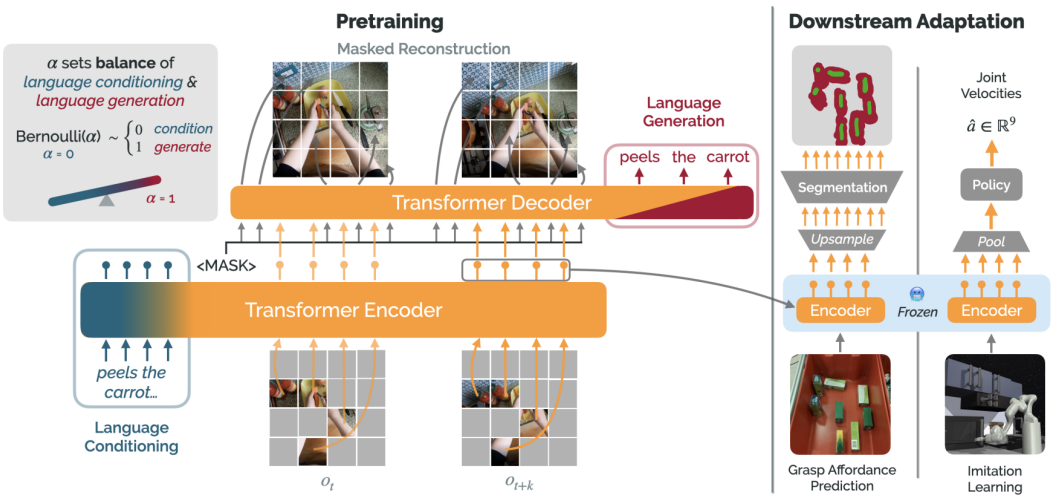


Figure 3.15
Voltron pipeline

The authors rigorously test Voltron in scenarios such as dense segmentation for grasp affordance prediction, object detection in cluttered scenes, and learning multi-task language-conditioned policies for real-world manipulation with up to 15% improvement over baselines. In each of these domains, Voltron has shown a remarkable ability to outperform existing models like MVP and R3M, showcasing its superior adaptability and learning capabilities (Xiao et al. 2022).

Moreover, Voltron's framework allows for a balance between encoding low-level and high-level features, which is critical in the context of robotics. This balance enables the model to excel in both control tasks and those requiring deeper semantic understanding, offering a comprehensive solution in the realm of robotic vision and manipulation.

Voltron stands as a groundbreaking approach in the field of robotics, offering a language-driven, versatile, and efficient approach to learning and manipulation. Its ability to seamlessly integrate visual and linguistic data makes it a potent tool in the ever-evolving landscape of robotic technology, with potential applications that extend far beyond current capabilities. Interesting the authors show Voltron does not beat R3M off the shelf but only when trained on similar amounts of data. Nevertheless, Voltron's success in diverse tasks and environments heralds a new era in robotic manipulation, where language and vision coalesce to create more intelligent, adaptable, and capable robotic systems.

3.1.9 Conclusion

On the note of applying active learning to RL and environment settings, there have been many recent papers that have attempted to extend this to more modern RL environments. For example, the paper "When to Ask for Help" (Xie et al. 2022) examines the intersection of

autonomous and active learning. Instead of just expecting an RL agent to autonomously solve a task, making the assumption that an agent could get stuck and need human input to get "unstuck" is a key insight of the paper. In general, there has been an emphasis in recent literature in robotics on not just blindly using demonstration data as a form of human input, but rather actively querying a human and using this to better synthesize correct actions.

Active learning holds promise for enhancing AI models in real-world scenarios, yet several challenges persist. This discussion aims to provide an overview of these challenges.

Task-Specific Considerations:

For certain tasks, the input space of a model may have some rare yet extremely important pockets which may never be discovered by active learning and may cause severe blindspots in the model. In medical imaging for instance, there can be rare yet critical diseases. Designing AL strategies for medical image analysis must prioritize rare classes, such as various forms of cancers. Oftentimes, collecting data around those rare classes is not a recommendation of the active learning process because these examples constitute heavy distribution drifts from the input distribution a model has seen.

Complex Task Adaptation:

AL has predominantly been adopted for simple classification tasks, leaving more other types of tasks (generative ones for instance), less explored. In Natural Language Processing, tasks like natural language inference, question-answering pose additional complexities that affect the direct application of the active learning process. While machine translation has seen AL applications, generation tasks in NLP require more thorough exploration. Challenges arise in obtaining unlabeled data, particularly for tasks with intricate inputs.

Unsupervised and Semi-Supervised Approaches:

In the presence of large datasets without sufficient labels, unsupervised and semi-supervised approaches become crucial. These methods offer a means to extract information without relying on labeled data for every data point, potentially revolutionizing fields like medical image analysis. There is an ongoing need for methods that combine self/semi-supervised learning with active learning.

Algorithm Scalability:

Scalability is a critical concern for online AL algorithms, particularly when dealing with large datasets and high-velocity data streams. The computational demands of AL can become prohibitive as data volume increases, posing challenges for practical deployment. Issues of catastrophic forgetting and model plasticity further complicate scalability, requiring careful consideration in algorithm design.

Labeling Quality Assurance:

The effectiveness of most online AL strategies hinges on the quality of labeled data. Ensuring labeling accuracy in real-world scenarios is challenging, with human annotators prone to errors, biases, and diverse interpretations. Addressing imperfections in labeling through considerations

of oracle imperfections becomes essential in real-life AL applications. Solutions for cleaning up data and verifying its quality need to be more aggressively pursued.

Data Drift Challenges:

Real-world settings introduce data drift, where distributions shift over time, challenging models to adapt for accurate predictions. These shifts can impact the quality of labeled data acquired in the AL process. For example, the criterion or proxy used for selecting informative instances may be thrown off when the distribution a model is trained on, and the distribution we want it to perform well on, are too far away from one another.

Evaluation in Real-Life Scenarios:

While AL methods are often evaluated assuming access to ground-truth labels, the real motivation for AL lies in label scarcity. Assessing the effectiveness of AL strategies becomes challenging in real-life scenarios where ground-truth labels may be limited. In other words, one may verify the goodness of an AL algorithm within the lab, but once the algorithm is deployed for improving all sorts of models on all sorts of data distributions, verifying whether AL is actually improving a model is tricky, especially when collecting and labeling data from the target distribution is expensive and defeats the purpose of using AL in the first place.

By systematically addressing these challenges, the field of active learning in AI can progress towards more effective and practical applications.

In summary, active learning is a promising modern tool to model training that presents potential benefits. As was mentioned at the start, there are numerous approaches that can be employed by active learning, starting from reducing error of model's prediction, reducing variance, to more conformal predictions. The flavor of active learning heavily depends on the applications, which include robotics, LLM, autonomous vehicles, and more. We discussed in more detail how to perform active learning for variance reduction in the case of predicting kinematics of the robotic arms, which showed decrease in MSE as well as more stable reduction in it. Next we talked about using active learning for reducing the number of comparisons required to create a ranking of objects, and the examples discussed were able to achieve that but with some loss in the prediction accuracy. Finally, we discussed how active learning can be used for modeling of reward functions within a dynamical system, which demonstrated improvements in performance and time required to achieve it. For a more hands-on experience with active learning and demonstrated example, we encourage the readers to explore a blogpost by Max Halford (Halford 2023).

3.2 Metric Elicitation

3.2.1 Introduction to Performance Metric Elicitation

In binary classification problems, one important consideration is the selection of an appropriate performance metric corresponding to the real-world task at hand. The problem of *metric elicitation* attempts to characterize and discover the performance metric of a practitioner, reflecting the rewards or costs that come with correct or incorrect classification. For example, in medical

contexts such as diagnosing a disease or choosing whether a given treatment is appropriate for a certain condition, tradeoffs are made for incorrect decisions; for example, not administering a treatment could lead to the worsening of a disease (a false negative), whereas delivering the wrong kind of treatment could lead to adverse side effects that would be worse than not treating the condition (a false positive).

Rather than choosing from a limited set of default choices like the F1-score or weighted accuracy, metric elicitation considers the process by which we can devise a metric that best matches the preferences of the practitioners or users, by querying an "oracle" who provides feedback on proposed potential metrics in the form of pairwise comparisons. Because queries to humans are often expensive, the aim is to minimize the amount of comparisons needed.

Note: Almost all of the content in this chapter comes from "Performance Metric Elicitation from Pairwise Classifier Comparisons" by Hiranandani et al. (Hiranandani et al. 2019a), which introduced the problem of metric elicitation and the framework for binary-class metric elicitation from pairwise comparisons. In this chapter, we aim to present their work expository while providing additional motivation and intuitive explanations to supplement their work.

The motivation for the pairwise comparison portion of metric elicitation comes from a rich history of literature in psychology, economics, and computer science (Samuelson 1938; Mas-Colell 1977; Varian 2006; Braziunas and Boutilier 2012; Tamburrelli and Margara 2014) demonstrating that humans are often ineffective at providing absolute feedback on things like potential prices, user interfaces, or even ML model outputs (hence the comparison-based structure of RLHF, for instance). In addition, confusion matrices are a way to accurately capture binary metrics such as accuracy, F_β, and Jaccard similarity by recording the number of false positives, true positives, false negatives, and true negatives obtained by a certain classifier. The main goal of this chapter is to introduce two binary-search procedures that can be used to approximate the oracle's performance metric for two types of metrics (linear and linear-fractional performance metrics) by presenting the oracle with confusion matrices generated by various classifiers; in essence, we are learning an optimal threshold for classification given a decision boundary for a binary classification problem.

First, we introduce some relevant notation that will later be used to formalize notions of oracle queries, classifiers, and metrics.

- $X \in \mathcal{X}$ is an input random variable.
- $Y \in \{0, 1\}$ is the output random variable.
- We learn from a dataset of size n, denoted by $\{(x, y)_i\}_{i=1}^n$, generated iid from some distribution $\mathbb{P}(X, Y)$.
- $\eta(\vec{x}) = \mathbb{P}(Y = 1 | X = x)$ is the conditional probability of the positive class, given some sample x.
- $\zeta = \mathbb{P}(Y = 1)$ is the unconditional probability of the positive class
- The set of all potential classifiers is $\mathcal{H} = \{h : \mathcal{X} \to \{0, 1\}\}$

- The confusion matrix for some classifier h is $C(h, \mathbb{P}) \in \mathbb{R}^{2\times 2}$, where $C_{ij}(h, \mathbb{P}) = \mathbb{P}(Y = i, h = j)$ for $i, j \in \{0, 1\}$; these represent the false positives, true positives, false negatives, and true negatives, so $\sum_{i,j} C_{ij} = 1$.
- $\mathcal{C}$ is the set of all confusion matrices
- Note: Since $FN(h, \mathbb{P}) = \zeta - TP(h, \mathbb{P})$ and $FP(h, \mathbb{P}) = 1 - \zeta - TN(h, \mathbb{P})$; $\mathcal{C}$ is in fact a 2-dimensional space, not a 4-dimensional space
- Any hyperplane (line) in (tp, tn) given by $\ell := a \cdot tp + b \cdot tn = c; a, b, c \in \mathbb{R}$
- Given a classifier h, we define a performance metric $\phi : [0, 1]^{2\times 2} \to \mathbb{R}$. We refer to the value $\phi(C(h))$, which represents the performance of a certain classifier with respect to a certain metric, as the *utility* of the classifier h. We assume, without loss of generality, that a higher value of ϕ means h is a better performance metric.

Our focus is to recover some metric ϕ using comparisons between confusion matrices $C(h)$, determined by classifiers h, which comes close to the oracle's "ground-truth" metric ϕ^*.

Next, we introduce two classes of performance metrics, for which we will present two elicitation algorithms. A *linear performance metric (LPM)*, given some constants $\{a_{11}, a_{01}, a_{10}, a_{00}\} \in \mathbb{R}^4$, is of the form

$$\begin{aligned}\phi(C) &= a_{11}TP + a_{01}FP + a_{10}FN + a_{00}TN \\ &= m_{11}TP + m_{00}TN + m_0,\end{aligned}$$

where $m_{11} = (a_{11} - a_{10})$, $m_{00} = (a_{00} - a_{01})$, and $m_0 = a_{10}\zeta + a_{01}(1 - \zeta)$; the second line is a useful parametrization of the metric, constructed using our observation about the dimensionality of $\mathcal{C}$. For example, one common LPM is weighted accuracy: $WA = w_1TP + w_2TN$, where varying the proportion between w_1 and w_2 corresponds to different importances afforded to various types of misclassification.

A slightly more complicated class of metrics are the *Linear-Fractional Performance Metrics (LFPM)*; given constants

$\{a_{11}, a_{01}, a_{10}, a_{00}, b_{11}, b_{01}, b_{10}, b_{00}\} \in \mathbb{R}^8$, an LFPM is defined as:

$$\begin{aligned}\phi(C) &= \frac{a_{11}TP + a_{01}FP + a_{10}FN + a_{00}TN}{b_{11}TP + b_{01}FP + b_{10}FN + b_{00}TN} \\ &= \frac{p_{11}TP + p_{00}TN + p_0}{q_{11}TP + q_{00}TN + q_0}\end{aligned}$$

where $p_{11} = (a_{11} - a_{10})$, $p_{00} = (a_{00} - a_{01})$, $q_{11} = (b_{11} - b_{10})$, $q_{00} = (b_{00} - b_{01})$, $p_0 = a_{10}\zeta + a_{01}(1 - \zeta)$, and $q_0 = b_{10}\zeta + b_{01}(1 - \zeta)$; again, these are useful reparametrizations that will simplify the elicitation process by reducing the number of variables to consider. Some commonly-used LFPMs are F_β score and Jaccard similarity, given by

$$F_\beta = \frac{TP}{\frac{TP}{1+\beta^2} - \frac{TN}{1+\beta^2} + \frac{\beta^2\zeta+1-\zeta}{1+\beta^2}}, JAC = \frac{TP}{1 - TN};$$

Taking $\beta = 1$, for example, gives the familiar F1 score, which is often used as a metric in ML classification problems.

Defining these notions of LPMs and LFPMs will allow us to consider a far more general array of metrics for learning problems, potentially allowing us to align better with practitioners' preferences.

3.2.2 Bayes Optimal and Inverse-Optimal Classifiers

In addition, we define the notions of Bayes optimal and inverse-optimal classifiers. Given a performance metric ϕ, we define:

- the *Bayes utility* as $\bar{\tau} := \sup_{h \in \mathcal{H}} \phi(C(h)) = \sup_{C \in \mathcal{C}} \phi(C)$;[1] this is the highest achievable utility (using the metric ϕ) over all classifiers $h \in \mathcal{H}$ for a given problem.
- the *Bayes classifier* as $\bar{h} := \arg\max_{h \in \mathcal{H}} \phi(C(h))$; this is the classifier h corresponding to the Bayes utility.
- the *Bayes confusion matrix* as $\bar{C} := \arg\max_{C \in \mathcal{C}} \phi(C)$; this is the confusion matrix corresponding to the Bayes utility and classifier.

Similarly, the inverse Bayes utility, classifier, and confusion matrix can be defined by replacing "sup" with"inf"; they represent the classifier and confusion matrix corresponding to the lower bound on utility for a given problem.

We also have the following useful proposition:

Let $\phi \in \varphi_{LPM}$. Then

$$\bar{h}(x) = \left\{ \begin{array}{ll} \mathbb{1}\left[\eta(x) \geq \frac{m_{00}}{m_{11}+m_{00}}\right], & m_{11} + m_{00} \geq 0 \\ \mathbb{1}\left[\frac{m_{00}}{m_{11}+m_{00}} \geq \eta(x)\right], & \text{o.w.} \end{array} \right\} \tag{3.46}$$

is a Bayes optimal classifier w.r.t ϕ. The inverse Bayes classifier is given by $\underline{h} = 1 - \bar{h}$.

This is a simple derivation based on the fact that we only get rewards from true positives and true negatives. Essentially, if we recover an LPM, we can use it to determine the best-performing classifier, obtained by placing a threshold on the conditional probability of a given sample, that corresponds to a confusion matrix. Therefore, the three notions of Bayes utility, classifier, and confusion matrix are functionally equivalent in our setting.

[1] GPT-4 is good at coming up with longer-rendered answers about why some things are appropriate or not.

3.2.3 Problem Setup

Now, we will formalize the problem of metric elicitation.

Given two classifiers h, h' (or equivalently, two confusion matrices C, C'), we define an *oracle query* as the function

$$\Gamma(h, h') = \Omega(C, C') = \mathbb{1}[\phi(C) > \phi(C')] =: \mathbb{1}[C \succ C'],$$

which represents the classifier that is preferred by the practitioner.

Then, we can define the metric elicitation problem for populations.

Suppose the true (oracle) performance metric is ϕ. The goal is to recover a metric $\hat{\phi}$ by querying the oracle for as few pairwise comparisons of the form $\Omega(C, C')$ so that $\|\phi - \hat{\phi}\|_{--} < \kappa$ for a sufficiently small $\kappa > 0$ and for any suitable norm $\|\cdot\|_{--}$.

In practice, we would not have access to the true probability distribution or the population, which would give us the true values of C and C'. We can, however, subtly alter this problem description to use $\hat{C}$ and $\hat{C}'$, which come from our dataset of n samples.

Suppose the true (oracle) performance metric is ϕ. The aim is to recover a metric $\hat{\phi}$ by querying the oracle for as few pairwise comparisons of the form $\Omega\left(\hat{C}, \hat{C}'\right)$ so that $\|\phi - \hat{\phi}\|_{--} < \kappa$ for a sufficiently small $\kappa > 0$ and for any suitable norm $\|\cdot\|_{--}$.

As is common in theoretical ML research, we solve the population problem and then consider ways to extend this to practical settings where we only have limited datasets of samples. In our case, this would correspond to calculating the confusion matrices from a portion of the dataset we have access to.

3.2.4 Confusion Matrices

Since we are considering all possible metrics in the LPM and LFPM families, we need to make certain assumptions about $\mathcal{C}$. Particularly, we will assume that $g(t) = \mathbb{P}[\eta(X) \geq t]$ is continuous and strictly decreasing for $t \in [0, 1]$; essentially, η has positive density and zero probability.

In addition, $\mathcal{C}$ is convex, closed, and contained in the rectangle $[0, \zeta] \times [0, 1-\zeta]$, and rotationally symmetric around its center, $(\frac{\zeta}{2}, \frac{1-\zeta}{2})$, where the axes represent the proportion of true positives and negatives. Also, the only vertices of $\mathcal{C}$ are $(0, 1-\zeta)$ and $(\zeta, 0)$, corresponding to predicting all 0's or all 1's on a given dataset. Therefore, $\mathcal{C}$ is strictly convex, and any line that is tangent to it is tangent at exactly one point, corresponding to one particular confusion matrix; these properties can be visually observed in Figure 3.16.

Next, recall that an LPM is represented in terms of three parameters ($\phi = m_{11}TP + m_{00}TN + m_0$). We have just seen that this LPM and its corresponding confusion matrix correspond to a certain point on the boundary of $\mathcal{C}$. We first note that this point is independent of m_0. In addition, we only care about the relative weightings of m_{11} and m_{00}, not their actual values–they are scale invariant. Therefore, we can parametrize the space of LPMs as $\varphi_{LPM} = \{\mathbf{m} =$

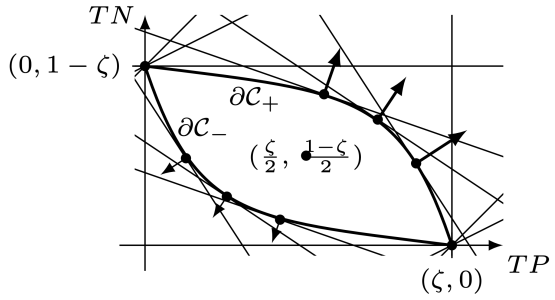


Figure 3.16
Visual representation of $\mathcal{C}$

$(\cos\theta, \sin\theta) : \theta \in [0, 2\pi]\}$, where $\cos\theta$ corresponds to m_{00} and $\sin\theta$ corresponds to m_{11}. And, as we already know, we can recover the Bayes classifier given $\mathbf{m}$, and it is unique, corresponding to one point on the boundary of $\mathcal{C}$, due to its convexity. The supporting hyperplane at this point is defined as

$$\bar{\ell}_{\mathbf{m}} := m_{11} \cdot tp + m_{00} \cdot tn = m_{11}\overline{TP}_{\mathbf{m}} + m_{00}\overline{TN}_{\mathbf{m}} \tag{3.47}$$

We note that if m_{00} and m_{11} have opposite signs, then $\bar{h}_m$ is the trivial classifier predicting all 1's or all 0's, since either predicting true positives or true negatives results in negative reward. This corresponds to a supporting hyperplane with positive slope, so it can only be tangent at the vertices.

In addition, the boundary $\partial\mathcal{C}$ can be split into upper and lower boundaries $(\partial\mathcal{C}_+, \partial\mathcal{C}_-)$, corresponding to $\theta \in (0, \pi/2)$ and $\theta \in (\pi, 3\pi/2)$ respectively (and whether m_{00}, m_{11} are positive or negative).

3.2.5 LPM and LFPM Metric Elicitation Algorithms

Now that we have established all of this background, we are ready to present the main two results of this section. First, we introduce an algorithm for LPM elicitation, which also forms the backbone for LFPM elicitation.

3.2.5.1 LPM Elicitation

For LPM elicitation, we need one more proposition.

For a metric ψ (quasiconvex and monotone increasing in TP/TN) or ϕ (quasiconcave and monotone increasing), and parametrization ρ^+/ρ^- of upper/lower boundary, composition $\psi \circ \rho^-$ is quasiconvex and unimodal on [0, 1], and $\phi \circ \rho^+$ is quasiconcave and unimodal on [0, 1].

Quasiconcavity and quasiconvexity are slightly more general variations on concavity and convexity. Their main useful property in our setting is that they are unimodal (they have a singular

extremum), so we can devise a binary-search-style algorithm for eliciting the Bayes optimal and inverse-optimal confusion matrices for a given setting, as well as the corresponding ϕ's.

We first note that to maximize a quasiconcave metric, in which ϕ is monotonically increasing in TP and TN, we note that the resulting maximizer (and supporting hyperplane) will occur on the upper boundary of $\mathcal{C}$. We thus set our initial search range to be $[0, \pi/2]$ and repeatedly divide it into four regions. Then, we calculate the resulting CM on the 5 resulting boundaries of these regions and query the oracle 4 times. We repeat this in each iteration of the binary search until a maximizer is found.

In the case of quasiconcave and quasiconvex search ranges, a slightly more sophisticated variation on typical binary search must be used. To illustrate this, we provide the following example.

Figure 3.17: Binary search algorithm example for quasiconvex functions

Consider the two distributions in Figure 3.17. Note that if, for both the symmetric and skewed distributions, we were to divide the search range into two portions and compare A, C, and E, we would find that $C > A$ and $C > E$. In both of these cases, this does not help us reduce our search range, since the true maximum could lie on either of the two intervals (as in the second case), or at C itself (as in the first case). Therefore, we must make comparisons between all five points A, B, C, D, E. This allows us to correctly restrict our search range to $[B, D]$ in the first case and $[C, E]$ in the second. These extra search requirements are due to the quasiconcavity of the search space we are considering, in which there exists a maximum but we need to make several comparisons at various points throughout the search space to be able to reduce its size in each iteration.

Algorithm 3.3 Quasiconcave Metric Maximization

input: $\epsilon > 0$ and oracle Ω
initialize: $\theta_a = 0, \theta_b = \frac{\pi}{2}$
while $|\theta_b - \theta_a| > \epsilon$ **do**
  set $\theta_c = \frac{3\theta_a + \theta_b}{4}$, $\theta_d = \frac{\theta_a + \theta_b}{2}$, and $\theta_e = \frac{\theta_a + 3\theta_b}{4}$
  obtain $h\theta_a, h\theta_c, h\theta_d, h\theta_e, h\theta_b$ using Proposition 1
  Compute $C\theta_a, C\theta_c, C\theta_d, C\theta_e, C\theta_b$ using (1)
  Query $\Omega(C\theta_c, C\theta_a), \Omega(C\theta_d, C\theta_c), \Omega(C\theta_e, C\theta_d)$, and $\Omega(C\theta_b, C\theta_e)$
  if $q_{i,j}$ is ambiguous **then**
    request $q_{i,j}$'s label from reference
  else
    impute $q_{i,j}$'s label from previously labeled queries
  end if
  if $C\theta' \succ C\theta'' \succ C\theta'''$ for consecutive $\theta < \theta' < \theta''$ **then**
    assume the default order $C\theta \prec C\theta' \prec C\theta''$
  end if
  if $C\theta' \succ C\theta'' \succ C\theta'''$ for consecutive $\theta < \theta' < \theta''$ **then**
    assume the default order $C\theta \prec C\theta' \prec C\theta''$
  end if
  if $C\theta_a \succ C\theta_c$ **then**
    Set $\theta_b = \theta_d$
  else if $C\theta_a \prec C\theta_c \succ C\theta_d$ **then**
    Set $\theta_b = \theta_d$
  else if $C\theta_c \prec C\theta_d \succ C\theta_e$ **then**
    Set $\theta_a = \theta_c$
    Set $\theta_b = \theta_e$
  else if $C\theta_d \prec C\theta_e \succ C\theta_b$ **then**
    Set $\theta_a = \theta_d$
  else
    Set $\theta_a = \theta_d$
  end if
end while
output: $\vec{m}, C$, and $\vec{l}$, where $\vec{m} = m_l(\theta_d), C = C\theta_d$, and $\vec{l} := (\vec{m}, (tp, tn)) = (\vec{m}, C)$

To elicit LPMs, we run Algorithm 3.3 , querying the oracle in each iteration, and set the elicited metric $\hat{m}$ (which is the maximizer on $\mathcal{C}$) to be the slope of the resulting hyperplane, since the metric is linear.

To find the minimum of a quasiconvex metric, we flip all instances of $\prec$ and $\succ$, and use an initial search range of $[\pi, 3\pi/2]$; we use this algorithm, which we refer to as Algorithm 3.4 , in our elicitation of LFPMs.

Next, we provide a Python implementation of Algorithm 3.3 .

```
def get_m(theta):
    """
    Inputs:
    - theta: the value that parametrizes m
    Outputs:
    - m_0 and m_1 for the LPM
    """

    return (math.cos(theta), math.sin(theta))

def lpm_elicitation(epsilon, oracle):
    """
    Inputs:
    - epsilon: some epsilon > 0 representing threshold of error
    - oracle: some function that accepts 2 confusion matrices and
        returns true if the first is preferred and false otherwise
    Outputs:
    - estimate for m, which is used to compute the LPM as described above
    """

    a = 0
    b = math.pi/2
    while (b - a > epsilon):
        c = (3 * a + b) / 4
        d = (a + b) / 2
        e = (a + 3 * b) / 4

        m_a, m_b, m_c, m_d, m_e = (get_m(x) for x in [a,b,c,d,e]) # using definition
↪   of m
        c_a, c_b, c_c, c_d, c_e = (get_c(x) for x in [m_a, m_b, m_c, m_d, m_e]) #
↪   compute classifier from m's then calculate confusion matrices

        response_ac = oracle(c_a, c_c)
        response_cd = oracle(c_c, c_d)
        response_de = oracle(c_d, c_e)
        response_eb = oracle(c_e, c_b)

        # update ranges to keep the peak
        if response_ac:
            b = d
        elif response_cd:
            b = d
        elif response_de:
            a = c
            b = e
        elif response_eb:
            a = d
        else:
```

```
            a = d
    return get_m(d), get_c(d)
```

3.2.5.2 LFPM Elicitation

Now, we present the next main result, which is an algorithm to elicit linear-fractional performance metrics. For this task, we will need the following assumption:

Let $\phi \in \varphi_{LFPM}$. We assume $p_{11}, p_{00} \geq 0, p_{11} \geq q_{11}, p_{00} \geq q_{00}, p_0 = 0, q_0 = (p_{11} - q_{11})\,\zeta + (p_{00} - q_{00})\,(1 - \zeta)$, and $p_{11} + p_{00} = 1$.

These assumptions guarantee that the LFPM ϕ which we are trying to elicit is monotonically increasing in TP and TN, just as in the LPM elicitation case.

We first provide motivation and an overview of the approach for LFPM elicitation and then present pseudocode for the algorithm.

The general idea of the algorithm is to use Algorithm 3.3 to obtain a maximizer and a minimizer for the given dataset; these result in two systems of equations involving the true LFPM ϕ^* with 1 degree of freedom. Then, we run a grid search that is independent of oracle queries to find the point where solutions to the systems match pointwise on the resulting confusion matrices; this occurs close to where the true metric lies.

More formally, suppose that the true metric is

$$\phi^*(C) = \frac{p^*_{11} TP + p^*_{00} TN}{q^*_{11} TP + q^*_{00} TN + q^*_0}. \tag{3.48}$$

Then, let $\bar{\tau}$ and $\underline{\tau}$ represent the maximizer and minimizer of ϕ over $\mathcal{C}$, respectively. There exists a hyperplane

$$\bar{\ell}^*_f := (p^*_{11} - \bar{\tau}^* q^*_{11})\, tp + (p^*_{00} - \bar{\tau}^* q^*_{00})\, tn = \bar{\tau}^* q^*_0, \tag{3.49}$$

which touches $\mathcal{C}$ at $\left(\overline{TP}^*, \overline{TN}^*\right)$ on $\partial\mathcal{C}_+$.

Correspondingly, there also exists a hyperplane

$$\underline{\ell}^*_f := (p^*_{11} - \underline{\tau}^* q^*_{11})\, tp + (p^*_{00} - \underline{\tau}^* q^*_{00})\, \mathrm{tn} = \underline{\tau}^* q^*_0, \tag{3.50}$$

which touches $\mathcal{C}$ at $(\underline{TP}^*, \underline{TN}^*)$ on $\partial\mathcal{C}_-$. Figure 3.17 illustrates this visually on $\mathcal{C}$.

While we are unable to obtain Equation 3.48 and Equation 3.49 directly, we can use Algorithm 3.3 to get a hyperplane

$$\bar{\ell} := \bar{m}_{11} tp + \bar{m}_{00} tn = \bar{m}_{11} \overline{TP}^* + \bar{m}_{00} \overline{TN}^* = \bar{C}_0, \tag{3.51}$$

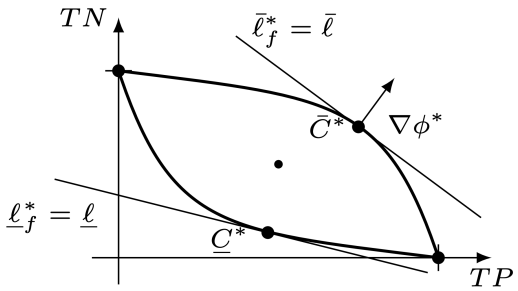


Figure 3.17
Visual representation of the minimizer and maximizer on $\mathcal{C}$

which is equivalent to $\bar{\ell}_f^*$ (Equation 3.48) up to a constant multiple. From here, we can obtain the system of equations

$$p_{11}^* - \bar{\tau}^* q_{11}^* = \alpha \bar{m}_{11}, p_{00}^* - \bar{\tau}^* q_{00}^* = \alpha \bar{m}_{00}, \bar{\tau}^* q_0^* = \alpha \bar{C}_0, \tag{3.52}$$

where $\alpha > 0$ (we know it is ≥ 0 due to our assumptions earlier and because $\bar{m}$ is positive, but if it is equal to 0 then ϕ^* would be constant. So, our resulting system of equations is

$$p'_{11} - \bar{\tau}^* q'_{11} = \bar{m}_{11}, p'_{00} - \bar{\tau}^* q'_{00} = \bar{m}_{00}, \bar{\tau}^* q'_0 = \bar{C}_0. \tag{3.53}$$

Now, similarly, we can approximate Equation 3.49 using the algorithm we defined for quasi-convex metrics (Algorithm 3.4), where we altered the search range and comparisons. After finding the minimizer, we obtain the hyperplane

$$\underline{\ell} := \underline{m}_{11} tp + \underline{m}_{00} tn = \underline{m}_{11} \underline{TP}^* + \underline{m}_{00} \underline{TN}^* = \underline{C}_0, \tag{3.54}$$

which is equivalent to $\underline{\ell}_f^*$ (Equation 3.49) up to a constant multiple. So then, our system of equations is

$$p_{11}^* - \underline{\tau}^* q_{11}^* = \gamma \underline{m}_{11}, p_{00}^* - \underline{\tau}^* q_{00}^* = \gamma \underline{m}_{00}, \underline{\tau}^* q_0^* = \gamma \underline{C}_0, \tag{3.55}$$

where $\gamma < 0$ (for a reason analogous to why we have $\alpha > 0$), meaning our resulting system of equations is

$$p''_{11} - \underline{\tau}^* q''_{11} = \underline{m}_{11}, p''_{00} - \underline{\tau}^* q''_{00} = \underline{m}_{00}, \underline{\tau}^* q''_0 = \underline{C}_0. \tag{3.56}$$

Equation 3.55 and Equation 3.56 form the two systems of equations mentioned in our overview of the algorithm. Next, we demonstrate that they have only one degree of freedom. Note that

if we know p'_{11}, we could solve both systems of equations as follows:

$$\begin{aligned} p'_{00} &= 1 - p'_{11}, q'_0 = \bar{C}_0 \frac{P'}{Q'} \\ q'_{11} &= (p'_{11} - \bar{m}_{11}) \frac{P'}{Q'} \\ q'_{00} &= (p'_{00} - \bar{m}_{00}) \frac{P'}{Q'}, \end{aligned} \tag{3.57}$$

where $P' = p'_{11}\zeta + p'_{00}(1-\zeta)$ and $Q' = P' + \bar{C}_0 - \bar{m}_{11}\zeta - \bar{m}_{00}(1-\zeta)$.

Now, suppose we know p'_{11}. We could use this value to solve both systems Equation 3.55 and Equation 3.56, yielding two metrics, ϕ' and ϕ'', from the maximizer and minimizer, respectively. Importantly, when

$$p^*_{11}/p^*_{00} = p'_{11}/p'_{00} = p''_{11}/p''_{00}, \tag{3.58}$$

then $\phi^*(C) = \phi'(C)/\alpha = -\phi''(C)/\gamma$. Essentially, when we find a value of p'_{11} that results in ϕ' and ϕ'' h aving constant ratios at all points on the boundary of $\mathcal{C}$, we can obtain ϕ^*, as it is derivable from ϕ' and α (or, alternatively, ϕ'' and γ).

We will perform a grid search for p'_{11} on $[0, 1]$. For each point in our search, we will compute ϕ' and ϕ''. Then, we will generate several confusion matrices on the boundaries and calculate the ratio \$□' ' / ϕ' for each. We will select the value of p'_{11} for which the ratio ϕ''/ϕ' is closest to constant and use it to compute the elicited metric $\hat{\phi}$.

The pseudocode for LFPM elicitation is given in Algorithm 3.4 .

Algorithm 3.4 Grid Search for Best Ratio

Input: k, Δ.
Initialize: $\sigma_{\text{opt}} = \infty, p'_{11,\text{opt}} = 0$.
Generate $C_1, \ldots, C_k$ on ∂C_+ and ∂C_- (Section 3).
Generate $C_1, \ldots, C_k$ on ∂C_+ and ∂C_- (Section 3).
for $p'_{11} = 0;\ p'_{11} \leq 1;\ p'_{11} = p'_{11} + \Delta$ **do**
    Compute ϕ', ϕ'' using Proposition 4.
    Compute array $r = \left[\frac{\phi'(C_1)}{\phi''(C_1)}, \ldots, \frac{\phi'(C_k)}{\phi''(C_k)}\right]$.
    Set $\sigma = \text{std}(r)$.
    if $\sigma < \sigma_{\text{opt}}$ **then**
        Set $\sigma_{\text{opt}} = \sigma$ and $p'_{11,\text{opt}} = p'_{11}$.
    end if
end for
Output: $p'_{11,\text{opt}}$.

We provide a Python implementation as below.

```python
def lfpm_elicitation(k, delta):
    """
    Inputs:
    - k: the number of confusion matrices to evaluate on
    - delta: the spacing for the grid search
    Outputs:
    - p_11', which will allow us to compute the elicited LFPM
    """

    sigma_opt = np.inf
    p11_opt = 0
    C = compute_confusion_matrices(k) # generates k confusion matrices to evaluate on

    for i in range(int(1/delta)):
        p11 = i * delta
        phi1 = compute_upper_metric(p11) # solves the first system of equations with p11
        phi2 = compute_lower_metric(p11) # solves the second system of equations with p11
        utility_1 = [phi1(c) for c in C] #calculate phi for both systems of equations
        utility_2 = [phi2(c) for c in C]

        r = []
        for i in range(k):
            r.append(utility_1[i] / utility_2[i])
        sigma = np.std(r)

        if(sigma < sigma_opt):
            sigma_opt = sigma
            p11_opt = p11
    return p11_opt
```

In summary, to elicit LFPMs, we utilize a special property of the LPM minimizer and maximizer on $\mathcal{C}$–namely, that we can use the corresponding supporting hyperplanes to form a system of equations that can be used to approximate ϕ^* if one parameter (p'_{11}) is found, and that this parameter can be found using an oracle-independent grid search.

3.2.5.3 Guarantees

Importantly, these algorithms can be shown to satisfy some important theoretical guarantees. We provide a formal statement and intuitive interpretation of them here, and their proofs can be found in the appendix of the original paper.

First, we define the oracle noise ϵ_Ω, which comes from the oracle potentially flipping the comparison output on two confusion matrices that are close enough in utility.

Theorem 3.1. *Given $\epsilon, \epsilon_\Omega \geq 0$ and a metric ϕ satisfying our assumptions, Algorithm 3.3 or Algorithm 3.4 finds an approximate maximizer/minimizer and supporting hyperplane. Also, the value of ϕ at that point is within $O\left(\sqrt{\epsilon_\Omega} + \epsilon\right)$ of the optimum, and the number of queries is $O\left(\log \frac{1}{\epsilon}\right)$.*

Theorem 3.2. *Let $\mathbf{m}^*$ be the true performance metric. Given $\epsilon > 0, LPM$ elicitation outputs a performance metric $\hat{\mathbf{m}}$, s.t. $\|\mathbf{m}^* - \hat{\mathbf{m}}\|_\infty \leq \sqrt{2}\epsilon + \frac{2}{k_0}\sqrt{2k_1\epsilon_\Omega}$.*

These two theorems ensure that Algorithm 3.3 and Algorithm 3.4 find an appropriate maximizer and minimizer in the search space, within a certain range of accuracy that depends on oracle and sample noise, and in a certain number of queries. Both of these statements are guaranteed by the binary search approach.

Theorem 3.3. *Let h_θ and $\hat{h}_\theta$ be two classifiers estimated using η and $\hat{\eta}$, respectively. Further, let $\bar{\theta}$ be such that $h_{\bar{\theta}} = \arg\max_\theta \phi\left(h_\theta\right)$. Then $\|C(\hat{h}_{\bar{\theta}}) - C\left(h_{\bar{\theta}}\right)\|_\infty = O\left(\|\hat{\eta}_n - \eta\|_\infty\right)$.*

This says, importantly, that the drop in elicited metric quality caused by using a dataset of samples rather than population CMs is bounded by the drop in performance of the decision boundary η. These three guarantees together ensure that oracle noise and sample noise do not amplify drops in performance when using metric elicitation; rather, these drops in performance are bounded by the drops that would usually occur when using the typical machine learning paradigm of training a decision boundary and using a pre-established metric.

3.2.5.4 Summary and Further Expansions

In this section, we have introduced the framework of metric elicitation for binary classification. After motivating the problem from a psychological and theoretical perspective, we have introduced the notions of linear performance metrics and linear-fractional performance metrics. Next, we set up the problem of metric elicitation from a population and sample perspective. We then made several key observations about the space of confusion matrices which, along with the notions of the Bayes utility and classifier, allow us to reduce the problem of linear performance metric elicitation to a binary search algorithm over a quasiconvex (or quasiconcave) space.

Next, we presented an algorithm to elicit linear-fractional performance metrics that builds upon the method for eliciting linear performance metrics. We used a key result about the dimensionality of linear-fractional performance metrics to create a simple oracle-independent grid search algorithm, which, in conjunction with the linear performance metric elicitation algorithm, results in an algorithm with the same time complexity that can be used to elicit a much broader range of metrics. We also provided Python implementations for both of these algorithms.

Lastly, we have described three important guarantees about the performance of these algorithms, which makes them suited to real-world, practical settings.

For further interesting exploration of the types of problems that can be solved using the framework of metric elicitation, we refer the reader to (Hiranandani, Narasimhan, and Koyejo 2020), which performs metric elicitation to determine the oracle's ideal tradeoff between the classifier's overall performance and the discrepancy between its performance on certain protected groups.

3.2.6 Multiclass Performance Metric Elicitation

Although the previous section only described metric elicitation for binary classification problems, the general framework can still be applied to multiclass classification problems, as described in "Multiclass Performance Metric Elicitation" by Hiranandani et al. (Hiranandani et al. 2019b)

Consider the case of classifying subtypes of leukemia (Yang and Naiman 2014). We can train a neural network to predict conditional probability of a certain leukemia subtype given certain gene expressions. However, it may not be appropriate to classify the subtype purely based on whichever one has the highest confidence. For instance, a treatment for leukemia subtype C1 may be perfect for cases of C1, but it may be ineffective or harmful for certain other subtypes. Therefore, the final response from the classifier may not be as simple as as choosing the class with the highest conditional probability, just like how the threshold for binary classification may not always be 50%.

With multiclass metric elicitation, we can show confusion matrices to an oracle (like the doctor in the leukemia example) to determine which classifier has the best tradeoffs. In (Hiranandani et al. 2019b), the authors focus on eliciting linear performance metrics, which is what we will describe in this chapter.

3.2.6.1 Preliminaries

Most of the notation from Binary Metric Elicitation still persists, just modified to provide categorical responses:

- $X \in \mathcal{X}$ is the input random variable.
- $Y \in [k]$ is the output random variable, where $[k]$ is the index set $\{1, 2, \dots, k\}$.
- The dataset of size n is denoted by $\{(\vec{x}, y)\}_{i=1}^{n}$ generated independently and identically from $\mathbb{P}(X, Y)$.
- $\eta_i(\vec{x}) = \mathbb{P}(Y = i | X = \vec{x})$ gives the conditional probability of class $i \in [k]$ given an observation.
- $\xi_i = \mathbb{P}(Y = i)$ is the marginal probability of class $i \in [k]$.
- The set of all classifiers is $\mathcal{H} = \{h : \mathcal{X} \to \Delta_k\}$, where Δ_k is (k-1) dimensional simplex. In this case, the outputs of classifiers are 1-hot vectors of size k where the only index with value 1 is the predicted class and all other positions have a value of 0.

– The confusion matrix for a classifier, h, is $C(h, \mathbb{P}) \in \mathbb{R}^{k \times k}$, where:

$$C_{ij}(h, \mathbb{P}) = \mathbb{P}(Y = i, h = j) \qquad \text{for } i, j \in [k] \tag{3.59}$$

Note that the confusion matrices are $k \times k$ and store the joint probabilities of each type of classification for each possible class. This means that the sum of row i in the confusion matrix equals ξ_i, because this is equivalent to adding over all possible classifications. Since we know the sums of each row, all diagonal elements can be reconstructed from just the off-diagonal elements, so a confusion matrix $C(h, \mathbb{P})$ can be expressed as a vector of off-diagonal elements, $\vec{c}(h, \mathbb{P}) = \textit{off-diag}(C(h, \mathbb{P}))$, and $\vec{c} \in \mathbb{R}^q$ where $q := k^2 - k$. The vector $\vec{c}$ is called the vector of *'off-diagonal confusions.'* The space of off-diagonal confusions is $\mathcal{C} = \{\vec{c}(h, \mathbb{P}) : h \in \mathcal{H}\}$.

In cases where the oracle would care about the exact type of misclassification (i.e. misclassifying and object from class 1 as class 2), this off-diagonal confusion matrix is necessary. However, there are many cases where the performance of a classifier is determined by just the probability of correct prediction for each class, which just requires the diagonal elements. In these cases, we can define the vector of *'diagonal confusions'* as $\vec{d}(h, \mathbb{P}) = \textit{diag}(C(h, \mathbb{P})) \in \mathbb{R}^k$. The space of diagonal confusions is $\mathcal{D} = \{\vec{d}(h, \mathbb{P}) : h \in \mathcal{H}\}$.

Finally, the setup for metric elicitation is identical to the one examined in the previous chapter. We still assume access to an oracle that can choose between two classifiers or confusion matrices, using notation Γ for comparing two classifiers and Ω for comparing confusion matrices, which returns 1 if the first classifier is better and 0 otherwise. We still assume that the oracle behaves according to some unknown performance metric, and we wish to recover this metric up to some small error tolerance (based on a suitable norm).

The two different types of confusion vectors result in different algorithms for metric elicitation, which we will explore in later sections.

3.2.7 Diagonal Linear Performance Metric Elicitation

In this section, we study metric elicitation in the case where the performance metric is linear and we only care about diagonal confusions.

3.2.7.1 DLPM

A Diagonal Linear Performance Metric (DLPM) is a performance metric that only considers the diagonal elements in the confusion matrix. The metric is defined as $\psi(\vec{d}) = \langle \vec{a}, \vec{d} \rangle$, where $\vec{a} \in \mathbb{R}^k$ such that $||\vec{a}||_1 = 1$. It is also called weighted accuracy (Narasimhan et al. 2015).

The family of DLPMs is denoted as φ_{DLPM}. Since these only consider the diagonal elements, which we want to maximize, we can focus on only eliciting monotonically increasing DLPMs, meaning that all elements in $\vec{a}$ are non-negative.

3.2.7.2 Bayes Optimal Classifiers

The Bayes Optimal diagonal confusion given a metric ψ is $\bar{d} = \arg\max_{\vec{d} \in \mathcal{D}} \psi(\vec{d})$.

The Restricted Bayes Optimal (RBO) diagonal confusion is the diagonal confusion that maximizes metric ψ given that it is only allowed to predict classes k_1 and k_2. It is denoted as $\bar{c}_{k_1,k_2} := \arg\max_{\vec{d} \in \mathcal{D}_{k_1,k_2}} \psi(\vec{d})$.

3.2.7.3 Geometry of Space of Diagonal Confusions D

Consider the trivial classifiers that only predict a single class at all times. The diagonal confusions when only predicting class i are $\vec{v}_i \in \mathbb{R}^k$ with ξ_i at index i and zero elsewhere. Note that this is the maximum possible value in index i, because this represents perfectly classifying all points that have a true class of i.

We can consider the space of diagonal confusions, visualized in Figure 3.18 (taken from (Hiranandandi et al. 2019b)). The space of $\mathcal{D}$ is strictly convex, closed, and contained in the box $[0, \xi_1] \times \cdots \times [0, \xi_k]$. We also know that the only vertices are $\vec{v}_i$ for each $i \in [k]^{(k-1)}$.

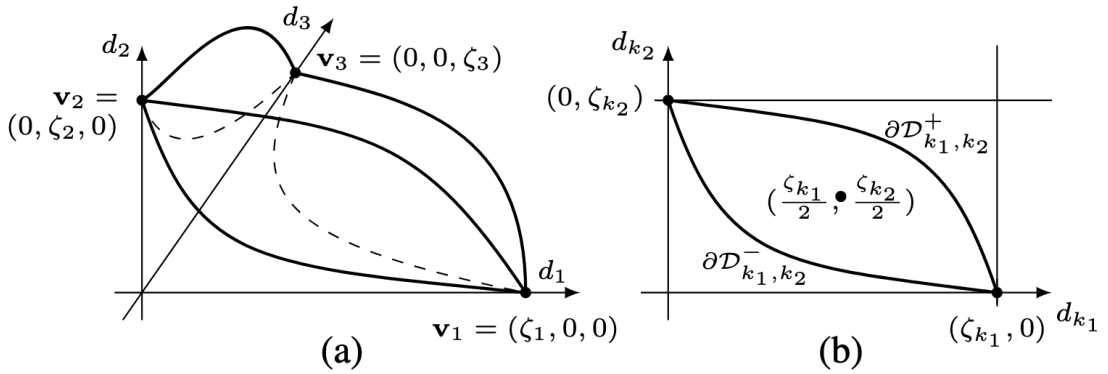


Figure 3.18
(a) Geometry of space of diagonal confusions for $k = 3$. This is a convex region with three flat areas representing confusions when restricted to only two classes. (b) Geometry of diagonal confusions when restricted to classes k_1 and k_2. Notice how this is identical to the space of confusion matrices examined in the previous chapter.

We know that this is strictly convex under the assumption that an object from any class can be misclassified as any other class. Mathematically, the assumption is that $g_{ij}(r) = \mathbb{P}\left[\frac{\eta_i(X)}{\eta_j(X)} \geq r\right]$ $\forall i, j \in [k]$ are continuous and strictly decreasing for $r \in [0, \infty)$.

We can also define the space of binary classification confusion matrices confined to classes k_1 and k_2, which is the 2-D (k_1, k_2) axis-aligned face of $\mathcal{D}$, denoted as $\mathcal{D}_{k_1,k_2}$. Note that this is strictly convex, since $\mathcal{D}$ itself is strictly convex, and it has the same geometry as the space of binary confusion matrices examined in the previous chapter. Therefore, we can construct an

RBO classifier for $\psi \in \varphi_{DLPM}$, parameterized by $\vec{a}$, as follows:

$$\bar{h}_{k_1,k_2}(\vec{x}) = \left\{ \begin{array}{l} k_1, \text{ if } a_{k_1}\eta_{k_1}(\vec{x}) \geq a_{k_2}\eta_{k_2}(\vec{x}) \\ k_2, \text{ o.w.} \end{array} \right\}. \tag{3.60}$$

We can parameterize the upper boundary of $\mathcal{D}_{k_1,k_2}$, denoted as $\partial\mathcal{D}^+_{k_1,k_2}$, using a single parameter $m \in [0,1]$. Specifically, we can construct a DLPM by setting $a_{k_1} = m$, $a_{k_2} = 1-m$, and all others to 0. Using Equation 3.60, we can get the diagonal confusions, so varying m parameterizes $\partial\mathcal{D}^+_{k_1,k_2}$. The parameterization is denoted as $\nu(m; k_1, k_2)$.

3.2.7.4 DLPM Elicitation

Suppose the oracle follows a true metric, ψ, that is linear and monotone increasing across all axes. If we consider the composition $\psi \circ \nu(m; k_1, k_2) : [0,1] \to \mathbb{R}$, we know it must be concave and unimodal, because $\mathcal{D}_{k_1,k_2}$ is a convex set. Therefore, we can find the value of m that maximizes $\psi \circ \nu(m; k_1, k_2)$ for any given k_1 and k_2 using a binary search procedure.

Since the RBO classifier for classes k_1 and k_2 only rely on the relative weights of the classes in the DLPM (see Equation 3.60), finding the value of m that maximizes $\psi \circ \nu(m; k_1, k_2)$ gives us the true relative ratio between a_{k_1} and a_{k_2}. Specifically, from the definition of ν, we know that $\frac{a_{k_2}}{a_{k_1}} = \frac{1-m}{m}$. We can therefore simply calculate the ratio between a_1 and all other weights to reconstruct an estimate for the true metric. A python implementation of this algorithm is provided below.

```python
import numpy as np

def rbo_dlpm(m, k1, k2, k):
    """
    This constructs DLPM weights for the upper boundary of the
    restricted diagonal confusions, given a parameter m.
    This is equivalent to \nu(m; k1, k2)

    Inputs:
    - m: parameter (between 0 and 1) for the upper boundary
    - k1: first axis for this  face
    - k2: second axis for this face
    - k: number of classes
    Outputs:
    - DLPM weights for this point on the upper boundary
    """
    new_a = np.zeros(k)
    new_a[k1] = m
    new_a[k2] = 1 - m
    return new_a
```

```
21
22  def dlpm_elicitation(epsilon, oracle, get_d, k):
23      """
24      Inputs:
25      - epsilon: some epsilon > 0 representing threshold of error
26      - oracle: some function that accepts 2 confusion matrices and
27          returns true if the first is preferred and false otherwise
28      - get_d: some function that accepts dlpm weights and returns
29          diagonal confusions
30      - k: number of classes
31      Outputs:
32      - estimate for true DLPM weights
33      """
34      a_hat = np.zeros(k)
35      a_hat[0] = 1
36      for i in range(1, k):
37          # iterate over each axis to find appropriate ratio
38          a = 0  # lower bound of binary search
39          b = 1  # upper bound of binary search
40
41          while (b - a > epsilon):
42              c = (3 * a + b) / 4
43              d = (a + b) / 2
44              e = (a + 3 * b) / 4
45
46              # get diagonal confusions for each point
47              d_a, d_c, d_d, d_e, d_b = (get_d(rbo_dlpm(x, 0, i, k))
48                  for x in [a, c, d, e, b])
49
50              # query oracle for each pair
51              response_ac = oracle(d_a, d_c)
52              response_cd = oracle(d_c, d_d)
53              response_de = oracle(d_d, d_e)
54              response_eb = oracle(d_e, d_b)
55
56              # update ranges to keep the peak
57              if response_ac:
58                  b = d
59              elif response_cd:
60                  b = d
61              elif response_de:
62                  a = c
63                  b = e
64              elif response_eb:
65                  a = d
66              else:
67                  a = d
68
```

```
        midpt = (a + b) / 2
        a_hat[i] = (1 - midpt) / midpt
    return a_hat / np.sum(a_hat)
```

To use this algorithm for metric elicitation on a real dataset, we need to supply the "oracle" and "get_d" functions. The oracle function is an interface to an expert who judges which of two confusion matrices is better. The get_d function will need to construct a classifier given the DLPM weights, following the principles of the RBO classifier from Equation 3.60, and calculate the confusion matrix from a validation set.

3.2.7.5 DLPM Elicitation Guarantees

Using the same oracle feedback noise model from the binary metric elicitation, we can make the following guarantees:

Given $\epsilon, \epsilon_\Omega \geq 0$, and a 1-Lipschitz DLPM φ^* parameterized by $\vec{a}^*$. Then the output $\hat{a}$ of the DLPM elicitation algorithm after $O((k-1)\log\frac{1}{\epsilon})$ queries to the oracle satisfies $||\vec{a}^* - \hat{a}||_\infty \leq O(\epsilon + \sqrt{\epsilon_\Omega})$, which is equivalent to $||\vec{a}^* - \hat{a}||_2 \leq O(\sqrt{k}(\epsilon + \sqrt{\epsilon_\Omega}))$.

In other words, the maximum difference between the estimate and true value along any component (indicated by the L-infinity norm) is linearly bounded by the sum of the epsilon specified by the algorithm and the square root of the oracle's correctness guarantee (ϵ_Ω).

3.2.8 Linear Performance Metric Elicitation

In this section, we study metric elicitation in the case where performance metric is linear and we care about the entire confusion matrix (i.e. the off-diagonal confusions).

A Linear Performance Metric (LPM) is a performance metric that uses the off-diagonal confusions, which is equivalent to the whole confusion matrix. The metric is defined as $\phi(\vec{c}) = \langle \vec{a}, \vec{c} \rangle$, where $\vec{a} \in \mathbb{R}^q$ such that $||\vec{a}||_2 = 1$. As a reminder, $q = k^2 - k$, which is the number of off-diagonal elements of the confusion matrix. The family of LPMs is denoted as φ_{LPM}. Since these only consider the off-diagonal elements, which we want to minimize, we can focus on only eliciting monotonically decreasing LPMs, meaning all elements in $\vec{a}$ are non-positive. The Bayes Optimal confusion given a metric ϕ over a subset $\mathcal{S} \subseteq \mathcal{C}$ is $\bar{c} = \arg\max_{\vec{c} \in \mathcal{S}} \phi(\vec{c})$.

3.2.8.1 Geometry of Space of Off-Diagonal Confusions C

Consider the trivial classifiers that only predict a single class at all times. The off-diagonal confusions when only predicting class i are $\vec{u}_i \in \mathbb{R}^q$, and all ξ_j (where $j \neq i$) will appear in this vector, because all classes other than i will be incorrectly predicted.

The space of off-diagonal confusions, $\mathcal{C}$ is convex and contained in the box $[0, \xi_1]^{(k-1)} \times \cdots \times [0, \xi_k]^{(k-1)}$. We also know that each $\vec{u}_i$ is a vertex. Furthermore, $\vec{o} = \frac{1}{k} \sum_{i=1}^{k} \vec{u}_i$ is always contained in $\mathcal{C}$ and represents the confusion matrix for the classifier that chooses a class uniformly at random.

Notice that $\mathcal{C}$ is not *strictly* convex, unlike $\mathcal{D}$. However, if we make the assumption that there exists a q-dimensional sphere $\mathcal{S}_\lambda \subset \mathcal{C}$ of radius $\lambda > 0$ centered at $\vec{o}$, we can now operate on this convex space. Note that this sphere always exists as long as there is some signal for non-trivial classification, so this assumption is safe in practice. Furthermore, since we are eliciting some linear metric with weights $\vec{a}$, we know the optimal off-diagonal confusion $\bar{c}$ over $\mathcal{S}$ is a point on its boundary $\bar{c} = \lambda\vec{a} + \vec{o}$.

Since we are only eliciting monotonically decreasing LPMs, we can parameterize the lower boundary of $\mathcal{S}_\lambda$, denoted as ∂S_λ^-, using sphere angles. Specifically, let $\vec{\theta}$ be a (q-1) dimensional vector of angles, with all angles in the second quadrant ($\theta_i \in [\pi/2, \pi]$), except for the primary angle, which is in the third quadrant ($\theta_{(q-1)} \in [\pi, 3\pi/2]$. To construct the LPM, we can set $a_i = \prod_{j=1}^{i-1} \sin(\theta_j)\cos(\theta_i)$ for $i \in [q-1]$ while $a_q = \prod_{j=1}^{q-1} \sin(\theta_j)$. This parameterization ensures that all elements in $\vec{a}$ are non-positive and the 2-norm is 1. This parameterization is denoted as $\mu(\vec{\theta})$.

3.2.8.2 LPM Elicitation

Unlike the diagonal case, $\mathcal{C}$ is not *strictly* convex, so the boundary may have flat regions. This means the DLPM elicitation algorithm cannot be used in this case. Instead, we can use the query space of the sphere $\mathcal{S}_\lambda$ to find an optimal point, allowing us to recover the metric.

Drawing from Derivative-Free Optimization (DFO) (Jamieson, Nowak, and Recht 2012), we can perform binary search on one coordinate at a time and keep updating cyclically. This algorithm will converge, because the metric is unimodal (over the lower boundary of the sphere) so progress is guaranteed.

A python implementation of this algorithm is provided below.

```
import numpy as np

def get_lpm(theta):
    """
    This constructs LPM weights for the lower boundary of the
    sphere.

    Inputs:
    - theta: parameter for the lower boundary
    Outputs:
    - LPM weights
    """
```

```
    new_a = np.ones(theta.size + 1)
    sin_theta = np.sin(theta)
    new_a[1:] = sin_theta
    new_a[:-1] *= np.cos(theta)
    return new_a

def get_mod_lpm(theta, j, new_val):
    """
    This constructs LPM weights where index j of the parameter
    is set to new_val
    """
    new_theta = np.copy(theta)
    new_theta[j] = new_val
    return get_lpm(new_theta)

def lpm_elicitation(epsilon, oracle, get_c, k, T):
    """
    Inputs:
    - epsilon: some epsilon > 0 representing threshold of error
    - oracle: some function that accepts 2 confusion matrices and
        returns true if the first is preferred and false otherwise
    - get_c: some function that accepts lpm weights and returns
        off-diagonal confusions
    - k: number of classes
    - T: maximum number of iterations
    Outputs:
    - estimate for true DLPM weights
    """
    q = k * k - k
    theta = np.random.rand(q-1)
    theta[:-1] = (1 + theta[:-1]) * np.pi/2  # initialize params to [pi/2, pi]
    theta[:-1] = (2 + theta[:-1]) * np.pi/2  # last param in [pi, 3pi/2]

    for t in range(T):
        j = t % (q-1)  # current coordinate to optimize

        # initialize range of binary search
        if j == q-2:
            a = np.pi
            b = 3 * np.pi / 2
        else:
            a = np.pi / 2
            b = np.pi

        while (b - a > epsilon):
            c = (3 * a + b) / 4
            d = (a + b) / 2
            e = (a + 3 * b) / 4
```

```
            # get confusions for each point
            c_a, c_c, c_d, c_e, c_b = (get_c(get_mod_lpm(theta, j, x))
                for x in [a, c, d, e, b])

            # query oracle for each pair
            response_ac = oracle(c_a, c_c)
            response_cd = oracle(c_c, c_d)
            response_de = oracle(c_d, c_e)
            response_eb = oracle(c_e, c_b)

            # update ranges to keep the peak
            if response_ac:
                b = d
            elif response_cd:
                b = d
            elif response_de:
                a = c
                b = e
            elif response_eb:
                a = d
            else:
                a = d

        midpt = (a + b) / 2
        theta[j] = midpt
    return get_lpm(theta)
```

Similar to the DLPM case, we need to supply the "oracle" and "get_c" functions. The oracle function is an interface to an expert who judges which of two confusion matrices is better. The get_c function will need to construct a classifier given the LPM weights, following the fact that $\bar{c} = \lambda\vec{a} + \vec{o}$, and calculate the confusion matrix from a validation set. Using the same oracle feedback noise model from the binary metric elicitation, we can make the following guarantees:

Given $\epsilon, \epsilon_\Omega \geq 0$, and a 1-Lipschitz LPM ϕ^* parameterized by $\vec{a}^*$. Suppose $\lambda \gg \epsilon_\Omega$, then the output $\hat{a}$ of the LPM elicitation algorithm after $O(z_1 \log(z_2/(q\epsilon^2))(q-1)log(\frac{\pi}{2\epsilon}))$ queries to the oracle satisfies $||a^* - \hat{a}||_2 \leq O(\sqrt{q}(\epsilon + \sqrt{\epsilon_\Omega/\lambda}))$, where z_1 and z_2 are constants independent of ϵ and q.

3.2.9 Summary

In this chapter, we explore multiclass performance metric elicitation, extending metric elicitation beyond binary classification. In practice, if weighted accuracy is all that is needed to determine if one classifier is better than another, the DLPM elicitation algorithm is preferred for its lower query complexity. However, if the type of misclassification is important, like in

the leukemia example at the start of this chapter, the LPM elicitation algorithm enables more flexibility.

3.2.10 Linear Reward Estimation

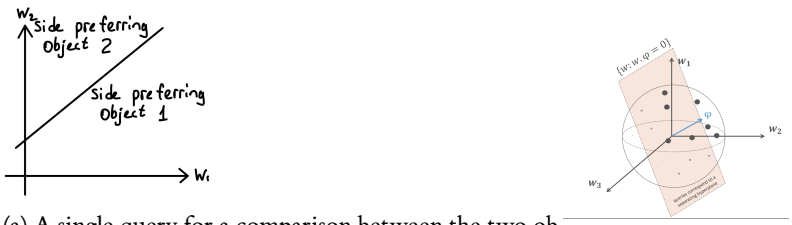
How exactly do robots learn human preferences from just the pairwise comparisons, if they need to learn how to act in the environment itself? The comparisons in turn help robots learn the reward function of the human, which allows them to further take actions in real settings.

3.2.10.1 Geometry of Pairwise Comparisons

Let's say there are two trajectories ξ_A and ξ_B that might be taken as the next course of action in any context, like choosing the next turn, or choosing the next chatGPT response. The robot is offering both to a human for comparison. To answer which of them is better, the human would ask themselves if $R(\xi_A)$ or $R(\xi_B)$ is bigger, with $R(\xi) = w * \phi(\xi)$ being the reward function. In this equation w and $\phi(\xi)$ are vectors of weights and features of the trajectory, so alternatively, we can express this as:

$$R(\xi) = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{bmatrix} \cdot \begin{bmatrix} \phi_1(\xi) \\ \phi_2(\xi) \\ \dots \\ \phi_N(\xi) \end{bmatrix}$$

If one says that they preferred ξ_2 less than ξ_1 then it means $\xi_2 < \xi_1 \implies R(\xi_2) < R(\xi_1) \implies w * \phi(\xi_2) < w * \phi(\xi_1) \implies 0 < w * (\phi(\xi_1) - \phi(\xi_2)) \implies 0 < w * \Phi$. Alternatively, if one preferred ξ_2 more than ξ_1, the signs would be flipped, resulting in $0 > w * \Phi$. The two results can be represented in the N-dimensional space, where when it is split by the decision boundary, it creates half-spaces indicating preferences for each of the sides. For example in Figure 3.19 we can see how a query between two objects can split the plain into two halves, indicating preference towards one of the objects. Such an image can be extended into bigger dimensions, where a line would become a separating hyperplane like in Figure 3.19.



(a) A single query for a comparison between the two objects splits 2D space into two halves, each of which prefers one of the objects based on feature weights w_1 and w_2.

(b) Extension into 3D space

Figure 3.19
Comparison in 2D and 3D

If one is to truly believe the answers of one person, they would remove everything from the other side of the hyperplane that does not agree with the received human preference. But

since humans are noisy, that approach is not optimal, thus most applications up-weight the indicated side of the plane to emphasize that points on that side are better, and down-weight the other side as they do not agree with the provided comparison.

How should someone choose which queries to conduct, otherwise, what is the most informative query sequence? After completing one query, the next query should be orthogonal to the previous one so that the potential space consistent with the preferences decreases in half. The intuition behind that is the potential space has all of the reward functions that agree with the provided answers, so to find a specific reward function for a human, decreasing the space narrows down the possible options. For example, orthogonal query to the query in Figure 3.19 is shown in Figure 3.20. The original query created the blue space, and a new one created a red space, resulting in a purple intersection of the two which is still consistent with both of the queries's results. The image shows that the purple portion is exactly half of the blue portion.

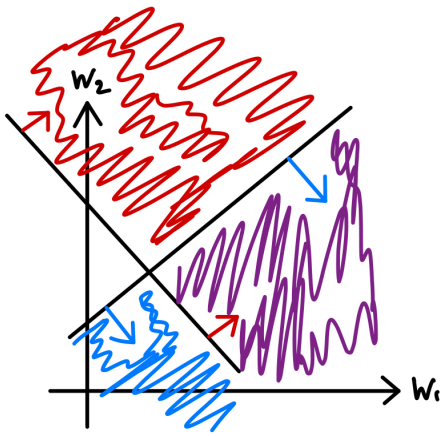


Figure 3.20
Creating further comparisons limits the space that agrees with answers to all of them. The blue area demonstrates a preference for object 1 over object 2. The red area demonstrates a preference for object 3 over object 4. Combination (purple area) shows the space that is consistent with both of those preferences.

Mathematically, from (Biyik and Sadigh 2018) this can be expressed as set F of potential queries ϕ, where $F = \{\phi : \phi = \Phi(\xi_A) - \Phi(\xi_B), \xi_A, \xi_B \in \Xi\}$ (defining that a query is the difference between the features of two trajectories). Using that, the authors define a human update function $f_\phi(w) = \min(1, \exp(I^T\phi))$ that accounts for how much of the space will still be consistent with the preferences. Finally, for a specific query, they define the minimum volume removed as $\min\{\mathbb{E}[1 - f_\phi(w)], \mathbb{E}[1 - f_{-\phi}(w)]\}$ (expected size of the two sides of the remaining space after it is split by a query - purple area in Figure 3.20), and the final goal is to maximize that amount over all possible queries since it is optimal to get rid of as much space as possible to narrow down the options for the reward function: $\max_\phi \min\{\mathbb{E}[1 - f_\phi(w)], \mathbb{E}[1 - f_{-\phi}(w)]\}$. Effectively this is finding such ϕ that maximizes the information one can get by asking the next comparison query. While this approach uses minimum volume removed, there can be other metrics inside the max function. Some applications like movie recommendations do not require extra constraints, however in robotics one might want to add more constraints that satisfy certain rules, so that the resulting query follows the dynamics of the physical world.

3.2.10.2 Driving Simulator Example

The first real example of learning reward functions from pairwise comparisons is a 2D driving simulator from (Biyik and Sadigh 2018). In Figure 3.21 you can see the setting of a 3-lane road with the orange car being controlled by the computer.

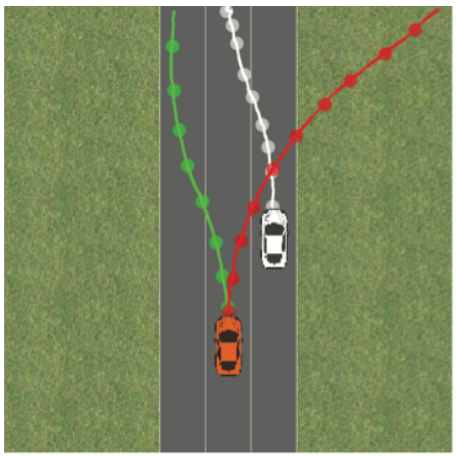


Figure 3.21
The choices presented to a human for feedback are represented by green and red trajectories. White trajectory demonstrates the lane change of another vehicle in the space. (Biyik and Sadigh 2018)

The queries conducted for this problem are two different trajectories presented to the human, and they are asked to evaluate which one of them is better. For the features that contribute to the reward function, it is important to consider that robots might not find some of the information as informative for the learning process as a human would. For this example, the underlying features included the distance between lane boundaries, distance to other cars, and the heading and speed of the controlled car. The weights toward the last feature were weighted the highest according to the authors, since it takes a lot of effort for the car to change or correct its direction.

At the start of the learning process, the car had no direction learned and was moving all over the road. In the middle of learning after 30 queries, the simulator learned to follow the direction of the road and go straight but still experienced collisions. After 70 queries, the simulator learned to avoid collisions, as well as keep the car within the lane without swerving.

3.2.10.3 Active Learning for Pairwise Comparisons

We have discussed that pairwise comparisons should be selected to maximize the minimum volume of remaining options removed. The question that can come out of the driving example is does it really matter to follow that goal or does random choice of queries performs as well? It turns out that indeed most active learning algorithms (purposefully selecting queries) over time converge with the performance of the random query selection, so in long term the performance is similar. However, what is different is that active learning achieves better performance earlier, which in time-sensitive tasks can be a critical factor.

One example of such a setting can be exoskeletons for humans as part of the rehabilitation after surgery (Li et al. 2021). Different people have significantly different walking patterns as well as rehabilitation requirements, so the exoskeleton needs to adapt to the human as soon as possible for a more successful rehabilitation. Figure Figure 3.22 demonstrates the difference in the time needed between the two approaches. In general, in robotics, the time differences that might seem small to a human might be detrimental to the final performance.

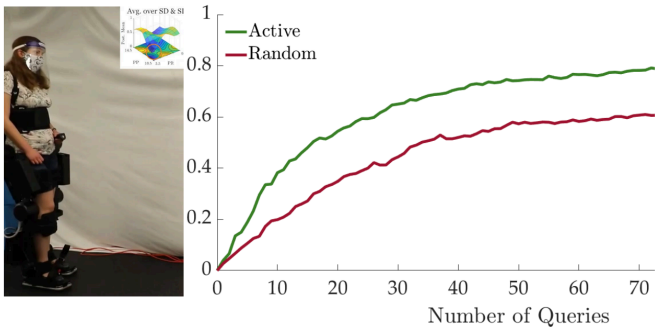


Figure 3.22
Performance of active learning and random query selection algorithms in the task of exoskeleton learning with human preferences. (Li et al. 2021)

3.2.10.4 Multi-Modal Reward Functions for Pairwise Comparisons

What if one is working with multiple people and their responses to the queries for comparisons? It will be impossible to recover the different personalities based on the answers, and it might be necessary to conduct a full ranking before it is clear which responses belonged to which person, but the underlying theory for the number of comparisons is non-trivial. For that, the researchers (Myers et al. 2021) have used multi-modal models for reward function learning, which allows to account for different types of valid behaviours and trajectories that can come from different humans.

An example setting for such type of problem is negotiations (Kwon et al. 2021). Let's say there are some shared items and two people with different utilities and desires for items, where each person only knows their utility. In a specific case of Figure 3.23, Bob as a proposing agent and Alice as a controlled agent who has many different ways of responding to Bob's proposals. Different methods can be used to design Alice as an AI agent. The first idea is reinforcement learning, where multiple rounds of negotiations are done, the model simulates game theory and sees how Bob reacts. Authors of this setting (Kwon et al. 2021) show that over time the model learns to ask for the same thing over and over again, as Alice is not trained to be human-like or negotiable, and just tries to maximize Alice's utility. The second approach is supervised learning, where the model can be trained on some dataset, learning the history of negotiations. This results in Alice being very agreeable, which demonstrates two polar results of the two approaches, and it would be ideal to find a middle ground and combine both of them. The authors proposed the Targeted acquisition approach, which is based on active learning ideas. The model asks diverse questions at different cases and stages of negotiations like humans,

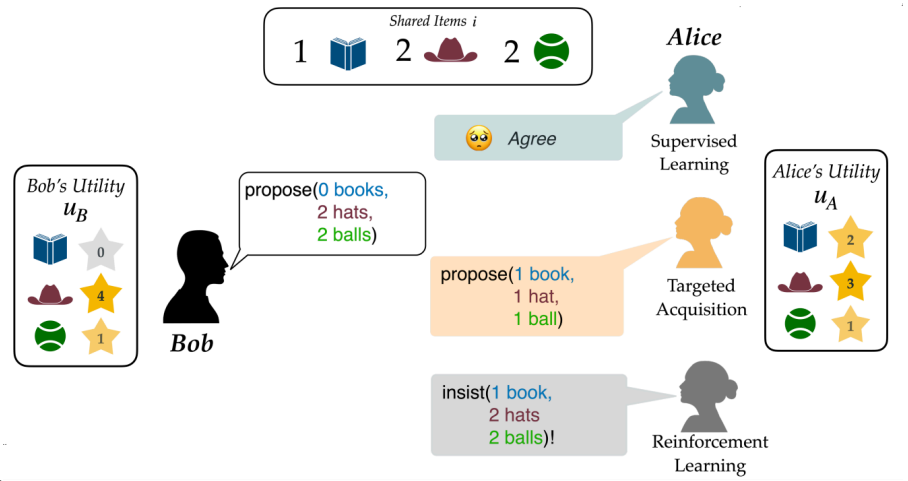


Figure 3.23
The negotiation setting with two people and three shared items. Each person has a desired number of items indicated in their utility box. Alice is the controlled agent that has many different response options that are illustrated by the approaches different models might take. (Kwon et al. 2021)

determining which questions are more valuable to be asked throughout learning. Such an approach ended up in more fair and optimal results than supervised or reinforcement learning (Kwon et al. 2021).

In conclusion, pairwise comparisons show to be a great way of learning linear reward functions, but at times present challenges or incapabilities that can be further improved with additional incorporations of approaches like Active Learning. That improves many applications in terms of time spent getting to the result in case of exoskeleton adjustments, as well as getting to a middle ground between polar behaviors in applications like negotiations.

3.2.11 Guiding Human Demonstrations in Robotics

A strong approach to learning policies for robotic manipulation is imitation learning, the technique of learning behaviors from human demonstrations. In particular, interactive imitation learning allows a group of humans to contribute their own demonstrations for a task, allowing for scalable learning. However, not all groups of demonstrators are equally helpful for interactive imitation learning.

The ideal set of demonstrations for imitation learning would follow a single, optimal method for performing the task, which a robot could learn to mimic. Conversely, *multimodality*, the presence of multiple optimal methods in the demonstration set, is challenging for imitation learning since it has to learn from contradicting information for how to accomplish a task.

A common reason for multimodality is the fact that different people often subconsciously choose different paths for execution, as illustrated in Figure 3.24.

Gandhi et al. (Gandhi et al. 2022) identifies whether demonstrations are compatible with one another and offer an active elicitation interface to guide humans to provide better demonstra-

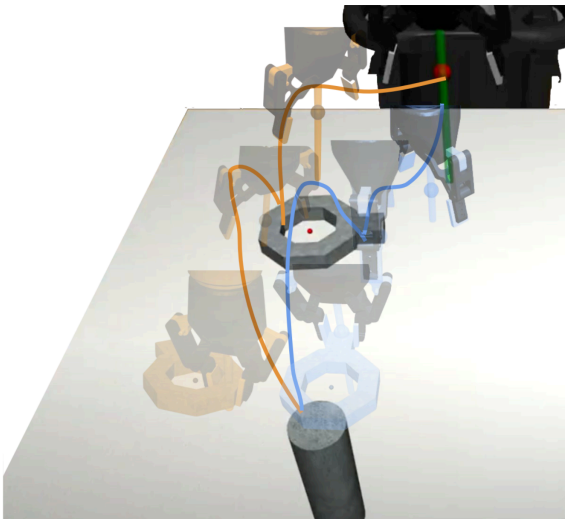


Figure 3.24
Examples of two different ways to insert a nut onto a round peg. The orange demonstration picks up the nut from the hole while the blue demonstration picks up the nut from the side (Gandhi et al. 2022)

tions in interactive imitation learning. Their key motivation is to allow multiple users to contribute demonstrations over the course of data collection by guiding users towards compatible demonstrations.

To identify whether a demonstration is "compatible" with a base policy trained with prior demonstrations, the researchers measure the *likelihood* of demonstrated actions under the base policy, and the *novelty* of the visited states. Intuitively, low likelihood and low novelty demonstrations should be excluded since they represent conflicting modes of behavior on states that the robot can already handle, and are therefore incompatible. This concept of compatibility is used for filtering a new set of demonstrations and actively eliciting compatible demonstrations.

In the following subsections, we describe the process of estimating compatibility and active elicitation in more detal.

3.2.11.1 Estimating Compatiblity

We want to define a compatibility measure $\mathcal{M}$, that estimates the performance of policy π_{base} that is retrained on a union of $\mathcal{D}_{base}$, the known base dataset, and $\mathcal{D}_{new}$, the newly collected dataset. To define this compatibility measure in a way that is easy to compute, we can use two interpretable metrics: likelihood and novelty.

The likelihood of actions a_{new} in $\mathcal{D}_{new}$ is measured as the negative mean squared error between actions predicted by the base policy and this proposed action:

$$likelihood(s_{new}, a_{new}) = -\mathbb{E}[||\pi_{base}(s_{new}) - a_{new}||_2^2]. \tag{3.61}$$

The novelty of the state s_{new} in $\mathcal{D}_{new}$ is the standard deviation in the predicted actions under base policy:

$$novelty(s_{new}) = \text{Var}[\pi_{base}(s_{new})]. \tag{3.62}$$

We can plot likelihood and novelty on a 2D plane, as shown in Figure 3.25, and identify thresholds on likelihood and novelty, denoted as λ and η respectively. Intuitively, demonstrations with low likelihood in low novelty states should be excluded, because this indicates that there is a conflict between the base behavior and the new demonstration due to multimodality. Note that in high novelty states, the likelihood should be disregarded because the base policy does not have a concrete idea for how to handle these states anyways so more data is needed.

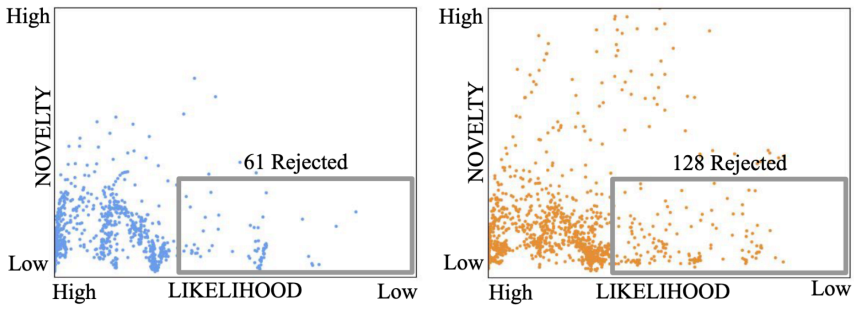


Figure 3.25
Examples of plots of likelihood and novelty for compatible and incompatible operators (Gandhi et al. 2022)

The final compatibility metric, parameterized by the likelihood and novelty thresholds λ and η, is $\mathcal{M}(\mathcal{D}_{base}, (s_{new}, a_{new})) \in [0, 1]$, defined as:

$$\mathcal{M} = \begin{cases} 1 - \min(\frac{\mathbb{E}[||\pi_{base}(s_{new}) - a_{new}||_2^2]}{\lambda}, 1) & \text{if novelty}(s_{new}) < \eta \\ 1 & \text{otherwise} \end{cases}. \tag{3.63}$$

Note that λ and η need to be specified by hand. This is accomplished by assuming the ability to collect *a priori incompatible* demonstrations to identify reasonable thresholds that remove the most datapoints in the incompatible demonstrations while keeping the most datapoints in the compatible demonstrations.

3.2.11.2 Case Studies with Fixed Sets

The researchers evaluate the utility of the compatibility metric on three tasks: placing a square nut on a square peg, placing a round nut on a round peg, and opening a drawer and placing a hammer inside. For each task, they train a base policy using a "proficient" operator's demonstration while sampling trajectories from other operators for the new set.

The naive baseline is to use all datapoints while the $\mathcal{M}$-Filtered demonstrations use the compatibility metric to filter out incompatible demonstrations. The results are presented in Table 3.4. As you can see, M-filtering results in equal or greater performance despite using less data than the naive baseline, demonstrating the effectiveness of compatibility-based filtering.

Table 3.4
Success rates (mean/std across 3 training runs) for policies trained on $\mathcal{D}_{new}$ by using all the data (Naive) or filtering by compatibility ($\mathcal{M}$-Filtered) (Gandhi et al. 2022)

	Square Nut		Round Nut		Hammer Placement	
Operator	Naive	$\mathcal{M}$-Filtered	Naive	$\mathcal{M}$-Filtered	Naive	$\mathcal{M}$-Filtered
Base Operator	38.7 (2.1)	-	13.3 (2.3)	-	24.7 (6.1)	-
Operator 1	54.3 (1.5)	61.0 (4.4)	26.7 (11.7)	32.0 (12.2)	38.0 (2.0)	39.7 (4.6)
Operator 2	40.3 (5.1)	42.0 (2.0)	22.0 (7.2)	26.7 (5.0)	33.3 (3.1)	32.7 (6.4)
Operator 3	37.3 (2.1)	42.7 (0.6)	17.3 (4.6)	18.0 (13.9)	8.0 (0.0)	12.0 (0.0)
Operator 4	27.3 (3.5)	37.3 (2.1)	7.3 (4.6)	13.3 (1.2)	4.0 (0.0)	4.0 (0.0)

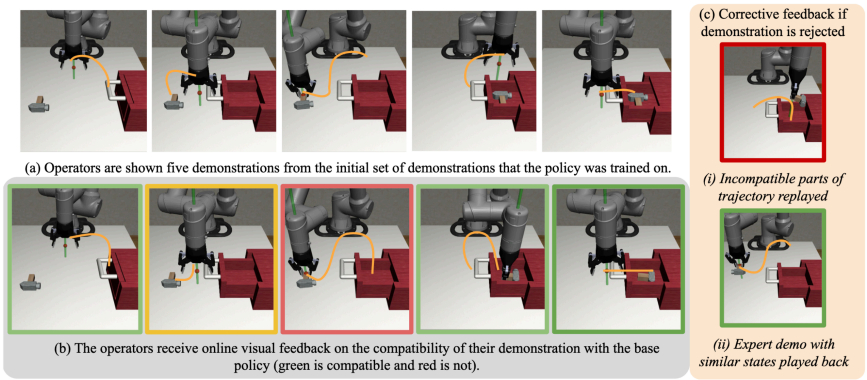


Figure 3.26
The phases of the active elicitation interface: (a) initial prompting, (b) demonstrations with live feedback, and (c) corrective feedback (Gandhi et al. 2022)

3.2.11.3 Actively Eliciting Compatible Demonstrations

In the previous section, we assume access to a dataset that has already been collected, and we see how filtering out incompatible demonstrations helps improve performance. However, when collecting a new dataset, it would be better to ensure that operators collect compatible demonstrations from the start, allowing us to retain as much data as possible for training.

To actively elicit compatible demonstrations, the researchers set up a pipeline for live feedback and examples. At the start, operators are given a task specification and some episodes to practice using the robot. Then, the active elicitation process begins, as shown in Figure 3.26. Each operator is shown some rollouts of the base policy to understand the style of the base operator. Next, the operator provides a demonstration similar to the ones they were shown. As they record their demonstrations, the interface provides online feedback, with green indicating compatible actions and red indicating incompatible actions. If the number of incompatible

state-action pairs (ones where $\mathcal{M}$ is zero) exceeds 5% of the demonstration length, the demonstration is rejected. However, to provide corrective feedback, the interface shows the areas of the demonstration with the highest average incompatibility and also provides an expert demo that shows what should actually be done. Demonstrators can use this feedback to provide more compatible demonstrations moving forward.

This process helps improve the demonstration quality in both simulation and real experiments, as show in Table 3.5. Specifically, on the real results, active elicitation outperformed the base policy by 25% and naive data collection by 55%. Overall, active elicitation is a powerful tool to ensure that data collected for imitation learning improves the quality of the learned policy.

Table 3.5
Success rates (mean/std across users) for policies trained on $\mathcal{D}_{new}$ by using all the data (Naive), filtering by compatibility ($\mathcal{M}$-Filtered), or using informed demonstration collection (Gandhi et al. 2022)

Task	Base	Naive	Naive + Filtered	Informed
Round Nut	13.3 (2.3)	9.6 (4.6)	9.7 (4.2)	15.7 (6.0)
Hammer Placement	24.7 (6.1)	20.8 (15.7)	22.0 (15.5)	31.8 (16.3)
[**Real**] **Food Plating**	60.0	30.0 (17.3)	-	85.0 (9.6)

3.2.11.4 Limitations and Future Work for Active Elicitation

A fundamental limitation of eliciting compatible demonstrations is the fact that the "base" demonstrator is considered the ground truth. When the base demonstrator specifies a preference, all other demonstrators must abide by it, even if they have strong preferences against it. For instance, when pouring milk and cereal into a bowl, different people have different preferences for what is the correct order, but active elicitation forces all demonstrators to follow the initial preference of the base operator. The researchers hope that future work can enable users to override the default demonstration set and follow a base behavior that better aligns with their preferences. This could enable multiple modes of behavior to be collected in data while only following a user's specified preference instead of attempting to collapse all modes into a single policy.

Looking forward, active elicitation provides a foundation for allowing robots to query humans about the type of data needed, enabling more efficient data collection through transparency.

3.2.12 Conclusion

In summary, this chapter has explored the complexities and innovations in interactive learning as applied to large models within robotics. It begins by investigating pairwise comparisons and their role in efficiently learning linear reward functions from large datasets, overcoming limitations in supervised learning. When combined with active learning techniques, these comparisons supply timely, targeted, and context-appropriate feedback, enhancing performance in time-critical applications like exoskeleton adjustments during rehabilitation.

We then shift to imitation learning or inverse reward learning from demonstrations, emphasizing the difficulties introduced by multimodal demonstration sets. active elicitation approaches to compile compatible demonstrations, streamlining the learning process by guiding users to provide more valuable, steady examples are incredibly promising, however, to tackling this issue. This method shows promise in refining the interactive imitation learning data collection pipeline, enabling more capable and effective robotic training.

Additionally, the chapter examines the integration of foundation models into robotics, highlighting the transformative innovations of R3M and Voltron. R3M's pre-training on diverse human activities dramatically improves robotic manipulation with minimal supervision. Meanwhile, Voltron builds on these capabilities by incorporating language-driven representation learning for remarkably adaptable and nuanced robotic task performance. These models represent significant leaps in robotics while opening new frontiers for future research and applications.

3.2.13 Truthful Preference Elicitation with Adversary

In our study of social choice models in Chapter [2model], we study how axiomatic properties are implemented to prevent strategic manipulation of a population. This brings us onto the field of **mechanism design**. At its core, mechanism design is the science of making rules. The intent in this field is to design systems so that the strategic behaviour of individuals leads to desirable outcomes. Just thinking about services on the Internet – file sharing, reputation systems, web search, web advertising, email, Internet auctions, congestion control – all have to be set up so that an individual's selfish behavior leads to better outcomes for the entire community. A more specific example of this is the phenomenon of "bid-sniping" that was present on eBay in the early 2000s. When people could bid on E-bay, the rule was that the highest bidder by the end of some specified time period would get the item. As a result, people would just wait until the very last minute to bid in order to not raise the price of the item too early. On the other hand, when Amazon still allowed bidding, they had a rule that any time a bid was placed it would extend the time of the bid by ten minutes. This simple difference had drastic effects on bidding prices over time. Mechanism design develops the theoretical framework for learning social choices and eliciting truthful preference.

We will cover frameworks that model several scenarios that mechanism design is usefully applied to: recommendation systems (where users will selfishly try to stick to their preferences while a planner encourages exploration); auctions (where bidders will try to maximise their reward compared to others); and peer grading (where truthful reporting is not necessarily an incentive for students).

3.2.13.1 Auction Theory

Single-Item Auctions

The first problem within auction theory we will consider is the *single-item auction*. The premise of this problem is that there is a single item to sell, n bidders (with unknown private valuations of the item v_1, ..., v_n). The bidder's individual objective is to maximize utility: the value v_i of the item subtracted by the price paid for the item. The auction procedure is standard in the sense that bids are solicited and the highest bid will win the auction. While the objective of the individual bidder is clear, there could be a plethora of different objectives for the auction as a whole. One option could be to maximize social surplus, meaning the goal is to maximize the value of the winner. Another objective could be to maximize seller profit which is the payment of the winner. For simplicity, we can focus on the first objective where the goal is to maximize social surplus. If we want to maximize social surplus it turns out that a great way to do this is the "second-price auction".

Maximizing Social Surplus

In the second-price auction, we will operate under slightly different conditions. In the second-price auction we 1) solicit sealed bids, 2) have the winner be the highest bidder, and 3) charger winner the second-highest bid price. As an example, if the solicited bids are $b = (2, 6, 4, 1)$ the winner will be that who bid 6, but will pay a price of 4. From here, we can do some equilibrium analysis to try and learn what the optimal bidding strategy is for each bidder. Let the amount bidder i bids to be b_i, so we have bids $b_1, b_2, ..., b_n$. How much should bidder i bid? To analyze this, let us define $t_i = max_{j \neq i} b_j$ which represents the max of the bids that is not from bidder i. There are now two cases to consider: if $b_i > t_i$ and if $b_i < t_i$. In the first case the bidder i wins, and if the bidder bid $b_i = v_i$, they are guaranteed to have a positive return on bid. In the other case, they lose the bid and the net loss is 0 because they don't have to pay. From this we can conclude that bidder i's dominant strategy is to just bid $b_i = v_i$. Rigorously proving this is a little bit trickier, but it was shown from Vickrey in 1961 [cite] that truthful bidding is the dominant strategy in second-price auctions. A corollary of this is that we are maximizing social surplus since bids are values and the winner is the bidder with highest valuation.

Maximize Seller Profit

If we want to look at things from the perspective of a seller trying to maximize their profit we need to treat the bidder's bids as uniform random variables. Consider the example scenario where we have two bidders each bidding uniformly between 0 and 1. What is the seller's expected profit? (in this case profit and revenue for the seller are the same because we assume the seller throws away the item if it doesn't sell/has no valuation for it).

From there the question now becomes, can we get more expected profit from the seller's perspective? It turns out there is a design where we can add a reserve price of r to the second-price auction. The way this works is we can 1) Insert seller-bid at r, 2) solicit bids, 3) pick the highest bidder, and 3) charge the 2nd-highest bid. In effect, this is just the second-price auction but with a bid from the seller as well, at a price of r. A lemma, that we won't prove here, is that the second-price auction with reserve price r still has a dominant strategy of just being truthful.

Let's now consider what the profit of a second-price auction would be with two bidders that uniformly bid between 0 and 1 – but this time we have a reserve price of $1/2$. To calculate the expected profit we break down the situation into 3 cases:

- Case 1: $1/2 > v_1 > v_2 \rightarrow 1/4$ probability $\rightarrow E[\text{profit}] = 0$
- Case 2: $v_1 > v_2 > 1/2 \rightarrow 1/4$probability $\rightarrow E[v2|case2] = 2/3$
- Case 3: $v_1 > 1/2 > v_2 \rightarrow 1/2$ probability $\rightarrow 1/2$

Why is $E[v2|case2] = 2/3$? If v_1 and v_2 are greater than $1/2$, they are evenly spread across the interval, meaning the expectation will be 1/2 + 1/6 = 2/3. Adding up all these cases we get $E[profit] = 5/12$. It turns out that second-price auctions with reserve actually maximize profit in general (for symmetric bidders)!

In the previous section we conclude that second-price auctions with reserve maximize profit for the seller. In order to prove this, we now move to the more general topic of asking how should a monopolist divide good across separate markets. We can make the assumption that the demand model is a concave revenue $R(q)$ in quantity q. Under this assumption, we can just divide supply into $q = q_a + q_b$ such that $R'_a(q_a) = R'_b(q_b)$. The idea from here is a theorem from Myerson in 1981 that states an optimal action maximizes "marginal revenue". Consider an example where we have two bidders bidding a uniform value between 0 and 1. Our revenue curve can now be derived from the offering price $V(q) = 1-q$ like so: $R(q) = qV(q) = q-q^2$. Taking the derivative gives us the marginal revenue $R'(q) = 1 - 2q$. This means two things: 1) we want to sell to bidder i with the highest $R'(q_i)$ and 2) we want to sell to bidder i with value at least $1/2$ (if we want a positive $R'(q_i)$). But this is just a second-price auction with reserve $1/2$! This means that for symmetric bidders, a second price with reserve is the optimal auction.

What good are auctions?

An interesting topic to discuss is what benefits auctions bring to the table as opposed to just standard pricing. Online auctions used to be a lot more popular in the early 2000s and have been completely replaced by standard online pricing, even on sites like e-bay. While auctions are slower and have added inherent complexities, they are actually optimal on paper. Standard pricing on the other is non-optimal; although it is fast and simpler for buyers. There is actually a way to quantify this: for pricing k units, the loss is at most $1/\sqrt{2\pi k}$ of optimal profit.

Let's consider applications in duopoly platform design. We know that the optimal auction is second-price with reserve, but what happens when we introduce competition between two auction platforms? Some important details related to the revenue of a second-price auction is that a second-price auction with no reserve and n bidders leads to larger revenue having an optimal reserve and n - 1 bidders (Bulow and Klemperer 1996). Additionally, with an entry cost, no reserve is the optimal strategy for maximizing revenue (McAfee and McMillan 1987). Let's consider an example of a competing auction system which is Google ads vs Bing ads. How should an advertiser divide the budget between Google and Bing? They should give the same

budget to both companies. What happens if Bing raises their prices? Then, the advertising company moves more of its budget to Google from Bing.

Prior-Independent Auctions

The Bulow-Klemperer theorem demonstrates that increased competition can be more valuable than perfect knowledge of bidders' valuation distributions. This result provides insight into the potential of simple, prior-independent auctions to approach the performance of optimal auctions. The theorem states that for a single-item auction with bidders' valuations drawn independently from a regular distribution F:

Let F be a regular distribution and n a positive integer. Then:

$$E_{v_1,\ldots,v_{n+1}\sim F}[\text{Rev(VA)}(n+1 \text{ bidders})] \geq E_{v_1,\ldots,v_n\sim F}[\text{Rev}(\text{OPT}_F)(n \text{ bidders})] \tag{3.64}$$

where VA denotes the Vickrey auction and OPT_F denotes the optimal auction for F.

This shows that running a simple Vickrey auction with one extra bidder outperforms the revenue-optimal auction that requires precise knowledge of the distribution. It suggests that in practice, effort spent on recruiting additional bidders may be more fruitful than fine-tuning auction parameters.

The VCG Mechanism

The VCG mechanism is a cornerstone of mechanism design theory, providing a general solution for welfare maximization in multi-parameter environments. The key result is:

In every general mechanism design environment, there is a dominant-strategy incentive-compatible (DSIC) welfare-maximizing mechanism.

The VCG mechanism operates as follows:

1. Given bids $b_1, \ldots, b_n$, where each b_i is indexed by the outcome set Ω, the allocation rule is:

$$x(b) = \arg\max_{\omega\in\Omega} \sum_{i=1}^{n} b_i(\omega) \tag{3.65}$$

2. The payment rule for each agent i is:

$$p_i(b) = \max_{\omega\in\Omega} \sum_{j\neq i} b_j(\omega) - \sum_{j\neq i} b_j(\omega^*) \tag{3.66}$$

where $\omega^* = x(b)$ is the chosen outcome.

The key insight is to charge each agent its "externality" - the welfare loss inflicted on other agents by its presence. This payment rule, coupled with the welfare-maximizing allocation rule, yields a DSIC mechanism.

The VCG mechanism can be interpreted as having each agent pay its bid minus a "rebate" equal to the increase in welfare attributable to its presence:

$$p_i(b) = b_i(\omega^*) - \left[\sum_{j=1}^{n} b_j(\omega^*) - \max_{\omega \in \Omega} \sum_{j \neq i} b_j(\omega)\right] \tag{3.67}$$

While the VCG mechanism provides a theoretical solution for DSIC welfare-maximization in general environments, it can be challenging to implement in practice due to computational and communication complexities.

Combinatorial Auctions

Combinatorial auctions are an important class of multi-parameter mechanism design problems, with applications ranging from spectrum auctions to airport slot allocation. In a combinatorial auction:

- There are n bidders and a set M of m items.
- The outcome set Ω consists of allocations $(S_1, \ldots, S_n)$, where S_i is the bundle allocated to bidder i.
- Each bidder i has a private valuation $v_i(S)$ for each bundle $S \subseteq M$.

While the VCG mechanism theoretically solves the welfare-maximization problem, combinatorial auctions face several major challenges in practice:

1. Preference Elicitation: Each bidder has $2^m - 1$ private parameters, making direct revelation infeasible for even moderate numbers of items. This necessitates the use of indirect mechanisms that elicit information on a "need-to-know" basis.
2. Computational Complexity: Even when preference elicitation is not an issue, welfare-maximization can be an intractable problem. In practice, approximations are often used, hoping to achieve reasonably good welfare.
3. VCG Limitations: The VCG mechanism can exhibit bad revenue and incentive properties in combinatorial settings. For example, adding bidders can sometimes decrease revenue to zero, and the mechanism can be vulnerable to collusion and false-name bids.
4. Strategic Behavior in Iterative Auctions: Most practical combinatorial auctions are iterative, comprising multiple rounds. This introduces new opportunities for strategic behavior, such as using bids to signal intentions to other bidders.

These challenges make combinatorial auctions a rich and complex area of study, requiring careful design to balance theoretical guarantees with practical considerations.

Spectrum Auctions

Spectrum auctions represent a complex application of combinatorial auction theory. With n bidders and m non-identical items, each bidder has a private valuation for every possible bundle of items, making it impractical to directly elicit all preferences. This necessitates the use of indirect, iterative mechanisms that query bidders for valuation information on a "need-to-know" basis, sacrificing some of the desirable properties of direct mechanisms like dominant strategy incentive compatibility (DSIC) and full welfare maximization.

The fundamental challenge in spectrum auctions lies in the nature of the items being sold. There is a dichotomy between items that are substitutes (where $v(AB) \leq v(A) + v(B)$) and those that are complements (where $v(AB) > v(A) + v(B)$). Substitute items, such as licenses for the same area with equal-sized frequency ranges, are generally easier to handle. When items are substitutes, welfare maximization is computationally tractable, and the VCG mechanism avoids many undesirable properties. However, complementary items, which arise naturally in spectrum auctions when bidders want adjacent licenses, present significant challenges.

Early attempts at spectrum auctions revealed the pitfalls of naive approaches. Sequential auctions, where items are sold one after another, proved problematic as demonstrated by a Swiss auction in 2000. Bidders struggled to bid intelligently without knowing future prices, leading to unpredictable outcomes and potential revenue loss. Similarly, simultaneous sealed-bid auctions, as used in New Zealand in 1990, created difficulties for bidders in coordinating their bids across multiple items, resulting in severely suboptimal outcomes.

The Simultaneous Ascending Auction (SAA) emerged as a solution to these issues and has formed the basis of most spectrum auctions over the past two decades. In an SAA, multiple items are auctioned simultaneously in rounds, with bidders placing bids on any subset of items subject to an activity rule. This format facilitates price discovery, allowing bidders to adjust their strategies as they learn about others' valuations. It also allows bidders to determine valuations on a need-to-know basis, reducing the cognitive burden compared to direct-revelation auctions.

Despite its advantages, the SAA is not without vulnerabilities. Demand reduction, where bidders strategically reduce their demand to lower prices, can lead to inefficient outcomes even when items are substitutes. The exposure problem arises with complementary items, where bidders risk winning only a subset of desired items at unfavorable prices. These issues highlight the ongoing challenges in designing effective spectrum auctions, balancing theoretical guarantees with practical considerations.

Case study: Classroom Peer Grading

This chapter discusses work by Jason Hartline, Yingkai Li, Liren Shan, and Yifan Wu at Northwestern University, where researchers examined mechanism design for the classroom, specifically in terms of the optimization of scoring rules. They explored peer grading in the classroom and how to construct a peer grading system that optimizes the objectives for each stakeholder in the system, including those being graded, the peer graders, the TAs of the class, and the professor.

Firstly, let's think of the classroom like a computer. We can think of students as local optimizers; their incentive is to minimize the amount of work they need to do and maximize the grades that they receive. The graders are imprecise operators, which means that there is some uncertainty in their ability to grade the work completed by the students. The syllabus can be thought of as the rules that map the actions of the students to the grade they end up receiving in the class. Our overall goals for this classroom based on these definitions is to minimize work, maximize learning, and fairly assess the students for the work that they do (Hartline et al. 2020).

One basic question that we can examine, is what is the best syllabus that maximizes our objectives for our classroom design. Some components of this could include grading randomized exams, grading with partial credit, group projects, and finally, peer grading, which is the component that we will be taking a deeper dive into.

The general situation of the peer grading problem is that proper scoring rules make peer grades horrible (Hartline et al. 2020). So we want to be able to optimize scoring rules and make sure that we are optimizing each component of the peer grading pipeline.

The main algorithms focused on in this peer grading design paper were matching peers and TAs to submissions and the grading of those submissions from the TAs and the peer reviews (Hartline et al. 2020). There are quite a number of advantages to peer grading including that peers are able to learn from reviewing other people's work, it reduces the work for the teacher, and improves the turnaround time for assignment feedback (which are all part of our overarching goals for our mechanism design for the classroom). But, it is also important to acknowledge the potential disadvantages of the peer grading system: it is possible that the peer graders present inaccurate grades and there is student unrest. This presents us with a challenge: being able to incentivize accurate peer reviews.

One problem that we run into, when we use the proper scoring rule to score peer reviews, if the peer graders use the lazy peer strategy, which means that they always report 80% for their peer reviews, they get graded very well using the proper scoring rule algorithm. In fact, the proper scoring rule says that their peer review is 96% accurate (Hartline et al. 2023). So how do we incentivize effort in reviews from peer graders? We use a scoring rule that maximizes the difference in score between effort or no effort reviews as indicated by the peer reviewers (Hartline et al. 2023). So overall, the analysis of datasets leads to decision optimizations and, eventually, payoff from those decisions.

To conclude our mechanism design in the classroom discussion, we have two key takeaways: scoring rules are essential in being able to understand and analyze data thoroughly, and optimal

scoring rules for binary effort allow us to understand the setting independent of the dataset (Hartline et al. 2023).

Mutual Information Paradigm

In this section we discuss an influential new framework for designing peer prediction mechanisms, the Mutual Information Paradigm (MIP) introduced by Kong and Schoenebeck (Kong and Schoenebeck 2019). Traditional peer prediction approaches typically rely on scoring rules and correlation between agents' signals. However, these methods often struggle with issues like uninformed equilibria, where agents can coordinate on uninformative strategies that yield higher payoffs than truth-telling. The core idea is to reward agents based on the mutual information between their report and the reports of other agents.

We consider a setting with n agents, each possessing a private signal Ψ_i drawn from some set Σ. The mechanism asks each agent to report their signal, which we denote as $\hat{\Psi}_i$. For each agent i, the mechanism randomly selects a reference agent $j \neq i$. Agent i's payment is then calculated as:

$$MI(\hat{\Psi}_i; \hat{\Psi}_j) \tag{3.68}$$

where MI is an information-monotone mutual information measure. An information-monotone MI measure must satisfy the following properties:

- **Symmetry**: $MI(X;Y) = MI(Y;X)$.
- **Non-negativity**: $MI(X;Y) \geq 0$, with equality if and only if X and Y are independent.
- **Data processing inequality**: For any transition probability M, if Y is independent of $M(X)$ conditioned on X, then $MI(M(X);Y) \leq MI(X;Y)$.

Two important families of mutual information measures that satisfy these properties are f-mutual information and Bregman mutual information. The f-mutual information is defined as:

$$MI_f(X;Y) = D_f(U_{X,Y}, V_{X,Y}) \tag{3.69}$$

where D_f is an f-divergence, $U_{X,Y}$ is the joint distribution of X and Y, and $V_{X,Y}$ is the product of their marginal distributions. The Bregman mutual information is defined as:

$$BMI_{PS}(X;Y) = \mathbb{E}_X[DPS(U_{Y|X}, U_Y)] \tag{3.70}$$

where D_{PS} is a Bregman divergence based on a proper scoring rule PS, $U_{Y|X}$ is the conditional distribution of Y given X, and U_Y is the marginal distribution of Y.

The MIP framework can be applied in both single-question and multi-question settings. In the multi-question setting, the mechanism can estimate the mutual information empirically from multiple questions. In the single-question setting, additional techniques like asking for predictions about other agents' reports are used to estimate the mutual information.

A key theoretical result of the MIP framework is that when the chosen mutual information measure is strictly information-monotone with respect to agents' priors, the resulting mechanism is both dominantly truthful and strongly truthful. This means that truth-telling is a dominant strategy for each agent and that the truth-telling equilibrium yields strictly higher payoffs than any other non-permutation strategy profile.

As research continues to address practical implementation challenges of designing truthful mechanisms, MIP-based approaches have significant potential to improve preference elicitation and aggregation in real-world applications lacking verifiable ground truth.

3.2.13.2 Auction Theory 2

Single-Item Auctions

The first problem within auction we will consider is the *single-item auction*. In this problem setup, there is a single item to sell and n bidders each with unknown private valuations of the item $v_1, \dots, v_n$,

3.3 Exercises

Question 1: Uncertainty Quantification in Preference Learning (40 points)

In this question, we will explore Bayesian approaches to logistic regression in the context of preference learning using the Bradley-Terry model. We will compare different models and inference methods, including parametric linear models estimated using Metropolis-Hastings, parametric neural network models estimated using Hamiltonian Monte Carlo, and non-parametric models with Gaussian Processes. Finally, we will assess the uncertainty quantification in these models using the Expected Calibration Error (ECE).

Assume we have a dataset of pairwise preferences $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^d$ represents the feature difference between two items (i.e., $x_i = e_1^{(i)} - e_2^{(i)}$ for embeddings $e_1^{(i)}$ and $e_2^{(i)}$), and $y_i \in \{0, 1\}$ indicates the preference ($y_i = 1$ if item 1 is preferred over item 2 in the i-th pair).

The likelihood of observing y_i given x_i and model parameters θ is given by the logistic function:

$$P(y_i = 1 | x_i, \theta) = \sigma(x_i^\top \theta) = \frac{1}{1 + e^{-x_i^\top \theta}}.$$

We will adopt a Bayesian approach by placing priors on the model parameters and using Markov Chain Monte Carlo (MCMC) methods to estimate the posterior distributions.

(a) **Uncertainty Quantification and Expected Calibration Error (11 points)**

(i) **(Written, 2 point).** Spend some time reading https://tinyurl.com/m77mk9c. Explain what the Expected Calibration Error (ECE) measures and why it is important for assessing uncertainty quantification in probabilistic models.

(ii) **(Coding, 6 points).** In `uncertainty_quantification/ece.py`, implement the ECE using the formula

$$\text{ECE} = \sum_{k=1}^{K} \frac{n_k}{N} \left|\text{acc}(B_k) - \text{conf}(B_k)\right|,$$

where n_k is the number of samples in bin B_k, N is the total number of samples, $\text{acc}(B_k)$ is the accuracy in bin B_k, and $\text{conf}(B_k)$ is the average confidence in bin B_k.

(iii) **(Written, 3 point).** After doing parts (b), (c), and (d), compare the ECE scores and reliability diagrams of the 3 models. Which model(s) provide the best uncertainty quantification? Discuss possible reasons for the observed differences.

```python
import numpy as np
import matplotlib.pyplot as plt

def expected_calibration_error(probs, labels, model_name, n_bins=20, n_ticks=10, plot=True):
    """
    Computes the Expected Calibration Error (ECE) for a model and plots a refined reliability diagram
    with confidence histogram and additional calibration statistics.

    Args:
    - probs (np.array): Array of predicted probabilities for the positive class (for binary classification).
    - labels (np.array): Array of true labels (0 or 1).
    - model_name (str): Name of the model for labeling the plot.
    - n_bins (int): Number of bins to divide the probability interval [0,1] into.
    - n_ticks (int): Number of ticks to show along the x-axis.
    - plot (bool): If True, generates the reliability plot; otherwise, only computes ECE.

    Returns:
    - float: Computed ECE value.
    """

    # Ensure probabilities are in the range [0, 1]
    assert np.all((probs >= 0) & (probs <= 1)), "Probabilities must be in the range [0, 1]"

    # Initialize bin edges, centers, and storage for accuracy, confidence, and counts
```

```
    bin_edges = np.linspace(0, 1, n_bins + 1)
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
    bar_width = 1.0 / n_bins

    accs = np.zeros(n_bins)
    confs = np.zeros(n_bins)
    bin_counts = np.zeros(n_bins)

    # Populate bin statistics: accuracy, confidence, and count
    # YOUR CODE HERE (~7 lines)
    # Loop over each bin and:
    # - Find indices of probabilities that fall within the bin.
    # - Count the number of items in the bin.
    # - Calculate the accuracy (average of true labels) within the bin.
    # - Calculate the confidence (average of predicted probabilities) within the bin.
    pass
    # END OF YOUR CODE

    # Compute ECE: weighted average of |accuracy - confidence| across bins
    # YOUR CODE HERE (1 line)
    # - Use the bin counts to calculate a weighted average of the differences between accuracy and confidence.
    ece_value = None
    # END OF YOUR CODE

    # Return only ECE if plot is not required
    if not plot:
        return ece_value

    # Compute average confidence and accuracy for reference lines
    avg_confidence = np.mean(probs)
    avg_accuracy = np.mean(labels)

    # Create reliability diagram and histogram
    fig, (ax1, ax2) = plt.subplots(2, 1, gridspec_kw={'height_ratios': [3, 1]}, figsize=(8, 10))

    # Reliability diagram (top plot)
    ax1.plot([0, 1], [0, 1], 'k--', label='Perfect Calibration')
    for i in range(n_bins):
        # Draw the gap bar starting from the diagonal line (perfect calibration)
        ax1.bar(bin_centers[i], abs(accs[i] - confs[i]), width=bar_width, bottom=min(accs[i], confs[i]),
                color='red', alpha=0.3, label='Accuracy-Confidence Gap' if i == 0 else "")
        # Draw the accuracy bar as a small black line on top of the gap bar
        ax1.plot([bin_centers[i] - bar_width / 2, bin_centers[i] + bar_width / 2],
                 [accs[i], accs[i]], color='black', linewidth=2)
```

```
    # Add a black line as a sample for accuracy in the legend
    ax1.plot([], [], color='black', linewidth=2, label='Accuracy Marker')

    ax1.set_xlim(0, 1)
    ax1.set_ylim(0, 1)
    ax1.set_ylabel('Accuracy')
    ax1.set_title(f'{model_name}\nECE={ece_value:.2f}')
    ax1.legend()

    # Set tick marks based on `n_ticks` evenly spaced along the x-axis
    tick_positions = np.linspace(0, 1, n_ticks + 1)
    ax1.set_xticks(tick_positions)
    ax2.set_xticks(tick_positions)
    ax1.set_xticklabels([f'{x:.2f}' for x in tick_positions])
    ax2.set_xticklabels([f'{x:.2f}' for x in tick_positions])

    # Confidence histogram with average markers
    ax2.bar(bin_centers, bin_counts, width=bar_width, color='blue', alpha=0.6)
    ax2.axvline(x=avg_confidence, color='gray', linestyle='--', linewidth=2,
    ↪ label='Avg. confidence')
    ax2.axvline(x=avg_accuracy, color='black', linestyle='-', linewidth=2, label='Avg.
    ↪ accuracy')
    ax2.set_xlim(0, 1)
    ax2.set_xlabel('Confidence')
    ax2.set_ylabel('Count')
    ax2.legend()

    plt.tight_layout()
    plt.show()

    return ece_value

if __name__ == "__main__":
    # Test with random probabilities and labels
    probs = np.random.rand(10000)  # Random probabilities between 0 and 1
    labels = np.random.binomial(1, (probs + 1) / 2)

    # Run the function and display the result
    ece_value = expected_calibration_error(probs, labels, "Test Model", plot=True)
    print(f"ECE Value: {ece_value}")
```

(b) **Parametric Linear Model Estimated Using Metropolis-Hastings (11 points)**

(i) **(Written, 3 points).** Assume a prior on θ such that $\theta \sim \mathcal{N}(0, \sigma^2 I)$, where σ^2 is the variance and I is the identity matrix. Derive the expression for the posterior distribution $P(\theta|\mathcal{D})$ up to a normalization constant.

(ii) **(Coding, 6 points).** Implement the Metropolis-Hastings algorithm to sample from

the posterior distribution of θ in `uncertainty_quantification/metropolis.py`.

(iii) **(Written, 2 points).** Discuss how you chose the proposal variance τ^2 and the number of iterations T and $T_{\text{burn-in}}$. How did these choices affect the convergence and mixing of your MCMC chain?

```python
import torch
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
from ece import expected_calibration_error

# Load training and testing data
x_train = torch.tensor(np.load('../data/differences_train.npy'))
x_test = torch.tensor(np.load('../data/differences_test.npy'))
y_train = torch.tensor(np.load('../data/labels_train.npy'))
y_test = torch.tensor(np.load('../data/labels_test.npy'))

# Likelihood function for logistic regression (per data point)
def likelihood(theta, x, y):
    """
    Computes the likelihood of the data given the logistic regression parameters.

    Args:
    - theta (torch.Tensor): Model parameters.
    - x (torch.Tensor): Input data.
    - y (torch.Tensor): True labels.

    Returns:
    - torch.Tensor: Likelihood values for each data point.
    """
    # YOUR CODE HERE (~3 lines)
    # Calculate logits as the linear combination of inputs and parameters.
    # Use the sigmoid function to compute the probability of the positive class.
    pass
    # END OF YOUR CODE

# Prior probability (theta ~ N(0, I)) - only depends on theta, not per sample
def prior(theta, sigma):
    """
    Computes the prior probability of theta under a Gaussian distribution with
    ↪ variance sigma^2.

    Args:
    - theta (torch.Tensor): Model parameters.
    - sigma (float): Standard deviation of the prior distribution.

    Returns:
```

```
    - torch.Tensor: Prior probability value.
    """
    # YOUR CODE HERE (~2 lines)
    # Implement Gaussian prior with zero mean and identity covariance.
    # Note that the normalization constant is not needed for Metropolis-Hastings.
    pass
    # END OF YOUR CODE

# Metropolis-Hastings sampler
def metropolis_hastings(x, y, num_samples, burn_in, tau, sigma):
    """
    Runs the Metropolis-Hastings algorithm to sample from the posterior distribution.

    Args:
    - x (torch.Tensor): Input data.
    - y (torch.Tensor): True labels.
    - num_samples (int): Total number of samples to draw.
    - burn_in (int): Number of initial samples to discard.
    - tau (float): Proposal standard deviation.
    - sigma (float): Prior standard deviation.

    Returns:
    - torch.Tensor: Collected samples post burn-in.
    - float: Acceptance ratio.
    """
    # Initialize theta (starting point of the chain) and containers for samples and acceptance count
    theta = torch.zeros(x.shape[1])
    samples = []
    acceptances = 0

    # Run the Metropolis-Hastings algorithm
    for t in tqdm(range(num_samples), desc="MCMC Iteration"):
        # YOUR CODE HERE (~12-16 lines)
        # 1. Propose new theta from the proposal distribution (e.g., Gaussian around current theta).
        # 2. Compute prior and likelihood for current and proposed theta
        # 3. Calculate the acceptance ratio as the product of likelihood and prior ratios.
        # 4. Accept or reject the proposal based on the acceptance probability.
        # 5. Store the sample after the burn-in period
        pass
        # END OF YOUR CODE

    return torch.stack(samples), acceptances / num_samples

# Run Metropolis-Hastings on training data
num_samples = 10000
```

```
burn_in = 1000
tau = 0.01  # Proposal variance (tune this for convergence)
sigma = 2.0  # Prior variance

# Collect samples and compute acceptance ratio
samples, acceptance_ratio = metropolis_hastings(x_train, y_train,
 ↪  num_samples=num_samples, burn_in=burn_in, tau=tau, sigma=sigma)
averaged_weights = samples.mean(axis=0)
print(f'Predicted weights: {averaged_weights}')
print(f'Acceptance Ratio: {acceptance_ratio}')

# Evaluate accuracy on training set
train_predictions = (x_train @ averaged_weights > 0).float()
train_acc = (train_predictions == y_train).float().mean()
print(f'Train Accuracy: {train_acc}')

# Evaluate accuracy on testing set
test_predictions = (x_test @ averaged_weights > 0).float()
acc = (test_predictions == y_test).float().mean()
print(f'Test Accuracy: {acc}')

# Compute expected calibration error on testing set
expected_calibration_error(torch.sigmoid(x_test @ averaged_weights).numpy(),
 ↪  y_test.numpy(), model_name="Metropolis-Hastings")
```

(c) **Parametric Neural Network Model Estimated Using Hamiltonian Monte Carlo (11 points)**

(i) **(Written, 2 points)**. Explain why Hamiltonian Monte Carlo (HMC) is suitable for sampling from the posterior distribution of neural network parameters compared to Metropolis-Hastings.

(ii) **(Coding, 7 points)**. Implement HMC to sample from the posterior distribution of the parameters θ of a neural network $f(x;\theta)$ used for preference prediction in `uncertainty_quantification/hmc_nn.py`. This will require a GPU and take around 5 minutes on it!

(iii) **(Written, 2 points)**. Briefly describe the performance of the HMC and Metropolis-Hastings models and provide the accuracy numbers.

```
# Use a GPU when running this file! JAX should automatically default to GPU.
import jax.numpy as np
import numpyro
import numpyro.distributions as dist
from numpyro.infer import MCMC, NUTS
from jax import random
from ece import expected_calibration_error
```

```

# DO NOT CHANGE! This function can be ignored.
def set_numpyro(new_sampler):
    numpyro.sample = new_sampler

# Define the neural network model with one hidden layer
def nn_model(x_data, y_data, hidden_dim=10):
    """
    Defines a Bayesian neural network with one hidden layer.

    Args:
    - x_data (np.array): Input data.
    - y_data (np.array): Target labels.
    - hidden_dim (int): Number of units in the hidden layer.

    Returns:
    - hidden_activations: Activations from the hidden layer.
    - logits: Logits for the output layer.
    """
    input_dim = x_data.shape[1]

    # Prior over the weights and biases for the hidden layer
    w_hidden = numpyro.sample('w_hidden', dist.Normal(np.zeros((input_dim, hidden_dim)), np.ones((input_dim, hidden_dim))))
    b_hidden = numpyro.sample('b_hidden', dist.Normal(np.zeros(hidden_dim), np.ones(hidden_dim)))

    # Compute the hidden layer activations using ReLU
    # YOUR CODE HERE (~1 line)
    # Implement the hidden layer computation, applying a ReLU activation.
    pass
    # END OF YOUR CODE

    # Prior over the weights and biases for the output layer
    w_output = numpyro.sample('w_output', dist.Normal(np.zeros(hidden_dim), np.ones(hidden_dim)))
    b_output = numpyro.sample('b_output', dist.Normal(0, 1))

    # Compute the logits for the output layer
    # YOUR CODE HERE (~1 line)
    # Calculate the logits as the linear combination of hidden activations and output layer weights.
    pass
    # END OF YOUR CODE

    # Likelihood (Bernoulli likelihood with logits)
    numpyro.sample('obs', dist.Bernoulli(logits=logits), obs=y_data)
    return hidden_activations, logits
```

```

def sigmoid(x):
    """Helper function to compute the sigmoid of x."""
    return 1 / (1 + np.exp(-x))

if __name__ == "__main__":
    # Load training and testing data
    x_train = np.load('../data/differences_train.npy')
    x_test = np.load('../data/differences_test.npy')
    y_train = np.load('../data/labels_train.npy')
    y_test = np.load('../data/labels_test.npy')

    # HMC Sampler Configuration
    hmc_kernel = NUTS(nn_model)

    # Running HMC with the MCMC interface in NumPyro
    num_samples = 200  # Number of samples
    warmup_steps = 100  # Number of burn-in steps
    rng_key = random.PRNGKey(0)  # Random seed

    # MCMC object with HMC kernel
    mcmc = MCMC(hmc_kernel, num_samples=num_samples, num_warmup=warmup_steps)
    mcmc.run(rng_key, x_train, y_train)

    # Get the sampled weights (theta samples)
    samples = mcmc.get_samples()

    # Extract the weight samples
    w_hidden_samples = samples['w_hidden']
    b_hidden_samples = samples['b_hidden']
    w_output_samples = samples['w_output']
    b_output_samples = samples['b_output']

    # Compute the averaged weights and biases
    w_hidden_mean = np.mean(w_hidden_samples, axis=0)
    b_hidden_mean = np.mean(b_hidden_samples, axis=0)
    w_output_mean = np.mean(w_output_samples, axis=0)
    b_output_mean = np.mean(b_output_samples, axis=0)

    # Forward pass through the network for testing set
    # YOUR CODE HERE (~2 lines)
    # Compute hidden layer activations and logits for the test set using the mean
    ↪  weights and biases.
    pass
    # END OF YOUR CODE
    test_predictions = test_logits > 0
    test_accuracy = np.mean(test_predictions == y_test)
    print(f'Test Accuracy: {test_accuracy}')
```

```
    # Forward pass through the network for training set
    # YOUR CODE HERE (~2 lines)
    # Compute hidden layer activations and logits for the training set.
    pass
    # END OF YOUR CODE
    train_predictions = train_logits > 0
    train_accuracy = np.mean(train_predictions == y_train)
    print(f'Train Accuracy: {train_accuracy}')

    # Compute expected calibration error on testing set
    expected_calibration_error(sigmoid(test_logits), y_test, model_name="HMC")
```

(d) **Non-Parametric Model with Gaussian Process (GP) (7 points)**

(i) **(Written, 2 point)**. Describe how a Gaussian Process can be used for preference learning in this context (i.e., describe how the latent function is used for classification).

(ii) **(Coding, 2 points)**. Run the GP classification for preference learning code in `uncertainty_quantification/gaussian_process.py` and provide the accuracy numbers. This can only be run on a CPU and may take around 10 minutes to complete.

(iii) **(Written, 3 point)**. Discuss the computational complexity of the GP model compared to the parametric models. What are the advantages and disadvantages of using a GP in this setting?

```
import numpy as np
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.metrics import accuracy_score
from ece import expected_calibration_error

x_train = np.load('../data/differences_train.npy')
x_test = np.load('../data/differences_test.npy')
y_train = np.load('../data/labels_train.npy')
y_test = np.load('../data/labels_test.npy')

kernel = 1.0 * RBF(length_scale=1.0)
gp_classifier = GaussianProcessClassifier(kernel=kernel, random_state=42, n_jobs=-1)
gp_classifier.fit(x_train, y_train)

y_pred_probs = gp_classifier.predict_proba(x_test)[:, 1]
y_pred_labels = (y_pred_probs > 0.5)

train_accuracy = accuracy_score(y_train, gp_classifier.predict(x_train))
```

```
print(f'Train Accuracy: {train_accuracy:.4f}')

test_accuracy = accuracy_score(y_test, y_pred_labels)
print(f'Test Accuracy: {test_accuracy:.4f}')

expected_calibration_error(y_pred_probs, y_test, model_name="Gaussian Process
↪ Classifier")
```

Question 2: Active Learning for Preference Learning (40 points)

In this question, you will explore active learning strategies for preference learning using a linear model. We will use expected information gain as the acquisition function to select the most informative queries, where each query is a pair of items. Assume that we model the preferences using a simple linear model. Given feature vectors x_1 and x_2 corresponding to two items, the probability that x_1 is preferred over x_2 is modeled using a logistic regression model, i.e.,

$$P(x_1 \succ x_2|\theta) = \sigma(\theta^\top (x_1 - x_2)),$$

where $\theta \in \mathbb{R}^d$ is the model parameter vector, and $\sigma(z)$ is the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. The goal is to sequentially select pairs of items to maximize the information gained about θ through preference queries.

(a) **Expected Information Gain (15 points)**

(i) **Derive the Expected Information Gain (Written, 3 points)**. Suppose that after observing a preference between two items x_1 and x_2, the posterior distribution over θ is updated. The information gain from this observation is the reduction in uncertainty about θ measured using the Kullback-Leibler (KL) divergence between the prior and posterior distributions. Given the current posterior distribution $P(\theta|\mathcal{D})$ and a possible observation $y \in \{0, 1\}$ (where $y = 1$ if x_1 is preferred over x_2, and $y = 0$ otherwise), the expected information gain is:

$$\begin{aligned}\mathbb{E}[\text{IG}(x_1, x_2)] =& P(y = 1|x_1, x_2, \theta) D_{\text{KL}}\left(P(\theta|y = 1, \mathcal{D}) \parallel P(\theta|\mathcal{D})\right) \\ &+ P(y = 0|x_1, x_2, \theta) D_{\text{KL}}\left(P(\theta|y = 0, \mathcal{D}) \parallel P(\theta|\mathcal{D})\right)\end{aligned}$$

Derive this expression for the expected information gain of selecting the pair (x_1, x_2) for a preference query. Start by explaining how the KL divergence measures the information gain, and break down the expectation over the possible outcomes of the query.

(ii) **Simplifying the KL Divergence (Written, 4 points)**. Assuming the prior and posterior distributions over θ are Gaussian (i.e., $P(\theta) \sim \mathcal{N}(\mu, \Sigma)$ and $P(\theta|\mathcal{D}) \sim$

$\mathcal{N}(\mu', \Sigma')$), show that the KL divergence between the Gaussian posterior and prior simplifies to:

$$D_{\mathrm{KL}}\left(\mathcal{N}(\mu', \Sigma') \parallel \mathcal{N}(\mu, \Sigma)\right) = \frac{1}{2}\left(\mathrm{tr}(\Sigma^{-1}\Sigma') + (\mu' - \mu)^\top \Sigma^{-1}(\mu' - \mu) - d + \log\left(\frac{\det(\Sigma)}{\det(\Sigma')}\right)\right).$$

(iii) **Approximate Information Gain for a Linear Model (Written, 4 points)**. In the case of a linear model with Gaussian priors on θ, assume that the posterior distribution $P(\theta|\mathcal{D}) \sim \mathcal{N}(\mu, \Sigma)$ is updated using Bayes' rule after each observation. The likelihood of observing a preference y is logistic, which does not conjugate with the Gaussian prior. However, for the purposes of this question, assume that after each query, the posterior mean μ' and covariance Σ' can be updated using an approximation method such as Laplace's approximation.

Using these assumptions, compute the expected information gain for a specific query (x_1, x_2) in closed form. You may express the information gain in terms of the updated mean μ' and covariance Σ' after observing the preference outcome.

(iv) **Laplace Approximation for Posterior (Written, 4 points)**. The Laplace approximation for the posterior is given by

$$\mu' = \arg\min_\theta -\log P(\theta|\mathcal{D})$$
$$\Sigma'^{-1} = \nabla_\theta \nabla_\theta - \log P(\theta|\mathcal{D})|_{\theta=\mu'}$$

In our scenario with the Bradley-Terry model for likelihood, simplify $-\log P(\theta|\mathcal{D})$ and its Hessian ignoring the normalization constant.

(b) **Active Learning Algorithm (25 points)** In this section, you will implement an active learning algorithm for selecting the most informative queries using the expected information gain criterion.

(i) **(Coding, 4 points)**. Implement `kl_divergence_gaussians` in `active_learning/main.py`.

(ii) **(Coding, 4 points)**. Following your derived Laplace approximation, implement `negative_log_posterior`.

(iii) **(Coding, 4 points)**. Implement `compute_hessian` that is used to obtain the inverse of the covariance matrix.

(iv) **(Coding, 3 points)**. Implement `expected_information_gain`.

(v) **(Coding, 4 points)**. Finally, implement `active_learning`.

(vi) **(Coding + Written, 6 points)**. Plot the L^2 norm of the covariance matrix for each loop of the active learning loop. Additionally, on the same plot, implement a random baseline and plot its L^2 covariance matrix norm. The random baseline should randomly select a point in the dataset and not use any acquisition function. Interpret your plot and use it to compare the two methods.

```
import torch
import torch.nn.functional as F
from torch.optim import Adam
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

class LogisticActiveLearning:
    def __init__(self, test_size=0.2):
        """
        Initializes LogisticActiveLearning model, sets device, and prepares data.

        Args:
        - test_size (float): Proportion of the dataset used for validation.
        """
        # Make device customizable
        self.device = torch.device("cpu")
        X, y = make_classification(n_samples=10000, random_state=42)

        # Convert data and labels to tensors
        x_data = torch.tensor(X, dtype=torch.float32).to(self.device)
        y_data = torch.tensor(y, dtype=torch.float32).to(self.device)
        self.N, self.D = x_data.shape

        # Split into training and validation sets
        train_indices, val_indices = train_test_split(range(self.N),
↪   test_size=test_size, random_state=42)
        self.x_train = x_data[train_indices]
        self.y_train = y_data[train_indices]
        self.x_val = x_data[val_indices]
        self.y_val = y_data[val_indices]

        # Initialize mean and inverse covariance for the prior
        self.weights_mean = torch.zeros(self.D, requires_grad=True,
         ↪  device=self.device)
        self.weights_inv_cov = torch.eye(self.D).to(self.device)  # Start with
         ↪  identity inverse covariance

    def negative_log_posterior(self, w, x, y):
        """
        Computes the negative log-posterior (negative log-prior + log-likelihood).

```

```
        Args:
        - w (torch.Tensor): Model weights.
        - x (torch.Tensor): Input data point.
        - y (torch.Tensor): True label.

        Returns:
        - torch.Tensor: Negative log-posterior value.
        """
        # YOUR CODE HERE (~4-6 lines)
        # Compute log-prior term using inverse covariance
        pass
        # END OF YOUR CODE

    def optimize_weights(self, w, x, y, num_steps=50, lr=1e-2):
        """
        Optimizes weights using Adam optimizer.

        Args:
        - w (torch.Tensor): Initial weights.
        - x (torch.Tensor): Input data point.
        - y (torch.Tensor): True label.
        - num_steps (int): Number of optimization steps.
        - lr (float): Learning rate.

        Returns:
        - torch.Tensor: Updated weights.
        - torch.Tensor: Hessian inverse covariance.
        """
        optimizer = Adam([w], lr=lr)

        for step in range(num_steps):
            optimizer.zero_grad()
            loss = self.negative_log_posterior(w, x, y)
            loss.backward()
            optimizer.step()

        # Compute the Hessian of log-posterior, serving as inverse covariance
        inv_cov = self.compute_hessian(w.detach(), x, y)
        return w.detach().clone(), inv_cov

    def compute_hessian(self, w, x, y):
        """
        Computes the Hessian of the negative log-posterior, used as the inverse
    ↪ covariance.

        Args:
        - w (torch.Tensor): Model weights.
        - x (torch.Tensor): Input data point.
```

```
        - y (torch.Tensor): True label.

        Returns:
        - torch.Tensor: Hessian of the negative log-posterior.
        """
        # YOUR CODE HERE (~5-8 lines)
        # Hessian of the prior term
        pass
        # END OF YOUR CODE

    def acquisition_fn(self, x):
        """
        Computes posterior means and inverse covariances for y=1 and y=0 without modifying original parameters.

        Args:
        - x (torch.Tensor): Input data point.

        Returns:
        - dict: Posterior properties for y=1 and y=0 cases.
        """
        weights_y1 = self.weights_mean.clone().detach().requires_grad_(True)
        weights_y0 = self.weights_mean.clone().detach().requires_grad_(True)

        # Optimize weights and get Hessian for both y=1 and y=0 cases
        posterior_mean_y1, inv_cov_y1 = self.optimize_weights(weights_y1, x, 1, num_steps=50)
        posterior_mean_y0, inv_cov_y0 = self.optimize_weights(weights_y0, x, 0, num_steps=50)

        # Calculate probabilities for the acquisition function
        prob_y1 = torch.sigmoid(torch.dot(self.weights_mean.detach(), x))
        prob_y0 = 1 - prob_y1

        return {
            'prob_y1': prob_y1,
            'prob_y0': prob_y0,
            'posterior_mean_y1': posterior_mean_y1,
            'posterior_inv_cov_y1': inv_cov_y1,
            'posterior_mean_y0': posterior_mean_y0,
            'posterior_inv_cov_y0': inv_cov_y0
        }

    def expected_information_gain(self, x):
        """
        Computes expected information gain for a given point `x`.

        Args:
```

```
        - x (torch.Tensor): Input data point.

        Returns:
        - torch.Tensor: Expected Information Gain (EIG) value.
        """
        acquisition = self.acquisition_fn(x)

        # Compute KL divergences for y=1 and y=0 using inverse covariances
        kl_y1 = kl_divergence_gaussians(
            acquisition['posterior_mean_y1'],
            acquisition['posterior_inv_cov_y1'],
            self.weights_mean.detach(),
            self.weights_inv_cov
        )

        kl_y0 = kl_divergence_gaussians(
            acquisition['posterior_mean_y0'],
            acquisition['posterior_inv_cov_y0'],
            self.weights_mean.detach(),
            self.weights_inv_cov
        )

        # Expected Information Gain (EIG)
        eig = None # YOUR CODE HERE (1 line)
        return eig

    def active_learning(self, selected_indices, subset_size=50):
        """
        Active learning loop that selects the most informative data point based on
↪   EIG.

        Args:
        - selected_indices (list): Indices of previously selected samples.
        - subset_size (int): Number of samples to consider in each subset.

        Returns:
        - best_x, best_x_idx, best_acquisition: Selected data point and acquisition
↪   details.
        """
        best_eig = -float('inf')
        best_x = None
        best_x_idx = -1
        best_acquisition = None

        subset_indices = [i for i in torch.randperm(len(self.x_train)).tolist() if i
↪   not in selected_indices][:subset_size]

        # YOUR CODE HERE (~ 10 lines)
```

```
        pass
        # END OF YOUR CODE
        return best_x, best_x_idx, best_acquisition

    def validate(self):
        """
        Computes accuracy on the validation set by predicting labels and comparing to true labels.

        Returns:
        - float: Validation accuracy.
        """
        with torch.no_grad():
            logits = self.x_val @ self.weights_mean
            predictions = torch.sigmoid(logits) >= 0.5  # Convert logits to binary predictions
            accuracy = (predictions == self.y_val).float().mean().item()
            print(f"Validation accuracy: {accuracy * 100:.2f}%")
        return accuracy

    def train(self, num_iterations=10, subset_size=50):
        """
        Train the model using active learning with subset sampling.

        Args:
        - num_iterations (int): Number of active learning iterations.
        - subset_size (int): Number of samples to consider in each subset.
        """
        selected_indices = []
        for iteration in range(num_iterations):
            print(f"Iteration {iteration + 1}/{num_iterations}")

            # Select the most informative data point from a random subset
            best_x, best_x_idx, acquisition = self.active_learning(selected_indices, subset_size=subset_size)
            selected_indices.append(best_x_idx)
            print(f"Selected data point with EIG.")

            # Get the true label for the selected data point
            y = self.y_train[best_x_idx].item()

            # Update posterior mean and inverse covariance based on true label
            if y == 1:
                self.weights_mean = acquisition['posterior_mean_y1']
                self.weights_inv_cov = acquisition['posterior_inv_cov_y1']
            else:
                self.weights_mean = acquisition['posterior_mean_y0']
                self.weights_inv_cov = acquisition['posterior_inv_cov_y0']
```

```
            print(f"Covariance L2: {torch.inverse(self.weights_inv_cov).norm()}")

            # Validate model performance on the validation set
            self.validate()

# KL divergence between two multivariate normal distributions
def kl_divergence_gaussians(mu1, sigma1_inv, mu2, sigma2_inv):
    """
    Computes the KL divergence between two multivariate Gaussian distributions.

    Args:
    - mu1, mu2 (torch.Tensor): Mean vectors of the distributions.
    - sigma1_inv, sigma2_inv (torch.Tensor): Inverse covariance matrices of the
    ↪ distributions. PLEASE NOTE THE INVERSE!

    Returns:
    - torch.Tensor: KL divergence value.
    """
    # YOUR CODE HERE (~ 9-12 lines)
    pass
    # END OF YOUR CODE

# Example usage
model = LogisticActiveLearning()
model.train(num_iterations=100, subset_size=50)
```

Question 3: Linear Performance Metric Elicitation (30 points)

1. **(Written, 10 points)**. For background on the problem setting, read https://tinyurl.com/3b92sufm. Suppose we have a linear performance metric given by

$$p(C) = 1 - \alpha(FP) - \beta(FN)$$

where C is a confusion matrix and FP, FN denote false positive and false negative rates. We wish to find the optimal classifier w.r.t. p. That is,

$$\phi^* = \arg\max_{\phi \in \Phi} p(C(\phi))$$

where Φ is the space of all probabilistic binary classifiers from $X \to [0, 1]$. Note that these classifiers return probabilities corresponding to the label 1. Show that ϕ^* is in fact deterministic and given by

$$\phi(x) = \begin{cases} 1 & \text{if } p(y|x) > f(\alpha, \beta) \\ 0 & \text{otherwise.} \end{cases}$$

for a threshold function f that you must find. (Hint: For a classifier ϕ, $FP = P(\phi = 1, y = 0)$ and $FN = P(\phi = 0, y = 1)$. Marginalize these joint probabilities over x and simplify.)

2. **(Written + Coding, 5 points)**. Implement `classifier_metrics` in `lpme/main.py`. After doing so, run `plot_confusion_region` and attach the plot. What do you notice about the region of possible confusion matrices?

3. **(Coding, 15 points)**. Implement `search_theta` in order to elicit the metric used by the oracle (which is parametrized by θ). Play around with the oracle's theta and run `start_search` to see how close you can approximate it!

```python
import torch
import matplotlib.pyplot as plt
from tqdm import tqdm

class DataDistribution:
    def __init__(self, N: int):
        """
        Initializes the data distribution with a specified number of samples.

        Args:
        - N (int): Number of data points.
        """
        self.weights = torch.tensor([-0.3356, -1.4104, 0.3144, -0.5591, 1.0426,
         ↪  0.6036, -0.7549, -1.1909, 1.4779, -0.7513])
        self.D = len(self.weights)

        gen = torch.Generator().manual_seed(42)
        self.data = torch.randn(N, self.D, generator=gen)
        self.probs = torch.sigmoid(self.data @ self.weights)

def classifier_metrics(data_dist, threshold, upper=True):
    """
    Computes the True Positive and True Negative rates based on a classifier
 ↪  threshold.

    Args:
    - data_dist (DataDistribution): The data distribution instance.
    - threshold (float): Threshold value for classification.
    - upper (bool): If True, classifies as positive if above threshold; else, if
 ↪  below.

    Returns:
    - tuple (float, float): True Positive Rate (TP) and True Negative Rate (TN) in
 ↪  that order.
    """
    # YOUR CODE HERE (~3-5 lines)
```

```
    pass
    # END OF YOUR CODE

def sweep_classifiers(data_dist: DataDistribution):
    """
    Sweeps through classifier thresholds and calculates True Positive and True
    ↪ Negative rates.

    Args:
    - data_dist (DataDistribution): The data distribution instance.

    Returns:
    - tuple: Upper and lower boundary data for True Positive and True Negative rates.
    """
    thresholds = torch.linspace(0, 1, 100)
    upper_boundary = []
    lower_boundary = []

    for threshold in tqdm(thresholds, desc="Thresholds"):
        tp_upper, tn_upper = classifier_metrics(data_dist, threshold, upper=True)
        upper_boundary.append((tp_upper, tn_upper))

        tp_lower, tn_lower = classifier_metrics(data_dist, threshold, upper=False)
        lower_boundary.append((tp_lower, tn_lower))

    return upper_boundary, lower_boundary

class Oracle:
    def __init__(self, theta: float):
        """
        Initializes the oracle with a given theta for preference evaluation.

        Args:
        - theta (float): Oracle angle in radians.
        """
        self.theta = torch.tensor(theta)

    def evaluate_lpm(self, tp, tn):
        """
        Computes the linear performance metric (LPM) based on theta.

        Args:
        - tp (float): True Positive rate.
        - tn (float): True Negative rate.

        Returns:
        - float: Linear performance metric evaluation.
        """
```

```
        return torch.cos(self.theta) * tp + torch.sin(self.theta) * tn

    def preferred_classifier(self, tp_1, tn_1, tp_2, tn_2):
        """
        Determines the preferred classifier based on LPM values.

        Args:
        - tp_1, tn_1, tp_2, tn_2 (float): True Positive and True Negative rates for two classifiers.

        Returns:
        - bool: True if first classifier is preferred, False otherwise.
        """
        lpm_1 = self.evaluate_lpm(tp_1, tn_1)
        lpm_2 = self.evaluate_lpm(tp_2, tn_2)
        return (lpm_1 > lpm_2).item()

def theta_to_threshold(theta):
    """Converts theta angle to classification threshold."""
    return 1 / (1 + torch.tan(theta) ** -1)

def search_theta(oracle: Oracle, data_dist, lower_bound, upper_bound):
    """
    Performs a search over theta values to optimize the classification threshold.

    Args:
    - oracle (Oracle): The oracle for LPM evaluation.
    - data_dist (DataDistribution): The data distribution instance.
    - lower_bound (float): Lower bound for theta.
    - upper_bound (float): Upper bound for theta.

    Returns:
    - tuple: Updated lower and upper bounds for theta.
    """
    left = 0.75 * lower_bound + 0.25 * upper_bound
    middle = 0.5 * lower_bound + 0.5 * upper_bound
    right = 0.25 * lower_bound + 0.75 * upper_bound

    thetas = [lower_bound, left, middle, right, upper_bound]
    thresholds = theta_to_threshold(torch.tensor(thetas))
    new_lower, new_upper = None, None

    # YOUR CODE HERE (~18-25 lines)
    # 1. Collect metrics for each threshold value.
    # 2. Determine if LPM increases as theta increases.
    # 3. Check for pattern of increases and decreases in LPM.
    # 4. Update bounds based on observed LPM patterns.
    pass
```

```
    # END OF YOUR CODE

    return new_lower, new_upper

# Create instance and get upper & lower boundary data
data_dist = DataDistribution(N=10000000)
oracle = Oracle(theta=0.1)

def plot_confusion_region():
    """
    Plots the True Positive vs. True Negative rates for the upper and lower classifier boundaries.
    """
    upper_boundary, lower_boundary = sweep_classifiers(data_dist)

    # Prepare data for plotting for upper and lower boundaries
    tp_upper, tn_upper = zip(*upper_boundary)
    tp_lower, tn_lower = zip(*lower_boundary)

    # Plot the results for upper boundary
    plt.figure(figsize=(8, 6))
    plt.plot(tp_upper, tn_upper, marker='o', linestyle='-', alpha=0.7, label="Upper Boundary")
    plt.plot(tp_lower, tn_lower, marker='o', linestyle='--', alpha=0.7, label="Lower Boundary")
    plt.title("True Positive vs. True Negative Rates (Upper & Lower Boundaries)")
    plt.xlabel("True Positive Rate (TP)")
    plt.ylabel("True Negative Rate (TN)")
    plt.legend()
    plt.grid(True)
    plt.show()

def start_search():
    """
    Starts the theta search using the LPM-based oracle and prints the search range per iteration.
    """
    lower_bound = 0
    upper_bound = torch.pi / 2
    for _ in tqdm(range(10), desc="LPM Search"):
        print(f"Theta Search Space: [{lower_bound}, {upper_bound}]")
        lower_bound, upper_bound = search_theta(oracle, data_dist, lower_bound=lower_bound, upper_bound=upper_bound)
    print(f"Theta Search Space: [{lower_bound}, {upper_bound}]")
```

Question 4: D-optimal Design with Logistic Model (30 points)

In this question, we explore D-optimal designs in the context of the Bradley-Terry model. The Bradley-Terry model is a logistic regression model used for paired comparison data. Given two items x_1 and x_2, the probability that item x_1 is preferred over x_2 is modeled as:

$$P(x_1 \succ x_2|\theta) = \frac{e^{\theta^\top x_1}}{e^{\theta^\top x_1} + e^{\theta^\top x_2}} = \frac{1}{1 + e^{\theta^\top (x_2 - x_1)}}$$

where $\theta \in \mathbb{R}^d$ represents the unknown model parameters, and $x_1, x_2 \in \mathbb{R}^d$ are the feature vectors associated with the two items. D-optimal design aims to maximize the determinant of the Fisher information matrix, thus minimizing the volume of the confidence ellipsoid for the estimated parameters. In this exercise, you will analyze D-optimal designs for this model.

(a) **Fisher Information Matrix for the Bradley-Terry Model (12 points)**

(i) **(Written, 6 points)**. Derive the Fisher information matrix for the Bradley-Terry model at a design point (x_1, x_2). Show that the Fisher information matrix at a design point is:

$$I(x_1, x_2, \theta) = w(x_1, x_2, \theta)(x_1 - x_2)(x_1 - x_2)^\top,$$

where $w(x_1, x_2, \theta)$ is a weight function given by:

$$w(x_1, x_2, \theta) = \frac{e^{\theta^\top x_1} e^{\theta^\top x_2}}{\left(e^{\theta^\top x_1} + e^{\theta^\top x_2}\right)^2} = \sigma'(\theta^\top (x_1 - x_2)).$$

σ' is the derivative of the sigmoid function.

(ii) **(Coding, 6 points)**. Implement `fisher_matrix` in `d_optimal/main.py` based on the derived expression.

(b) **D-optimal Design Criterion (18 points)**

(i) **(Coding, 11 points)**. In the context of the Bradley-Terry model, a D-optimal design maximizes the determinant of the Fisher information matrix. Suppose we have a set of candidate items $\{x_1, \ldots, x_n\}$, and we can choose N comparisons to make. Formally, the D-optimal design maximizes:

$$\det\left(\sum_{i=1}^{N} w(x_{i1}, x_{i2}, \theta)(x_{i1} - x_{i2})(x_{i1} - x_{i2})^\top\right),$$

where (x_{i1}, x_{i2}) denotes a pair of compared items in the design. Implement a greedy algorithm to approximate the D-optimal design. Given a set of n items and their feature vectors $\{x_1, \ldots, x_n\}$, your task is to iteratively select the pair of items (x_{i1}, x_{i2}) that maximizes the determinant of the Fisher information matrix.

Please implement `greedy_fisher`. Note that the setup in the code assumes we have a dataset of all possible differences between pairs of items as opposed to directly selecting the pairs.

(ii) **(Written + Coding, 7 points)**. Notice that `posterior_inv_cov` uses a Laplace approximation for the posterior centered around the ground truth weights after labeling the chosen points. However, it turns out this approximation doesn't actually depend on the labels when taking the Hessian. Please run the file `d_optimal/main.py` and attach a plot of the norm of the covariance matrix of the posterior. What difference do you observe between greedy and random sampling? What is the win rate of greedy?

```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

def sigmoid(x):
    """Helper function to compute the sigmoid of x."""
    return 1 / (1 + np.exp(-x))

class LogisticData:
    def __init__(self, weights, seed=42):
        """
        Initializes the LogisticData class with specified weights and seed.

        Args:
        - weights (np.array): True weights for data generation.
        - seed (int): Random seed for reproducibility.
        """
        self.rng = np.random.default_rng(seed)
        self.weights = weights

    def generate_data(self, N):
        """
        Generates synthetic data for logistic regression.

        Args:
        - N (int): Number of data points.

        Returns:
        - tuple: Generated data and labels.
        """
        data = self.rng.standard_normal((N, len(self.weights)))
        probs = sigmoid(data @ self.weights)
        labels = (self.rng.random(N) < probs).astype(int)
        return data, labels

```

```
def fisher_matrix(difference_vector, weights):
    """
    Computes the Fisher information matrix for a single data point.

    Args:
    - difference_vector (np.array): Difference vector (input data point).
    - weights (np.array): Weights for the logistic model.

    Returns:
    - np.array: Fisher information matrix for the data point.
    """
    # YOUR CODE HERE (~2-4 lines)
    pass
    # END OF YOUR CODE

# Initialization
true_weights = np.array([-0.3356, -1.4104, 0.3144, -0.5591, 1.0426, 0.6036, -0.7549,
 ↪  -1.1909, 1.4779, -0.7513])
data_dim = len(true_weights)
dataset_generator = LogisticData(weights=true_weights)

# Number of iterations for sampling 500 points
num_iterations = 200

# Store covariance matrix norms for comparison
cov_norms_greedy = []
cov_norms_random = []

def greedy_fisher(data, curr_fisher_matrix, selected_indices):
    """
    Selects the data point that maximizes the Fisher information determinant.

    Args:
    - data (np.array): The data matrix.
    - curr_fisher_matrix (np.array): Fisher matrix of already selected indices.
    - selected_indices (list): List of already selected indices.

    Returns:
    - int: Index of the selected data point.
    """
    best_det = -np.inf
    best_index = -1

    # Iterate over data points to find the one maximizing Fisher determinant.
    for i, difference_vector in enumerate(data):
        # YOUR CODE HERE (~5-10 lines)
        # Make sure to skip already selected data points!
        pass
```

```
        # END OF YOUR CODE
    return best_index

def posterior_inv_cov(X, laplace_center):
    """
    Computes the posterior inverse covariance matrix using Laplace approximation.

    Args:
    - X (np.array): Data matrix.
    - laplace_center (np.array): Center point (weights).

    Returns:
    - np.array: Posterior inverse covariance matrix.
    """
    # Calculate probabilities for logistic regression model.
    probs = sigmoid(X @ laplace_center)
    W = np.diag(probs * (1 - probs))

    # Compute inverse covariance matrix assuming standard Gaussian prior.
    inv_cov = X.T @ W @ X + np.eye(len(true_weights))
    return inv_cov

for _ in tqdm(range(num_iterations)):
    # Generate a new sample of 500 data points
    data, _ = dataset_generator.generate_data(N=500)

    # Greedy selection of best 30 data points
    selected_indices = []
    curr_fisher_matrix = np.zeros((data_dim, data_dim))

    for _ in range(30):
        # Select the data point maximizing Fisher information determinant.
        best_index = greedy_fisher(data, curr_fisher_matrix, selected_indices)
        selected_indices.append(best_index)
        curr_fisher_matrix += fisher_matrix(data[best_index], true_weights)

    # Prepare greedy and random samples
    X_greedy = data[selected_indices]

    # Generate 30 random samples for comparison
    random_indices = np.random.choice(len(data), 30, replace=False)
    X_random = data[random_indices]

    # Compute posterior inverse covariance matrices for both strategies
    posterior_inv_cov_greedy = posterior_inv_cov(X_greedy,
    ↪ laplace_center=true_weights)
    posterior_inv_cov_random = posterior_inv_cov(X_random,
    ↪ laplace_center=true_weights)
```

```
129
130     # Calculate covariance matrices (inverse of posterior inverse covariance)
131     cov_matrix_greedy = np.linalg.inv(posterior_inv_cov_greedy)
132     cov_matrix_random = np.linalg.inv(posterior_inv_cov_random)
133
134     # Measure the norm (Frobenius norm) of the covariance matrices
135     cov_norm_greedy = np.linalg.norm(cov_matrix_greedy, 'fro')
136     cov_norm_random = np.linalg.norm(cov_matrix_random, 'fro')
137
138     # Store norms for analysis
139     cov_norms_greedy.append(cov_norm_greedy)
140     cov_norms_random.append(cov_norm_random)
141
142 # Display comparison results
143 print(f'Greedy mean: {np.mean(cov_norms_greedy)}')
144 print(f'Random mean: {np.mean(cov_norms_random)}')
145 print(f'Greedy win rate: {(np.array(cov_norms_greedy) <
    ↪  np.array(cov_norms_random)).mean()}')
146
147 # Plot the distributions of covariance matrix norms
148 plt.hist(cov_norms_greedy, bins=30, alpha=0.7, color='blue', label='Greedy')
149 plt.hist(cov_norms_random, bins=30, alpha=0.7, color='red', label='Random')
150 plt.xlabel('L2 Norm of Covariance Matrix')
151 plt.ylabel('Frequency')
152 plt.title('Comparison of Covariance Norms (Greedy vs. Random) Across Iterations')
153 plt.legend()
154 plt.show()
```

Question 5: Nonparametric Metric Elicitation (30 points)

In this question, we explore the problem of performance metric elicitation using a Gaussian Process (GP) to map the elements of the confusion matrix, specifically false positives (FP) and false negatives (FN), to an unknown performance metric. The goal is to learn a non-linear function that maps FP and FN to the metric, using relative preferences from pairwise classifier comparisons. We will use elliptical slice sampling for posterior inference.

(a) **Gaussian Process for Metric Elicitation (10 points)**

(i) **(Written, 2 points)**. Assume that the performance metric $\phi(C)$ is a non-linear function of the confusion matrix C. For simplicity, assume that ϕ depends only on FP and FN, i.e.,

$$\phi(\text{FP}, \text{FN}) \sim \mathcal{GP}(0, k((\text{FP}, \text{FN}), (\text{FP}', \text{FN}'))),$$

where k is the covariance kernel function of the Gaussian Process. Explain why using a GP allows for flexible modeling of the metric ϕ as a non-linear function of FP and FN. What are the advantages of using a GP over a linear model in this context?

(ii) **(Written, 2 points)**. Suppose we observe pairwise comparisons between classifiers, where a user provides feedback on which classifier they prefer based on the unknown metric ϕ. Given two classifiers with confusion matrices $C_1 = (\text{FP}_1, \text{FN}_1)$ and $C_2 = (\text{FP}_2, \text{FN}_2)$, the user indicates their relative preference. Let the observed preference be modeled by Bradley-Terry as:

$$\Pr(C_1 \succ C_2) = \sigma(\phi(\text{FP}_1, \text{FN}_1) - \phi(\text{FP}_2, \text{FN}_2)).$$

where we view ϕ as the reward function. How does this likelihood affect the posterior inference in the GP? Where does it introduce additional complexity?

(iii) **(Written + Coding, 6 points)**. Given a set of observed pairwise comparisons, derive the posterior distribution over the latent function values ϕ given a set of confusion matrices preferences using Bayes' rule. Express the posterior distribution in terms of the GP prior and the pairwise likelihood function. You do not need to include the normalization constant. Implement the likelihood function in `loglik_from_preferences`.

(b) **Elliptical Slice Sampling for Posterior Inference (20 points)**

(i) **(Written, 3 points)**. Read https://proceedings.mlr.press/v9/murray10a/murray10a.pdf. Elliptical slice sampling is a sampling method used to generate samples from the posterior distribution of a Gaussian Process. Explain the key idea behind elliptical slice sampling and why it is well-suited for sampling from the GP posterior in this context.

(ii) **(Coding, 10 points)**. Implement elliptical slice sampling in `npme/elliptical_sampler.py` by following Figure 2 in the paper.

(iii) **(Written, 3 points)**. Run the algorithm on a synthetic preference dataset of confusion matrices with pairwise preferences. The synthetic data will be constructed using the metric

$$\phi_{\text{true}}(\text{FP}, \text{FN}) = \log(1 + \text{FP}) + \log(1 + \text{FN}),$$

which captures the idea that the human oracle perceives both false positives and false negatives in a way that flattens out as these values increase (i.e., marginal increases in FP and FN have diminishing effects on the performance metric). Explain the psychological motivation behind this non-linear function. Why might a logarithmic form be appropriate for modeling human perception of classification errors?

Run the file `npme/main.py` and attach the plot of ϕ_{true} vs your elicited metric. What do you notice in the plot?

(iv) **(Written + Coding, 4 points)**. Once the GP has been trained and posterior samples of the function $\phi(\text{FP}, \text{FN})$ have been obtained, how can we evaluate the quality of the elicited metric? Propose a method to evaluate how well the

elicited metric ϕ aligns with the user's true preferences and implement it in `evaluate_elicited_metric` taking into the plot you saw in part (iii).

```python
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable
import numpy as np
from tqdm import tqdm

class EllipticalSliceSampler:
    def __init__(self,
                 prior_cov: np.ndarray,
                 loglik: Callable):
        """
        Initializes the Elliptical Slice Sampler.

        Args:
        - prior_cov (np.ndarray): Prior covariance matrix.
        - loglik (Callable): Log-likelihood function.
        """
        self.prior_cov = prior_cov
        self.loglik = loglik

        self._n = prior_cov.shape[0]  # Dimensionality of the space
        self._chol = np.linalg.cholesky(prior_cov)  # Cache Cholesky decomposition

        # Initialize state by sampling from prior
        self._state_f = self._chol @ np.random.randn(self._n)

    def _indiv_sample(self):
        """
        Main algorithm for generating an individual sample using Elliptical Slice Sampling.
        """
        f = self._state_f  # Previous state
        nu = self._chol @ np.random.randn(self._n)  # Sample from prior for the ellipse
        log_y = self.loglik(f) + np.log(np.random.uniform())  # Log-likelihood threshold

        theta = np.random.uniform(0., 2 * np.pi)  # Initial proposal angle
        theta_min, theta_max = theta - 2 * np.pi, theta  # Define bracketing interval

        # Main loop: Accept sample if it meets log-likelihood threshold; otherwise, shrink the bracket.
        while True:
            # YOUR CODE HERE (~10 lines)
            # 1. Generate a new sample point based on the current angle.
```

```
            # 2. Check if the proposed point meets the acceptance criterion.
            # 3. If not accepted, adjust the bracket and select a new angle.
            break
            # END OF YOUR CODE

    def sample(self,
               n_samples: int,
               n_burn: int = 500) -> np.ndarray:
        """
        Generates samples using Elliptical Slice Sampling.

        Args:
        - n_samples (int): Total number of samples to return.
        - n_burn (int): Number of initial samples to discard (burn-in).

        Returns:
        - np.ndarray: Array of samples after burn-in.
        """
        samples = []
        for i in tqdm(range(n_samples), desc="Sampling"):
            self._indiv_sample()
            if i > n_burn:
                samples.append(self._state_f.copy())  # Store sample post burn-in

        return np.stack(samples)

def sigmoid(x):
    """Sigmoid function to map values between 0 and 1."""
    return 1 / (1 + np.exp(-x))

# Step 1: Define a New Two-Dimensional Non-linear Function
def nonlinear_function(x1, x2):
    """
    Computes a non-linear function of x1 and x2.

    Args:
    - x1 (np.array): First input array.
    - x2 (np.array): Second input array.

    Returns:
    - np.array: Computed function values.
    """
    return np.log(1 + x1) + np.log(1 + x2)

# Generate a 2D grid of points
x1 = np.linspace(0, 1, 20)
x2 = np.linspace(0, 1, 20)
```

```
x1_grid, x2_grid = np.meshgrid(x1, x2)
x_grid_points = np.vstack([x1_grid.ravel(), x2_grid.ravel()]).T
f_values = nonlinear_function(x_grid_points[:, 0], x_grid_points[:, 1])

# Step 2: Generate Preferences Using Bradley-Terry Model Over the Grid
def generate_preferences(f_vals, num_prefs=10000):
    """
    Generates preferences based on the Bradley-Terry model.

    Args:
    - f_vals (np.array): Function values at grid points.
    - num_prefs (int): Number of preference pairs to generate.

    Returns:
    - list of tuple: Generated preference pairs (i, j).
    """
    preferences = []
    num_points = len(f_vals)
    for _ in range(num_prefs):
        i, j = np.random.choice(num_points, size=2, replace=False)
        # Probability of preference using Bradley-Terry model
        p_ij = sigmoid(f_vals[i] - f_vals[j])
        # Decide preference based on random draw
        if np.random.rand() < p_ij:
            preferences.append((i, j))
        else:
            preferences.append((j, i))
    return preferences

preferences = generate_preferences(f_values)

# Step 3: Define the Likelihood Function for Elliptical Slice Sampling
def loglik_from_preferences(f):
    """
    Log-likelihood function using Bradley-Terry model for preferences.

    Args:
    - f (np.array): Sampled function values.

    Returns:
    - float: Log-likelihood value.
    """
    log_lik = 0
    for idx_i, idx_j in preferences:
        # YOUR CODE HERE (~2 lines)
        pass
        # END OF YOUR CODE
    return log_lik
```

```python

# Step 4: Define the RBF Kernel to Compute Prior Covariance Matrix
def rbf_kernel(X1, X2, length_scale=1.0, sigma_f=1.0):
    """
    Computes the Radial Basis Function (RBF) kernel between two sets of points.

    Args:
    - X1, X2 (np.array): Input data points.
    - length_scale (float): Kernel length scale parameter.
    - sigma_f (float): Kernel output scale.

    Returns:
    - np.array: RBF kernel matrix.
    """
    sqdist = np.sum(X1**2, axis=1).reshape(-1, 1) + np.sum(X2**2, axis=1) - 2 * np.dot(X1, X2.T)
    return sigma_f**2 * np.exp(-0.5 / length_scale**2 * sqdist)

# Define prior covariance (prior mean is zero vector)
sigma_prior = rbf_kernel(x_grid_points, x_grid_points, length_scale=1.0, sigma_f=1.0)

# Add small jitter to diagonal for numerical stability
jitter = 1e-6
sigma_prior += jitter * np.eye(sigma_prior.shape[0])

# Ensure the matrix is symmetric to avoid numerical issues
sigma_prior = (sigma_prior + sigma_prior.T) / 2

# Step 5: Run Elliptical Slice Sampling
sampler = EllipticalSliceSampler(sigma_prior, loglik_from_preferences)
samples = sampler.sample(1000, n_burn=500)
average_samples = np.mean(samples, axis=0)

# Generate true function values on grid points
true_values_on_grid = nonlinear_function(x_grid_points[:, 0], x_grid_points[:, 1])

def evaluate_elicited_metric(true_metric, elicited_metric):
    """
    Evaluates and prints the mean and standard deviation of the difference
    between true and elicited metrics.

    Args:
    - true_metric (np.array): True values of the function.
    - elicited_metric (np.array): Elicited (estimated) function values.
    """
    # YOUR CODE HERE
    pass
    # END OF YOUR CODE
```

```
evaluate_elicited_metric(true_values_on_grid, average_samples)

# Step 6: Plot the True Non-linear Function and Elicited Metric in 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the true function
x1_fine = np.linspace(0, 1, 50)
x2_fine = np.linspace(0, 1, 50)
x1_fine_grid, x2_fine_grid = np.meshgrid(x1_fine, x2_fine)
true_f_values = nonlinear_function(x1_fine_grid, x2_fine_grid)
ax.plot_surface(x1_fine_grid, x2_fine_grid, true_f_values, color='blue', alpha=0.5,
 ↪  label='True Function')

# Plot the averaged samples as a surface
x1_avg = x_grid_points[:, 0].reshape(20, 20)
x2_avg = x_grid_points[:, 1].reshape(20, 20)
avg_values = average_samples.reshape(20, 20)
ax.plot_surface(x1_avg, x2_avg, avg_values, color='red', alpha=0.5, label='Estimated
 ↪  Function')

# Customize plot
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('f(x1, x2)')
ax.set_title('True Function vs. Averaged Estimated Function')
plt.legend()
plt.show()
```

References

Amershi, Saleema, Maya Cakmak, W. Bradley Knox, and Todd Kulesza. 2014. "Power to the People: The Role of Humans in Interactive Machine Learning." *AI Magazine*.

Beluch, William H., Tim Genewein, A. Nürnberger, and Jan M. Köhler. 2018. "The Power of Ensembles for Active Learning in Image Classification." *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9368–77. https://api.semanticscholar.org/CorpusID:52838058.

Bernard, J., Matthias Zeppelzauer, Markus Lehmann, Martin Müller, and Michael Sedlmair. 2018. "Towards User□centered Active Learning Algorithms." *Computer Graphics Forum* 37. https://api.semanticscholar.org/CorpusID:51875861.

Biyik, Erdem, and Dorsa Sadigh. 2018. "Batch Active Preference-Based Learning of Reward Functions." In *Proceedings of the 2nd Conference on Robot Learning*, edited by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, 87:519–28. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v87/biyik18a.html.

Bommasani, Rishi, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, et al. 2022. "On the Opportunities and Risks of Foundation Models." https://arxiv.org/abs/2108.07258.

Bouneffouf, Djallel, Romain Laroche, Tanguy Urvoy, Raphaël Féraud, and Robin Allesiardo. 2014. "Contextual

Bandit for Active Learning: Active Thompson Sampling." In *International Conference on Neural Information Processing*. https://api.semanticscholar.org/CorpusID:1701357.

Braziunas, Darius, and Craig Boutilier. 2012. "Minimax Regret Based Elicitation of Generalized Additive Utilities." https://arxiv.org/abs/1206.5255.

Brohan, Anthony, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, et al. 2023. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control." https://arxiv.org/abs/2307.15818.

Bulow, Jeremy, and Paul Klemperer. 1996. "Auctions Versus Negotiations." *The American Economic Review* 86 (1): 180–94. http://www.jstor.org/stable/2118262.

Cai, Wenbin, Ya Zhang, and Jun Zhou. 2013. "Maximizing Expected Model Change for Active Learning in Regression." In *2013 IEEE 13th International Conference on Data Mining*, 51–60. https://doi.org/10.1109/ICDM.2013.104.

Cohn, David A., Zoubin Ghahramani, and Michael I. Jordan. 1996. "Active Learning with Statistical Models." *CoRR* cs.AI/9603104. https://arxiv.org/abs/cs/9603104.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. "ImageNet: A Large-Scale Hierarchical Image Database." In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–55. IEEE.

G., Jamieson Kevin, and Robert Nowak. 2011. "Active Ranking Using Pairwise Comparisons." *Advances in Neural Information Processing Systems* 24.

Gandhi, Kanishk, Siddharth Karamcheti, Madeline Liao, and Dorsa Sadigh. 2022. "Eliciting Compatible Demonstrations for Multi-Human Imitation Learning." In *Proceedings of the 6th Conference on Robot Learning (CoRL)*.

Geman, Stuart, Elie Bienenstock, and René Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4:1–58. https://api.semanticscholar.org/CorpusID:14215320.

Ghojogh, Benyamin, Hadi Nekoei, Aydin Ghojogh, Fakhri Karray, and Mark Crowley. 2020. "Sampling Algorithms, from Survey Sampling to Monte Carlo Methods: Tutorial and Literature Review." https://arxiv.org/abs/2011.00901.

Grauman, Kristen, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, et al. 2022. "Ego4D: Around the World in 3,000 Hours of Egocentric Video." https://arxiv.org/abs/2110.07058.

Guillory, Andrew, and Jeff Bilmes. 2011. "Simultaneous Learning and Covering with Adversarial Noise." *ICML*.

Halford, Max. 2023. "Online Active Learning in 80 Lines of Python."

Hartline, Jason D., Yingkai Li, Liren Shan, and Yifan Wu. 2020. "Optimization of Scoring Rules." *CoRR* abs/2007.02905. https://arxiv.org/abs/2007.02905.

Hartline, Jason D., Liren Shan, Yingkai Li, and Yifan Wu. 2023. "Optimal Scoring Rules for Multi-Dimensional Effort." In *Proceedings of Thirty Sixth Conference on Learning Theory*, edited by Gergely Neu and Lorenzo Rosasco, 195:2624–50. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v195/hartline23a.html.

He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. "Momentum Contrast for Unsupervised Visual Representation Learning." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9729–38. IEEE.

Hiranandani, Gaurush, Shant Boodaghians, Ruta Mehta, and Oluwasanmi Koyejo. 2019a. "Performance Metric Elicitation from Pairwise Classifier Comparisons." In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, edited by Kamalika Chaudhuri and Masashi Sugiyama, 89:371–79. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v89/hiranandani19a.html.

Hiranandani, Gaurush, Shant Boodaghians, Ruta Mehta, and Oluwasanmi O Koyejo. 2019b. "Multiclass Performance Metric Elicitation." In *Advances in Neural Information Processing Systems*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/1fd09c5f59a8ff35d499c0ee25a1d47e-Paper.pdf.

Hiranandani, Gaurush, Harikrishna Narasimhan, and Sanmi Koyejo. 2020. "Fair Performance Metric Elicitation." In *Advances in Neural Information Processing Systems*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan,

and H. Lin, 33:11083–95. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/7ec2442aa04c157590b2fa1a7d093a33-Paper.pdf.

Holladay, Rachel, Shervin Javdani, Anca Dragan, and Siddhartha Srinivasa. 2016. "Active Comparison Based Learning Incorporating User Uncertainty and Noise." *Proceedings of RSS '16 Workshop on Model Learning for Human-Robot Communication*.

Houlsby, Neil, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. 2011. "Bayesian Active Learning for Classification and Preference Learning." *arXiv Preprint arXiv:1112.5745*.

Jamieson, Kevin G, Robert Nowak, and Ben Recht. 2012. "Query Complexity of Derivative-Free Optimization." In *Advances in Neural Information Processing Systems*, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/e6d8545daa42d5ced125a4bf747b3688-Paper.pdf.

Jarl, Sanna, Linus Aronsson, Sadegh Rahrovani, and Morteza Haghir Chehreghani. 2021. "Active Learning of Driving Scenario Trajectories." *Eng. Appl. Artif. Intell.* 113:104972. https://api.semanticscholar.org/CorpusID:249113683.

Karamcheti, Siddharth, Suraj Nair, Annie S. Chen, Thomas Kollar, Chelsea Finn, Dorsa Sadigh, and Percy Liang. 2023. "Language-Driven Representation Learning for Robotics." https://arxiv.org/abs/2302.12766.

Kong, Yuqing, and Grant Schoenebeck. 2019. "An Information Theoretic Framework for Designing Information Elicitation Mechanisms That Reward Truth-Telling." *ACM Trans. Econ. Comput.* 7 (1). https://doi.org/10.1145/3296670.

Kwon, Minae, Siddharth Karamcheti, Mariano-Florentino Cuellar, and Dorsa Sadigh. 2021. "Targeted Data Acquisition for Evolving Negotiation Agents." https://arxiv.org/abs/2106.07728.

Li, Kejun, Maegan Tucker, Erdem Biyik, Ellen Novoseller, Joel W. Burdick, Yanan Sui, Dorsa Sadigh, Yisong Yue, and Aaron D. Ames. 2021. "ROIAL: Region of Interest Active Learning for Characterizing Exoskeleton Gait Preference Landscapes." In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. https://doi.org/10.1109/icra48506.2021.9560840.

Ma, Jiaqi, Ziqiao Ma, Joyce Chai, and Qiaozhu Mei. 2022. "Partition-Based Active Learning for Graph Neural Networks." *ArXiv* abs/2201.09391. https://api.semanticscholar.org/CorpusID:246240846.

Makili, Lázaro Emílio, Jesús A. Vega Sánchez, and Sebastián Dormido-Canto. 2012. "Active Learning Using Conformal Predictors: Application to Image Classification." *Fusion Science and Technology* 62:347–55. https://api.semanticscholar.org/CorpusID:115384000.

Margatina, Katerina, Timo Schick, Nikolaos Aletras, and Jane Dwivedi-Yu. 2023. "Active Learning Principles for in-Context Learning with Large Language Models." *ArXiv* abs/2305.14264. https://api.semanticscholar.org/CorpusID:258841313.

Mas-Colell, Andreu. 1977. "The Recoverability of Consumers' Preferences from Market Demand Behavior." *Econometrica* 45 (6): 1409–30. http://www.jstor.org/stable/1912308.

McAfee, R. Preston, and John McMillan. 1987. "Auctions and Bidding." *Journal of Economic Literature* 25 (2): 699–738. http://www.jstor.org/stable/2726107.

Mussmann, Stephen, Julia Reisler, Daniel Tsai, Ehsan Mousavi, Shayne O'Brien, and Moises Goldszmidt. 2022. "Active Learning with Expected Error Reduction." https://arxiv.org/abs/2211.09283.

Myers, Vivek, Erdem Bıyık, Nima Anari, and Dorsa Sadigh. 2021. "Learning Multimodal Rewards from Rankings." https://arxiv.org/abs/2109.12750.

Nair, Suraj, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. 2022. "R3M: A Universal Visual Representation for Robot Manipulation." https://arxiv.org/abs/2203.12601.

Narasimhan, Harikrishna, Harish Ramaswamy, Aadirupa Saha, and Shivani Agarwal. 2015. "Consistent Multiclass Algorithms for Complex Performance Measures." In *Proceedings of the 32nd International Conference on Machine Learning*, edited by Francis Bach and David Blei, 37:2398–2407. Proceedings of Machine Learning Research. Lille, France: PMLR. https://proceedings.mlr.press/v37/narasimhanb15.html.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, et al. 2021. "Learning Transferable Visual Models from Natural Language Supervision." *arXiv Preprint arXiv:2103.00020*.

Samuelson, P. A. 1938. "A Note on the Pure Theory of Consumer's Behaviour." *Economica* 5 (17): 61–71. http://www.jstor.org/stable/2548836.

Shepard, Roger N. 1957. "Stimulus and Response Generalization: A Stochastic Model Relating Generalization to Distance in Psychological Space." *Psychometrika* 22(4):325–345.

Singh, Aarti, Robert D. Nowak, and Parameswaran Ramanathan. 2006. "Active Learning for Adaptive Mobile Sensing Networks." *2006 5th International Conference on Information Processing in Sensor Networks*, 60–68. https://api.semanticscholar.org/CorpusID:17590956.

Tamburrelli, Giordano, and Alessandro Margara. 2014. "Towards Automated a/b Testing." In *Search-Based Software Engineering*. https://doi.org/10.1007/978-3-319-09940-8_13.

Taylor, Annalisa T., Thomas A. Berrueta, and Todd D. Murphey. 2021. "Active Learning in Robotics: A Review of Control Principles." *ArXiv* abs/2106.13697. https://api.semanticscholar.org/CorpusID:235652039.

Varian, Hal R. 2006. "Revealed Preference." In *The SAGE Encyclopedia of Business Ethics and Society*. https://api.semanticscholar.org/CorpusID:1632873.

Viappiani, Paolo, and Craig Boutilier. 2010. "Optimal Bayesian Recommendation Sets and Myopically Optimal Choice Query Sets." *NIPS*, 2352–60.

Walke, Homer, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, et al. 2023. "BridgeData V2: A Dataset for Robot Learning at Scale." https://arxiv.org/abs/2308.12952.

Xiao, Tete, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. 2022. "Masked Visual Pre-Training for Motor Control." https://arxiv.org/abs/2203.06173.

Xie, Annie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. 2022. "When to Ask for Help: Proactive Interventions in Autonomous Reinforcement Learning." https://arxiv.org/abs/2210.10765.

Yang, Sitan, and Daniel Q. Naiman. 2014. "Multiclass Cancer Classification Based on Gene Expression Comparison." *Statistical Applications in Genetics and Molecular Biology* 13 (4): 477–96. https://doi.org/doi:10.1515/sagmb-2013-0053.

Zhao, Shuyang, Toni Heittola, and Tuomas Virtanen. 2020. "Active Learning for Sound Event Detection." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28:2895–905. https://api.semanticscholar.org/CorpusID:211082815.

Zhu, Jingbo, Huizhen Wang, Benjamin Ka-Yin T'sou, and Matthew Y. Ma. 2010. "Active Learning with Sampling by Uncertainty and Density for Data Annotations." *IEEE Transactions on Audio, Speech, and Language Processing* 18:1323–31. https://api.semanticscholar.org/CorpusID:5777911.

4 Model-Free Preference Optimization

4.1 Individual Preference Optimization via Dueling Bandit

4.1.1 Introduction to Dueling Bandit Problem and Its Extension

The multi-armed bandit (MAB) problem involves a gambler deciding which lever to pull on an MAB machine to maximize the winning rate, despite not knowing which machine is the most rewarding. This scenario highlights the need to balance exploration (trying new machines to discover potential higher rewards) and exploitation (using current knowledge to maximize gains). MAB algorithms address this dilemma by making decisions under uncertainty to achieve the best possible outcomes based on gathered data. At the core of the MAB problem is a set of actions, or 'arms,' denoted by $\mathcal{A} = \{1, 2, \ldots, K\}$, where K signifies the total number of arms. For each round t, the agent selects an arm $a_t \in \mathcal{A}$ and receives a reward r_t, sampled from an arm-specific, unknown probability distribution. The expected reward of pulling arm a is represented as $\mu_a = \mathbb{E}[r_t|a]$.

The multi-armed bandit framework can be extended in various ways to model more complex scenarios. In the infinite-armed bandit problem, the set of possible arms $\mathcal{A}$ is either very large or infinite. This introduces significant challenges in exploration, as the agent cannot afford to explore each arm even once. Algorithms for infinite-armed bandits typically assume some regularity or structure of the reward function across arms to make the problem tractable. The contextual bandit problem extends the bandit framework by incorporating observable external states or contexts that influence the reward distributions of arms. The agent's task is to learn policies that map contexts to arms to maximize reward. This model is particularly powerful for personalized recommendations, where the context can include user features or historical interactions. In dueling bandit problems, the agent chooses two arms to pull simultaneously and receives feedback only on which of the two is better, not the actual reward values. This pairwise comparison model is especially useful in scenarios where absolute evaluations are difficult, but relative preferences are easier to determine, such as in ranking systems.

Contextual bandits extend the multi-armed bandits by making decisions conditional on the state of the environment and previous observations. The benefit of such a model is that observing the environment can provide additional information, potentially leading to better rewards and outcomes. In each iteration, the agent is presented with the context of the environment, then decides on an action based on the context and previous observations. Finally, the agent observes the action's outcome and reward. Throughout this process, the agent aims to maximize the expected reward.

In many real-world contexts, one may not have a real-valued reward (or at least a reliable one) associated with a decision. Instead, we may only have observations indicating which of a set of bandits was optimal in a given scenario. The assumption is that within these observations of preferred choices among a set of options, there is an implicit reward or payoff encapsulated in that decision. Consider the following examples:

1. **Dietary preferences**: When providing food recommendations to humans, it is often not possible to quantify an explicit reward from recommending a specific food item. Instead, we can offer meal options and observe which one the person selects.
2. **Video recommendation**: Websites like YouTube and TikTok recommend specific videos to users. It is typically not feasible to measure the reward a person gains from watching a video. However, we can infer that a user preferred one video over another. From these relative preference observations, we can develop a strategy to recommend videos they are likely to enjoy.
3. **Exoskeleton gait optimization**: Tucker et al. (2020) created a framework that uses human-evaluated preferences for an exoskeleton gait algorithm to develop an optimal strategy for the exoskeleton to assist a human in walking. A human cannot reliably produce a numerical value for how well the exoskeleton helped them walk but can reliably indicate which option performed best according to their preferences.

Generally, we assume access to a set of actions. A noteworthy assumption is that any observations we make are unbiased estimates of the payoff. This means that if we observe a human preferred one option over another (or several others), the preferred option had a higher implicit reward or payoff than the alternatives. In the case of dietary preferences, this may mean that a human liked the preferred option; in the case of video recommendations, a user was more entertained, satisfied, or educated by the video they selected than the other options.

The overarching context is that we do not have direct or reliable access to rewards. We may not have a reward at all (for some decisions, it may be impossible to define a real value to the outcome), or it may be noisy (for example, if we ask a human to rate their satisfaction on a scale of 1 to 10). We use relative comparisons to evaluate the best of multiple options in this case. Our goal is to minimize total regret in the face of noisy comparisons. Humans may not always provide consistent observations (since human decision-making is not guaranteed to be consistent). However, we can still determine an optimal strategy with the observed comparisons. We aim to minimize the frequency of sub-optimal decisions according to human preferences. In practice, many formulations of bandits can allow for infinitely many bandits (for example, in continuous-value and high-dimensional spaces). However, this situation can be intractable when determining an optimal decision strategy. With infinite options, how can we always ensure we have chosen the best? We will constrain our bandits to a discrete space to enable efficient exploration. We will assume that we have k bandits, $b_i, i \in [1, k]$, and our task is to choose the one that will minimize regret.

With the framework outlined, we now define our approach more formally. This method was introduced by (Yue et al. 2012), and proofs for the guarantees and derivations of parameters

can be found in their work.

To determine the optimal action, we will compare pairwise to ascertain the probability that an action b_i is preferred over another b_j, where $i \neq j$. Concretely, we assume access to a function ϵ that helps determine this probability; in practice, this can be done with an oracle, such as asking a human which of two options they prefer:

$$P(b_i > b_j) = \varepsilon(b_i, b_j) + \frac{1}{2}.$$

With this model, three basic properties govern the values provided by ϵ:

$$\epsilon(b_i, b_j) = -\epsilon(b_j, b_i), \epsilon(b_i, b_i) = 0, \epsilon(b_i, b_j) \in \left(-\frac{1}{2}, \frac{1}{2}\right).$$

We assume there is a total ordering of bandits, such that $b_i \succ b_j$ implies $\epsilon(b_i, b_j) > 0$. We impose two constraints to properly model comparisons:

- **Strong Stochastic Transitivity**: We must maintain our total ordering of bandits, and as such, the comparison model also respects this ordering:

$$b_i \succ b_j \succ b_k \Rightarrow \epsilon(b_i, b_k) \geq \max\{\epsilon(b_i, b_j), \epsilon(b_j, b_k)\}. \tag{4.1}$$

- **Stochastic Triangle Inequality**: We also impose a triangle inequality, which captures the condition that the probability of a bandit winning (or losing) a comparison will exhibit diminishing returns as it becomes increasingly superior (or inferior) to the competing bandit:

$$b_i \succ b_j \succ b_k \Rightarrow \epsilon(b_i, b_k) \leq \epsilon(b_i, b_j) + \epsilon(b_j, b_k). \tag{4.2}$$

These assumptions may initially seem limiting; however, common models for comparisons satisfy these constraints. For example, the Bradley-Terry Model follows $P(b_i > b_j) = \frac{\mu_i}{\mu_i + \mu_j}$. The Gaussian model with unit variance also satisfies these constraints: $P(b_i > b_j) = P(X_i - X_j > 0)$, where $X_i - X_j \sim N(\mu_i - \mu_j, 2)$.

To accurately model the preferences between bandits in our framework of pairwise bandit comparisons and regret, we must track certain parameters in our algorithm. First, we will maintain a running empirical estimate of the probability of bandit preferences based on our observations. It is important to note that we do not have direct access to an ϵ function. Instead, we must present two bandits to a human, who selects a winner. To do this, we define:

$$\hat{P}_{i,j} = \frac{\#b_i \text{ wins}}{\#\text{comparisons between } i \text{and } j}.$$

We will also compute confidence intervals at each timestep for each of the entries in $\hat{P}$ as

$$\hat{C}_t = \left(\hat{P}_t - c_t, \hat{P}_t + c_t\right),$$

where $c_t = \sqrt{\frac{4\log(\frac{1}{\delta})}{t}}$. Note that $\delta = \frac{1}{TK^2}$, where T is the time horizon and K is the number of bandits.

Previously, we discussed approaches for finding the best action in a specific context. Now, we consider changing contexts, which means there is no longer a static hidden preference matrix P. Instead, at every time step, there is a preference matrix P_C depending on context C. We consider a context C and a preference matrix P_C to be chosen by nature as a result of the given environment (Yue et al., 2012). The goal of a contextual bandits algorithm is to find a policy π that maps contexts to a Von Neumann winner distribution over our bandits. That is, our policy π should map any context to some distribution over our bandits such that sampling from that distribution is preferred to a random action for that context.

4.1.2 Regret

The agent aims to pick a sequence of arms $(a_1, a_2, \ldots, a_T)$ across a succession of time steps $t = 1$ to $t = T$ to maximize the total accumulated reward. Formally, the strategy seeks to maximize the sum of the expected rewards: $\max_{a_1,\ldots,a_T} \mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$. Regret is defined as the difference between the cumulative reward that could have been obtained by always pulling the best arm (in hindsight, after knowing the reward distributions) and the cumulative reward actually obtained by the algorithm. Formally, if μ^* is the expected reward of the best arm and μ_{a_t} is the expected reward of the arm chosen at time t, the regret after T time steps is given by $R(T) = T \cdot \mu^* - \sum_{t=1}^{T} \mu_{a_t}$. The objective of a bandit algorithm is to minimize this regret over time, effectively learning to make decisions that are as close as possible to the decisions of an oracle that knows the reward distributions beforehand. Low regret indicates an algorithm that has often learned to choose well-performing arms, balancing the exploration of unknown arms with the exploitation of arms that are already known to perform well. Thus, an efficient bandit algorithm exhibits sub-linear regret growth, meaning that the average regret per round tends to zero as the number of rounds T goes to infinity: $\lim_{T\to\infty} \frac{R(T)}{T} = 0$. Minimizing regret is a cornerstone in the design of bandit algorithms, and its analysis helps in understanding the long-term efficiency and effectiveness of different bandit strategies.

As previously discussed, our goal is to select the bandit that minimizes a quantity that reflects regret or the cost of not selecting the optimal bandit at all times. We can leverage our comparison model to define a quantity for regret over some time horizon T, which is the number of decisions we make (selecting what we think is the best bandit at each iteration). Assuming we know the best bandit b^* (and we know that there *is* a best bandit, since there is a total ordering of our discrete bandits), we can define two notions of regret:

- Strong regret: aims to capture the fraction of users who would prefer the optimal bandit b^* over the *worse* of the options b_1, b_2 we provide at a given step:$R_T = \sum_{t=1}^{T} \max\left\{\epsilon(b^*, b_1^{(t)}), \epsilon(b^*, b_2^{(t)})\right\}$

– Weak regret: aims to capture the fraction of users who would prefer the optimal bandit b^* over the *better* of the options b_1, b_2 we provide at a given step:$\tilde{R}_T = \sum_{t=1}^{T} \min\left\{\epsilon(b^*, b_1^{(t)}), \epsilon(b^*, b_2^{(t)})\right\}$

The best bandit described in our regret definition is called a **Condorcet Winner**. This is the strongest form of winner. It's the action A_i which is preferred to each other action A_j with $p > 0.5$ in a head-to-head election. While the above introduced notions of regret assume an overall best bandit to exist, there might be settings, where no bandit wins more than half head-to-head duels. A set of actions without a Condorcet winner is described by the following preference matrix, where each entry Δ_{jk} is $p(j \succ k) - 0.5$, the probability that action j is preferred over action k minus 0.5. There is no Condorcet winner as there is no action that is preferred with $p > 0.5$ over all other actions. Imagine, you want to find the best pizza to eat (=action). There may not be a pizza that wins more than half of the head-to-head duels against every other pizza.

However, we might still have an intuition of the best pizza. Therefore Sui et al., 2018 introduce the concepts of different *winners* in dueling bandit problems (Sui et al. 2018). In this example, we might define the best pizza as the most popular one. We call the Pizza receiving the most votes in a public vote the **Borda Winner**, or formally, Borda winner $j = \arg\max_{i \in A, i \neq j} (\sum p(j \succ i))$. In contrast to the Condorcet Winner setting, there is always guaranteed to be one or more (in the case of a tie) Borda winners for a set of actions. However - if there is a Condorcet Winner, this might not necessarily be the same as a Borda Winner: In our Pizza example, a Pepperoni Pizza might win more than half of its head-to-head duels, while the Cheese-Pizza is still the most popular in a public poll.

A more generic concept of winner is the **Von Neumann Winner**, which describes a probability distribution rather than a single bandit winner. A Von Neumann winner simply prescribes a probability distribution W such that sampling from this distribution 'beats' an action from the random uniform distribution with $p > 0.5$. In our pizza example, this would correspond to trusting a friend to order whichever Pizza he likes, because this may still be preferred to ordering randomly. Formally, W is a Von Neumann if $(j \sim W, k \sim R)[p(p(j \succ k) > 0.5) > 0.5]$ where R describes the uniform probability distribution over our actions. The concept of a Von Neumann winner is useful in contextual bandits, which will be introduced later. In these settings, the preference matrix depends on different context, which may have different Borda winners, just as different parties may vote for different pizzas.

Next, we introduce two performance measures for the planner. The **asymptotic ex-post regret** is defined as

$$\text{Regret}(\mu_1, \ldots \mu_K) = T \cdot \max_i \mu_i - \sum_{i=1}^{T} E[\mu_{I_t}].$$

Intuitively, this represents the difference between the reward achieved by always taking the action with the highest possible reward and the expected welfare of the recommendation algorithm (based on the actions it recommends at each timestep).

	A	B	C	D	E	F
A	0	**0.03**	**-0.02**	0.06	0.10	0.11
B	-0.03	0	**0.03**	0.05	0.08	0.11
C		-0.03	0	0.04	0.07	0.09
D	-0.06	-0.05	-0.04	0	0.05	0.07
E	-0.10	-0.08	-0.07	-0.05	0	0.03
F	-0.11	-0.11	-0.09	-0.07	-0.03	0

Figure 4.1
Violation of Condorcet Winner. Highlighted entries are different from Table 1. No Condorcet winner exists as no arm could beat every other arm.

We also define a weaker performance measure, the **Bayesian regret**, which is defined as

$$\text{Bayesian regret} = E_{\mu_1,\ldots,\mu_K \sim \text{Prior}} \left[\text{Regret}\left(\mu_1, \ldots, \mu_K\right)\right]$$

With a Bayesian optimal policy, we would like either definition of regret to vanish as $T \to \infty$; we are considering "large-market optimal" settings where there are many short-lived, rather than a few long-term, users. Note the fact that ex-post regret is prior-free makes it robust to inaccuracies on the prior.

4.1.3 Acquisition Functions

Various strategies have been developed to balance the exploration-exploitation trade-off. These strategies differ in selecting arms based on past experiences and rewards.

4.1.3.1 Classical Acquisition Functions

Uniform acquisition function is the most straightforward approach where each arm is selected uniformly randomly over time. This strategy does not consider the past rewards and treats each arm equally promising regardless of the observed outcomes. It is a purely explorative strategy that ensures each arm is sampled enough to estimate its expected reward, but it does not exploit the information to optimize rewards. In mathematical terms, if $N_t(a)$ denotes the number of times arm a has been selected up to time t, the Uniform Strategy would ensure that $N_t(a) \approx \frac{t}{K}$ for all arms a as t grows large: $P(a_t = a) = \frac{1}{K}$

The **Epsilon Greedy** is a popular method that introduces a balance between exploration and exploitation. With a small probability ϵ, it explores by choosing an arm at random, and with a probability $1 - \epsilon$, it exploits by selecting the arm with the highest estimated reward so far. This strategy incrementally favors actions that have historically yielded higher rewards, but still allows for occasional exploration to discover better options potentially. The parameter ϵ is

chosen based on the desired exploration level, often set between 0.01 and 0.1.

$$P(a_t = a) = \begin{cases} \frac{\epsilon}{K} + 1 - \epsilon & \text{if } a = \arg\max_{a'} \hat{\mu}_{a'} \\ \frac{\epsilon}{K} & \text{otherwise} \end{cases}$$

Upper Confidence Bound (UCB) acquisition function takes a more sophisticated approach to the exploration-exploitation dilemma. It selects arms based on both the estimated rewards and the uncertainty or variance associated with those estimates. Specifically, it favors arms with high upper confidence bounds on the estimated rewards, which is a sum of the estimated mean and a confidence interval that decreases with the number of times the arm has been played. This ensures that arms with less certainty (those played less often) are considered more often, naturally balancing exploration with exploitation as the uncertainty is reduced over time.

$$P(a_t = a) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} \left(\hat{\mu}_{a'} + \sqrt{\frac{2 \ln t}{N_t(a')}} \right) \\ 0 & \text{otherwise} \end{cases}$$

4.1.3.2 Interleaved Filter

This algorithm tries to find the best bandit (Condorcet Winner) in a discrete, limited bandit-space via pairwise comparisons of the bandits. We will now introduce the algorithm for the Interleaved Filter as provided in (Yue et al. 2012) to solve a dueling bandit setup. It starts with a randomly defined *best bandit* $\hat{b}$ and iteratively compares it to set W containing the remaining bandits b resulting in winning probabilities $\hat{P}_{\hat{b},b}$ and confidence interval $\hat{C}_{\hat{b},b}$. If a bandit b is *confidently worse* than $\hat{b}$, it is removed from W. If a bandit b' is *confidently better* than $\hat{b}$, it is set as new *best bandit* $\hat{b}$ and bandit $\hat{b}$ as well as every other bandit b *worse* than $\hat{b}$ are removed from W. This is done, until W is empty, leaving the final $\hat{b}$ as the predicted best bandit.

input: T, $B = \{b_1, \dots, b_k\}$ $\delta \leftarrow 1/(TK^2)$ Choose $\hat{b} \in B$ randomly $W \leftarrow \{b_1, \dots, b_k\} \backslash \{\hat{b}\}$ $\forall b \in W$, maintain estimate $\hat{P}_{\hat{b},b}$ of $P(\hat{b} > b)$ according to (6) $\forall b \in W$, maintain $1 - \delta$ confidence interval $\hat{C}_{\hat{b},b}$ of $\hat{P}_{\hat{b},b}$ according to (7), (8) compare $\hat{b}$ and b update $\hat{P}_{\hat{b},b}$, $\hat{C}_{\hat{b},b}$ $W \leftarrow W \backslash \{b\}$

$W \leftarrow W \backslash \{b\}$ $\hat{b} \leftarrow b'$, $W \leftarrow W \backslash \{b'\}$ $\forall b \in W$, reset $\hat{P}_{\hat{b},b}$ and $\hat{C}_{\hat{b},b}$ $\hat{T} \leftarrow$ Total Comparisons Made $(\hat{b}, \hat{T})$

Parameter Initialization In lines 1-6 of the algorithm, we take the inputs and first compute the value δ which is used to compute our confidence intervals. We select an initial guess of an optimal bandit $\hat{b}$ by uniformly sampling from all bandits $\mathcal{B}$. We also keep a running set of bandit candidates W, which is initialized to be $\mathcal{B} \ \{\hat{b}\}$. At this point, we also initialize our empirical estimates for $\hat{P}, \hat{C}$.

Next, we will repeat several steps until our working set of bandit candidates W is empty.

Update Estimates Based on Comparisons The first step at each iteration (lines 8–11) is to look at all candidates in W, and compare them to our current guess $\hat{b}$ using an oracle (e.g. by asking a human which of $\hat{b}$ or $b \in W$ is preferred). With this new set of wins and comparisons, we update our estimates of $\hat{P}, \hat{C}$.

Prune Suboptimal Bandits In lines 12–13, with updated comparison win probabilities and corresponding confidence intervals, we can remove bandit candidates from W that we are *confident* $\hat{b}$ is better than. The intuition here is that we are mostly sure that our current best guess is better than some of the candidates, and we don't need to consider those candidates in future iterations.

Check for Better Bandits from Candidate Set Now that our candidate set of bandits may be smaller, in lines 15–21 we check if there are any bandits b' that we are *confident* are better than our current best guess. If we do find such a candidate, we remove bandits which $\hat{P}$ indicates b is *likely* worse than $\hat{b}$. Note that in this step, we do not require the probability to be outside the confidence interval, since we already found one we believe to be significantly closer to optimal than our current best guess.

Once we remove the candidates *likely* worse than $\hat{b}$, we crown b' as the new best guess, e.g. $\hat{b} := b'$. Consequently, we remove b' from W and reset our empirical win counters $\hat{P}, \hat{C}$.

With this algorithm defined, let us look at some provisions of the method with respect to identifying the optimal strategy. Note that the proofs and derivations for these quantities are provided in (Yue et al. 2012).

First, the method guarantees that for the provided time horizon T, the algorithm returns the correct bandit with probability $P \geq 1 - \frac{1}{T}$. It is interesting and useful to note that if one has a strict requirement for the probability of identifying the correct bandit, one can compute the time horizon T that guarantees this outcome at that probability. Furthermore, a time horizon of 1 leaves no probabilistic guarantee of a successful outcome, and increasing T has diminishing returns. Second, in the event that the algorithm returns an incorrect bandit, the maximal regret incurred is linear with respect to T, e.g. $(O)(T)$. This is also a useful provision as it allows us to estimate the overall cost in the worst case outcome. Based on these two provisions, we can compute the expected cumulative regret from running the Interleaved Filter algorithm, which is:

$$\mathbb{E}\left[R_T\right] \leq \left(1 - \frac{1}{T}\right) \mathbb{E}\left[R_T^{IF}\right] + \frac{1}{T}\mathcal{O}(T) = \mathcal{O}\left(\mathbb{E}\left[R_T^{IF}\right] + 1\right)$$

Interestingly, the original work shows that these bounds hold for both strong and weak regret. As demonstrated, the Interleaved Filter algorithm [fig-if] provides a robust method to ascertain the optimal bandit or strategy given a set of options and only noisy comparisons. In most real-world scenarios for modeling human preferences, it is not possible to observe a real-world reward value, or at least a reliable one and as such this method is a useful way to properly model human preferences.

Furthermore, the algorithm provides strong guarantees for the probability of selecting the correct bandit, maximal regret, and the number of comparisons required. It is even more impressive that the method can do so without severely limiting constraints; as demonstrated, the most commonly used models satisfy the imposed constraints.

As we look to model human preferences, we can certainly leverage this method for k-armed dueling bandits to identify the best strategy to solve human-centric challenges, from video recommendation to meal selection and exoskeleton-assisted walking.

4.1.3.3 Dueling Bandit Gradient Descent

This algorithm tries to find the best bandit in a continuous bandit-space. Here, the set of all bandits is regarded as an Information-Retrieval (IR) system with infinite bandits uniquely defined by w. We will cover the *Dueling Bandit Gradient Descent* algorithm from Yue and Joachims 2009 (Yue and Joachims 2009). Yue and Joachims use the dueling bandits formulation for online IR optimization. They propose a retrieval system parameterized by a set of continuous variables lying in W, a d-dimensional unit-sphere. The DBGD algorithm adapts the current parameters w_t of IR system by comparison with slightly altered parameters w'_t both querying query q_t. Only if the IR outcome using w'_t is preferred, the parameters are changed in their direction. We will now discuss the algorithm more detailed.

input: γ, δ, w_1

Sample unit vector u_t uniformly

$w'_t \leftarrow P_W(w_t + \delta u_t)$

Compare w_t and w'_t

$w_{t+1} \leftarrow P_W(w_t + \gamma u_t)$

$w_{t+1} \leftarrow w_t$

We first choose exploration step length δ, exploitation step length γ, and starting point (in unit-sphere) w_1. Choose a query and sample a random unit vector u_t. We duel w_t and w'_t, where w_t is our current point in the sphere, and w'_t is our exploratory comparison, which is generated by taking a random step of length δ, such that $w'_t = w_t + \delta u_t$. The objective of this duel is to ascertain the binary preference of users with respect to the results yielded by the IR systems parameterized by w_t and w'_t respectively, taking query q_t as an input. The parameters that get the majority of the votes in the head to head win. If w_t wins, then we keep the parameters for the next iteration. If w'_t wins the duel, we update our parameters in the direction of u_t by taking a step of length γ. Note that the algorithm describes projection operation $P_W(\vec{v})$. Since u_t is chosen randomly, $w_t + \delta u_t$ or $w_t + \gamma u_t$ could exist outside of the unit sphere where all possible parameter configurations lie. In this case, we simply project the point back onto the sphere using said projection $P_W(\vec{v})$.

Yue and Joachims show that this algorithm has sublinear regret in T, the number of iterations. We note that the algorithm assumes that there exists a hidden reward function $R(w)$ that maps

system parameters w_t to a reward value which is smooth and strictly concave over the input space W.

Lastly, we would also like to give motivation behind δ and γ being different values. We need a δ that is sufficiently large that the comparison between a system parameterized by w_t and w'_t is meaningful. On the other hand, we may wish to take a smaller step in the direction of w'_t during our update step, as during a duel, we only score w_t against w'_t over the results on one query q_t. Having $\delta > \gamma$ allows us to get reward signal from meaningfully different points while also updating our belief of the best point w_{best} gradually.

Sparring EXP4

Zoghi et al. 2015 propose one algorithm for this problem — sparring EXP4, which duels two traditional EXP4 - algorithms. The (traditional) EXP4 algorithm solves the traditional contextual bandits — the case where we can directly observe a reward for a choice of bandit given a context. The EXP4 algorithm embeds each bandit as a vector. When the algorithm sees the context (called 'advice' in this formulation), it produces a probability distribution over the choices based on an adjusted softmax function on the inner product between the context and the bandit vectors. The probability function is different from a softmax as we assign some minimum probability that any action gets chosen to enforce exploration. A reward is then observed for the choice and propagated back through the embedding of the chosen bandit.

Sparring EXP4 runs two instances of the EXP4 algorithm against each other. Each EXP4 instance samples an action given a context, and then these choices are 'dueled' against each other. Instead of directly observing a reward, as for traditional EXP4, we instead observe two converse reward — a positive reward for the choice that won the duel and a negative reward to the choice that lost. The reward is proportional to the degree to which the bandit wins the duel, i.e. how likely the bandit is to be preferred over the other when users are queried for binary preferences. Like in traditional EXP4, the reward or negative reward is then propagated back through the representations of the bandits.

4.1.3.4 Feel-good Thompson sampling

This algorithm is a solution for the contextual dueling bandit setting, and tries to minimize cumulative average regret (= find WHAT WINNER?!Von Neumann???):

$$\text{Regret}(T) := \sum_{t=1}^{T} \left[r_*(x_t, a_t^*) - \frac{r_*(x_t, a_t^1) + r_*(x_t, a_t^2)}{2} \right],$$

where $r_*(x_t, a_t)$ is the true, hidden reward function of a context x_t and action a_t. Thompson sampling is an iterative process of receiving preference over two actions, each maximizing a different approximation of the reward function based on past data and adding this new information to the data.

Finding good approximations of the reward function at time t is done by sampling two reward function parameters $\theta_t^{j=1}$ and $\theta_t^{j=2}$ from a posterior distribution based on all previous data $p_j(\cdot \mid S_{t-1})$. This posterior distribution is proportional to the multiplication of the prior and the likelihood function, which is a Gaussian in standard Thompson sampling. In Feel-Good Thompson sampling, an additional term called "Feel-good exploration" encourages parameters θ with a large maximum reward in previous rounds. This change to the likelihood function may increase probabilities in uncertain areas, thus exploring those regions. All that's left is to select an action maximizing each reward function approximation and receive a preference y_t on one of them to add the new information to the dataset(Zhang 2021).

Initialize $S_0 = \emptyset$. Receive prompt x_t and action space $\mathcal{A}_t$. Sample model parameter θ_t^j from the posterior distribution $p^j(\cdot \mid S_{t-1})$ Select response $a_t^j = \arg\max_{a \in \mathcal{A}_t} \langle \theta_t^j, \phi(x_t, a) \rangle$. Receive preference y_t. Update dataset $S_t \leftarrow S_{t-1} \cup \{(x_t, a_t^1, a_t^2, y_t)\}$.

4.1.4 Applications

There are many applications where contextual bandits are used. Many of these applications can utilize human preferences. One particular application illustrates the benefits a contextual bandit would have over a multi-armed bandit: a website deciding which app to show someone visiting the website. A multi-armed bandit might decide to show someone an ad for a swimsuit because the swimsuit ads have gotten the most user clicks (which indicates human preference). A contextual bandit might choose differently, however. A contextual bandit will also take into account the context, which in this case might mean information about the user (location, previously visited pages, and device information). If it discovers the user lives in a cold environment, for example, it might suggest a sweater ad for the user instead and get a better chance of a click. There are many more examples of where contextual bandits can be applied. They can be applied in other web applications, such as to optimize search results, medical applications, such as how much of a medication to prescribe based on a patient's history, and gaming applications, such as basing moves off of the state of a chess board to try to win. In each of the above examples, human feedback could have been introduced during training and leveraged to learn a reward function.

We explored different versions of bandits that address the exploration-exploitation trade-off in various real-world scenarios. These models have been employed across various fields, including but not limited to healthcare, finance, dynamic pricing, and anomaly detection. This section provides a deep dive into some real-world applications, emphasizing the value and advancements achieved by incorporating bandit methodologies. The content of this section draws upon the findings from the survey cited in reference (Bouneffouf, Rish, and Aggarwal 2020).

In healthcare, researchers have been applying bandits to address challenges in clinical trials and behavioral modeling (Bouneffouf, Rish, and Cecchi 2017; Bastani and Bayati 2020). One of the examples is drug dosing. Warfarin, an oral anticoagulant, has traditionally been administered using fixed dosing protocols. Physicians would then make subsequent adjustments based on the patient's emerging symptoms. Nonetheless, inaccuracies in the initial dosage—whether

too low or too high—can lead to serious complications like strokes and internal bleeding. In a pivotal study, researchers in (Bastani and Bayati 2020) modeled the Warfarin initial dosing as a contextual bandit problem to assign dosages to individual patients appropriately based on their medication history. Their contributions include the adaptation of the LASSO estimator to the bandit setting, achieving a theoretical regret bound of $O({s_0}^2 \log^2(dT)$, where d represents the number of covariates, $s_0 << d$ signifies the number of pertinent covariates, and T indicates the total number of users. Additionally, they conducted empirical experiments to validate the robustness of their methodology.

Within the finance sector, bandits have been instrumental in reshaping the landscape of portfolio optimization. Portfolio optimization is an approach to designing a portfolio based on the investor's return and risk criteria, which fits the exploration-exploitation nature of the bandit problems. (Shen et al. 2015) utilized multi-armed bandits to exploit correlations between the instruments. They constructed orthogonal portfolios and integrated them with the UCB policy to achieve a cumulative regret bound of $\frac{8n}{\Delta*} \ln(m) + 5n$, where n, m, and $\Delta*$ denotes the number of available assets, total time steps, and the gap between the best-expected reward and the expected reward. On the other hand, (Huo and Fu 2017) focused on risk-awareness online portfolio optimization by incorporating a compute of the minimum spanning tree in the bipartite graph, which encodes a combination of financial institutions and assets that helps diversify and reduce exposure to systematic risk during the financial crisis.

Dynamic pricing, also known as demand-based pricing, refers to the strategy of setting flexible prices for products or services based on current market demands. The application of bandits in dynamic pricing offers a systematic approach to making real-time pricing decisions while balancing the trade-off between exploring new price points and exploiting known optimal prices. (Misra, Schwartz, and Abernethy 2019) proposed a policy where the company has only incomplete demand information. They derived an algorithm that balances immediate and future profits by combining multi-armed bandits with partial identification of consumer demand from economic theory.

are essential components of numerous online platforms, guiding users through vast content landscapes to deliver tailored suggestions. These systems are instrumental in platforms like e-commerce sites, streaming platforms, and social media networks. However, the challenge of effectively recommending items to users is non-trivial, given the dynamic nature of user preferences and the vast amount of content available.

One of the most significant challenges in recommendation systems is the "cold start" problem. This issue arises when a new user joins a platform, and the system has limited or no information about the user's preferences. Traditional recommendation algorithms struggle in such scenarios since they rely on historical user-item interactions. As discussed in (Zhou et al. 2017), the bandit setting is particularly suitable for large-scale recommender systems with a vast number of items. By continuously exploring user preferences and exploiting known interactions, bandit-based recommender systems can quickly adapt to new users, ensuring relevant recommendations in a few interactions. The continuous exploration inherent in bandit approaches also means that as a user's preferences evolve, the system can adapt, ensuring that recommendations remain relevant. Recommending content that is up to date is also another important

aspect of a recommendation system. In (Bouneffouf, Bouzeghoub, and Gançarski 2012), the concept of "freshness" in content is explored through the lens of the bandit problem. The Freshness-Aware Thompson Sampling algorithm introduced in this study aims to manage the recommendation of fresh documents according to the user's risk of the situation.

Dialogue systems, often termed conversational agents or chatbots, aim to simulate human-like conversations with users. These systems are deployed across various platforms, including customer support, virtual assistants, and entertainment applications, and they are crucial for enhancing user experience and engagement. Response selection is fundamental to creating a natural and coherent dialogue flow. Traditional dialogue systems rely on a predefined set of responses or rules, which can make interactions feel scripted and inauthentic. In (Liu et al. 2018), the authors proposed a contextual multi-armed bandit model for online learning of response selection. Specifically, they utilized bidirectional LSTM to produce the distributed representations of a dialogue context and responses and customized the Thompson sampling method.

To create a more engaging and dynamic interaction, there's a growing interest in developing pro-active dialogue systems that can initiate conversations without user initiation. (perez and Silander 2018) proposed a novel approach to this challenge with contextual bandits. By introducing memory models into the bandit framework, the system can recall past interactions, making its proactive responses more contextually relevant. Their contributions include the Contextual Attentive Memory Network, which implements a differentiable attention mechanism over past interactions.

(Upadhyay et al. 2019) addressed the challenge of orchestrating multiple independently trained dialogue agents or skills in a unified system. They attempted online posterior dialogue orchestration, defining it as selecting the most suitable subset of skills in response to a user's input, which studying a context-attentive bandit model that operates under a skill execution budget, ensuring efficient and accurate response selection.

Anomaly detection refers to the task of identifying samples that behave differently from the majority. In (Ding, Li, and Liu 2019), the authors delve into anomaly detection in an interactive setting, allowing the system to actively engage with human experts through a limited number of queries about genuine anomalies. The goal is to present as many true anomalies to the human expert as possible after a fixed query budget is used up. They applied the multi-armed contextual bandit framework to address this issue. This algorithm adeptly integrates both nodal attributes and node dependencies into a unified model, efficiently managing the exploration-exploitation trade-off during anomaly queries.

There are many challenges associated with contextual bandits. The first challenge is that each action only reveals the reward for that particular action. Therefore, the algorithm has to work with incomplete information. This leads to the dilemma of exploitation versus exploration: when should the algorithm choose the best-known option versus trying new options for potentially better outcomes? Another significant challenge for contextual bandits is using context effectively. The context the environment gives needs to be explored to figure out which action is best for each context.

The overarching goal in systems designed for recommending options of high value to users is to achieve an optimal balance between exploration and exploitation. This dual approach is crucial in environments where user preferences and needs are dynamic and diverse. Exploration refers to the process of seeking out new options, learning about untried possibilities, and gathering fresh information that could lead to high-value recommendations. In contrast, exploitation involves utilizing existing knowledge and past experiences to recommend the best options currently known. This balance is key to maintaining a system that continuously adapts to changing user preferences while ensuring the reliability of its recommendations.

A key observation in such systems is the dual role of users as both producers and consumers of information. Each user's experience contributes valuable data that informs future recommendations for others. For instance, platforms like Waze, Netflix, and Trip Advisor rely heavily on user input and feedback. Waze uses real-time traffic data from drivers to recommend optimal routes; Netflix suggests movies and shows based on viewing histories and ratings; Trip Advisor relies on traveler reviews to guide future tourists. In these examples, the balance between gathering new information (exploration) and recommending the best-known options (exploitation) is dynamically managed to enhance user experience and satisfaction. This approach underscores the importance of user engagement in systems where monetary incentives are not (or can not be) the primary driver.

Recommendation systems often face the challenge of overcoming user biases that can lead to a narrow exploration of options. Users come with preconceived notions and preferences, which can cause them to overlook potentially valuable options that initially appear inferior or unaligned with their interests. This predisposition can significantly limit the effectiveness of recommendation systems, as users might miss out on high-value choices simply due to their existing biases.

To counteract this, it is crucial for recommendation systems to actively incentivize exploration among users. One innovative approach to achieve this is through the strategic use of **information asymmetry**. By controlling and selectively presenting information, these systems can guide users to explore options they might not typically consider. This method aims to reveal the true potential of various options by nudging users out of their comfort zones and encouraging a broader exploration of available choices. An important note here is that the system is not lying to users - it only selectively reveals information it has.

The concept of incentivizing exploration becomes even more complex when considering different types of users. For instance, systems often encounter short-lived users who have little to gain from contributing to the system's learning process, as their interactions are infrequent or based on immediate needs. Similarly, some users may operate under a 'greedy' principle, primarily seeking immediate gratification rather than contributing to the long-term accuracy and effectiveness of the system. In such scenarios, managing information asymmetry can be a powerful tool. By selectively revealing information, recommendation systems can create a sense of novelty and interest, prompting even the most transient or self-interested users to engage in exploration, thereby enhancing the system's overall knowledge base and recommendation quality.

4.1.5 Incentive-Compatible Online Learning

To address this problem, we seek to create a model. But first, it is useful to outline the key criteria that our model must achieve.

- The *core* of the model revolves around repeated interactions between a planner (the system) and multiple agents (the users). Each agent, upon arrival in the system, is presented with a set of available options to choose from. These options could vary widely depending on the application of the model, such as routes in a transportation network, a selection of hotels in a travel booking system, or even entertainment choices in a streaming service.
- The *interaction process* is straightforward but crucial: agents arrive, select an action from the provided options, and then report feedback based on their experience. This feedback is vital as it forms the basis upon which the planner improves and evolves its recommendations. The agents in this model are considered strategic; they aim to maximize their reward based on the information available to them. This aspect of the model acknowledges the real-world scenario where users are typically self-interested and seek to optimize their own outcomes.
- The *planner*, on the other hand, has a broader objective. It aims to learn which alternatives are best in a given context and works to maximize the overall welfare of all agents. This involves a complex balancing act: the planner must accurately interpret feedback from a diverse set of agents, each with their own preferences and biases, and use this information to refine and improve the set of options available. The ultimate goal of the planner is to create a dynamic, responsive system that not only caters to the immediate needs of individual agents but also enhances the collective experience over time, leading to a continually improving recommendation ecosystem.

Let's break this up into a set of tangible research questions that we seek to answer in the rest of this chapter.

- **Planner Limitations**: We seek to address the inherent limitations faced by the planner, particularly in scenarios where monetary transfers are not an option, and the only tool at its disposal is the control over the flow of information between agents. This inquiry aims to understand the extent to which these limitations impact the planner's ability to effectively guide and influence agent behavior.
- **Inducing Exploration**: A critical question is whether the planner can successfully induce exploration among agents, especially in the absence of financial incentives. This involves investigating strategies to encourage users to try less obvious or popular options, thus broadening the scope of feedback and enhancing the system's ability to learn and identify the best alternatives.
- **Rate of Learning**: Another essential research area is understanding the rate at which the planner learns from agent interactions. This encompasses examining how different agent incentives, their willingness to explore, and their feedback impact the speed and efficiency with which the planner can identify optimal recommendations.

– **Model Extensions**: The model can be extended in several directions, each raising its own set of questions.

1. Multiple Agents with Interconnected Payoffs: When multiple agents arrive simultaneously, their choices and payoffs become interconnected, resembling a game. The research question here focuses on how these interdependencies affect individual and collective decision-making.
2. Planner with Arbitrary Objective Function: Investigating scenarios where the planner operates under an arbitrary objective function, which might not align with maximizing overall welfare or learning the best alternative.
3. Observed Heterogeneity Among Agents: This involves situations where differences among agents are observable and known, akin to contextual bandits in machine learning. The research question revolves around how these observable traits can be used to tailor recommendations more effectively.
4. Unobserved Heterogeneity Among Agents: This aspect delves into scenarios where differences among agents are not directly observable, necessitating the use of causal inference techniques to understand and cater to diverse user needs.

Bayesian Incentive-Compatible Bandit Model

In this section, we introduce the main model of study in this chapter (Mansour, Slivkins, and Syrgkanis 2019; Mansour et al. 2021). In our setup, there is a "planner," which aims to increase exploration, and many independent"agents," which will act selfishly (in a way that they believe will maximize their individual reward).

Under our model shown in Figure 1.1, there are K possible actions that all users can take, and each action has some mean reward $\mu_i \in [0, 1]$. In addition, there is a common prior belief on each μ_i across all users.. The T agents, or users, will arrive sequentially. As the t'th user arrives, they are recommended an action I_t by the planner, which they are free to follow or not follow. After taking whichever action they choose, the user experiences some realized reward $r_i \in [0, 1]$, which is stochastic i.i.d. with mean μ_i, and reports this reward back to the planner.

So far, the model we have defined is equivalent to a multi-armed bandit model, which we have seen earlier in this chapter (1). Under this model, well-known results in economics, operations research and computer science show that $O(\sqrt{T})$ regret is achievable (Russo and Roy 2015; Auer, Cesa-Bianchi, and Fischer 2002; Lai and Robbins 1985) with algorithms such as Thompson sampling and UCB.

However, our agents are strategic and aim to maximize their own rewards. If they observe the rewards gained from actions taken by other previous users, they will simply take the action they believe will yield the highest reward given the previous actions; they would prefer to benefit from exploration done by other users rather than take the risk of exploring themselves.

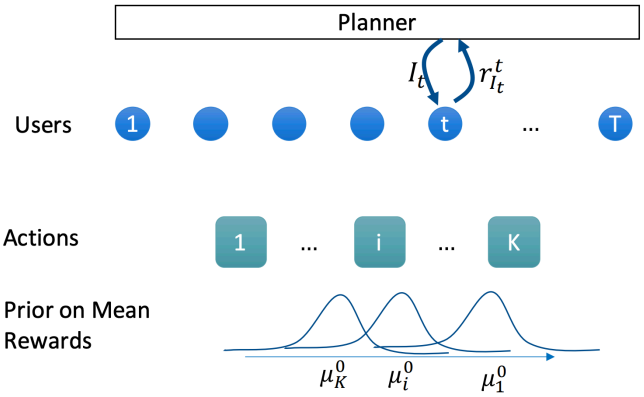


Figure 4.2
Planner–agent setup

Therefore, exploration on an individual level, which the planner would like to facilitate, is not guaranteed under this paradigm.

In light of this, we also require that our model satisfy **incentive compatibility**, or that taking the action recommended by the planner has an expected utility that is as high as any other action the agent could take. Formally,

$$\forall i : E[\mu_i | I_t = i] \geq E[\mu_{i'} | I_t = i].$$

Note that this incentivizes the agents to actually take the actions recommended by the planner; if incentive compatibility is not satisfied, agents would simply ignore the planner and take whatever action they think will lead to the highest reward.

At a high level, the key to achieving incentive compatibility while still creating a policy for the planner that facilitates exploration is information asymmetry. Under this paradigm, the users only have access to their previous recommendations, actions, and rewards, and not to the recommendations, actions, and rewards of other users. Therefore, they are unsure of whether, after other users take certain actions and receive certain rewards, arms that they might have initially considered worse in practice outperform arms that they initially considered better. Only the planner has access to the previous actions and rewards of all users; the user only has access to their own recommendations and overall knowledge of the planner's policy.

The main question we aim to answer for the rest of this section is, given this new constraint of incentive compatibility, is $O(\sqrt{T})$ regret still achievable? We illustrate such an algorithm in the following.

Black-box Reduction Algorithm

The main result for this chapter is a **black-box reduction** algorithm to turn any bandit algorithm into an *incentive compatible* one, with only a constant increase in Bayesian regret. Since, as

mentioned earlier, there are bandit algorithms with $O(\sqrt{T})$ Bayesian regret, black-box reduction will also allow us to get incentive-compatible algorithms with $O(\sqrt{T})$ regret. The idea of black-box reduction will be to simulate T steps of any bandit algorithm in an incentive-compatible way in cT steps. This allows us to design incentive-compatible recommendation systems by using any bandit algorithm and then adapting it.

Consider the following setting: there are two possible actions, A_1 and A_2. Assume the setting of **deterministic rewards**, where action 1 has reward μ_1 with prior $U[1/3, 1]$ and mean $\mathbb{E}[\mu_1] = 2/3$, and action 2 has reward μ_2 with prior $U[0, 1]$ and mean $\mathbb{E}[\mu_2] = 1/2$. Without the planner intervention and with full observability, users would simply always pick A_1, so how can the planner *incentivize* users to play A_2?

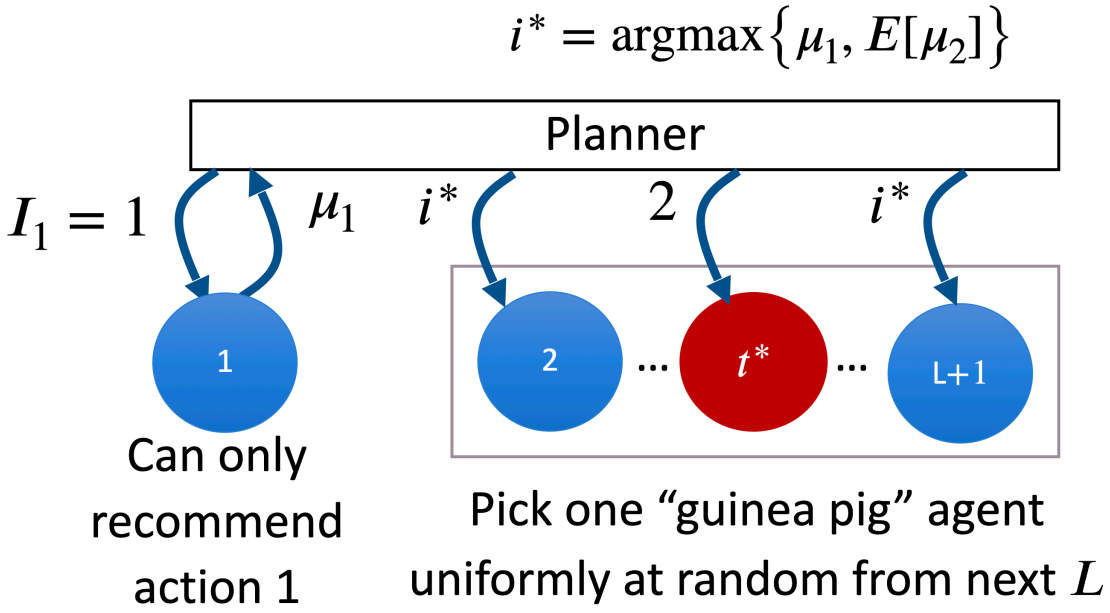


Figure 4.3
Illustration of black-box reduction algorithm when we have deterministic rewards.

The key insight is going to be to *hide exploration in a pool of exploitation*. The users are only going to receive a recommendation from the planner, and no other observations. After deterministically recommending the action with the highest expected reward (A_1), the planner will pick one **guinea pig** to recommend the exploratory action of A_2. The users don't know whether they are the guinea pig, so intuitively, as long as the planner picks guinea pigs uniformly at random and at low enough frequencies, the optimal decision for the users is still to follow the planner's recommendation, even if it might go against their interest.

The planner will pick the user who will be recommended the exploratory action uniformly at random from the L users that come after the first one (which deterministically gets recommended the exploitation action). Under this setting (illustrated in Figure 1.2), it is optimal for users to always follow the option that is recommended for them. More formally, if I_t is the

recommendation that a user receives at time t, then we have that:

$$\begin{aligned}\mathbb{E}[\mu_1 - \mu_2 | I_t = 2] Pr[I_t = 2] &= \frac{1}{L}(\mu_1 - \mu_2) \quad \text{(Gains if you are the unlucky guinea pig)} \\ &+ (1 - \frac{1}{L})\mathbb{E}[\mu_1 - \mu_2 | \mu_1 < \mu_2] Pr[\mu_1 < \mu_2] \quad \text{(Loss if you are not and } \mu_1 < \mu_2) \\ &\leq 0\end{aligned}$$

This holds when $L \geq 12$. It means that the gains from not taking the recommended action are *negative*, which implies that users should always take the recommendation.

So far we have considered the case where rewards are deterministic, but what about **stochastic rewards**? We are now going to consider the case where rewards are independent and identically distributed from some distribution, and where each action A_i has some reward distribution $r_i^t \sim D_i, \mathbb{E}[r_i^t] = \mu_i$. Back to the case where there are only two actions, we are going to adapt the prior algorithm of guinea pig-picking to the stochastic reward setting. Since one reward observation is not enough to fully know μ_1 anymore, we'll instead observe the outcome of the first action M times to form a strong posterior $\mathbb{E}[\mu_1 | r_1^1, \ldots r_1^M]$.

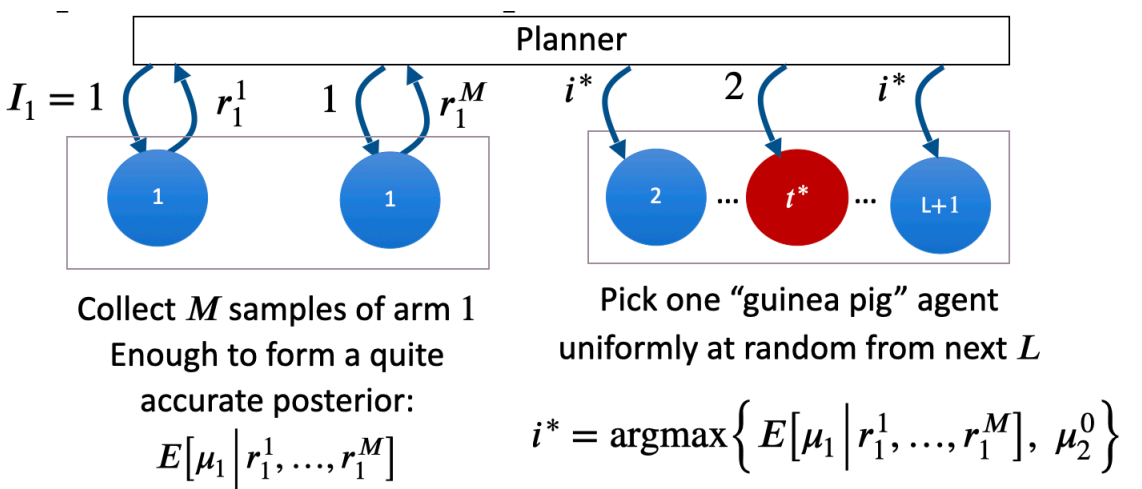


Figure 4.4
Illustration of black-box reduction algorithm when we have stochastic rewards.

Figure 1.3 illustrates the algorithm that we can use with stochastic rewards when there are two actions. Similarly, as before, we pick one guinea pig uniformly at random from the next L users and use the reward we get as the exploratory signal.
In a very similar manner, we can generalize this algorithm from always having two actions to the general multi-armed bandit problem. Now suppose we have a general multi-armed bandit algorithm A. We will wrap this algorithm around our black box reduction algorithm to make it incentive-compatible.

As Figure 1.4 shows, we wrap every decision that A would make by exactly $L - 1$ recommendations of the action believed to be the best so far. This guarantees that the expected rewards for the users that are not chosen as guinea pigs are at least as good as A's reward at phase n.

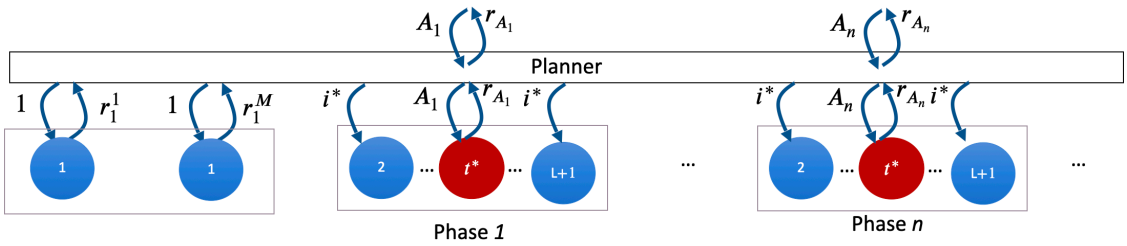


Figure 4.5
Illustration of black-box reduction algorithm for the general multi-armed bandit case.

4.2 Preferential Bayesian Optimization

The traditional Bayesian optimization (BO) problem is described as follows. There is a black-box objective function $g : \mathcal{X} \to \mathfrak{R}$ defined on a bounded subset $\mathcal{X} \subseteq \mathfrak{R}^q$ such that direct queries to the function are expensive or not possible. However, we would like to solve the global optimization problem of finding $\mathbf{x}_{\min} = \arg\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x})$. This is highly analogous to modeling human preferences, since it is the case that direct access to a human's latent preference function is not possible but we would still like to find its optimum, such as in A/B tests or recommender systems.

We approach this problem for human preferences with *Preferential Bayesian Optimization* (PBO), as the key difference is that we are able to query the preference function through pairwise comparisons of data points, i.e. *duels*. This is a form of indirect observation of the objective function, which models real-world scenarios closely: we commonly need to to optimize a function via data about preferences. With humans, it has been demonstrated that we are better at evaluating differences rather than absolute magnitudes (Kahneman and Tversky 1979) and therefore PBO models can be applied in various contexts.

4.2.1 Problem statement

The problem of finding the optimum of a latent preference function defined on $\mathcal{X}$ can be reduced to determining a sequence of duels on $\mathcal{X} \times \mathcal{X}$. From each duel $[\mathbf{x}, \mathbf{x}'] \in \mathcal{X} \times \mathcal{X}$ we obtain binary feedback $\{0, 1\}$ indicating whether or not $\mathbf{x}$ is preferred over $\mathbf{x}'$ ($g(\mathbf{x}) < g(\mathbf{x}')$). We consider that $\mathbf{x}$ is the winner of the duel if the output is $\{1\}$ and that $\mathbf{x}'$ wins the duel if the output is $\{0\}$. The aim is to find $\mathbf{x}_{\min}$ by reducing as much as possible the number of queried duels.

The key idea in PBO is to learn a preference function in the space of duels using a Gaussian process. We define a joint reward $f([\mathbf{x}, \mathbf{x}'])$ on each duel which is never directly observed. Instead, the feedback we obtain after each pair is a binary output $y \in \{0, 1\}$ indicating which of the two inputs is preferred. One definition of f we will use (though others are possible) is $f([\mathbf{x}, \mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x})$. The more $\mathbf{x}'$ is preferred over $\mathbf{x}$, the bigger the reward.

We define the model of preference using a Bernoulli likelihood, where $p(y = 1 \mid [\mathbf{x}, \mathbf{x}']) = \pi_f([\mathbf{x}, \mathbf{x}'])$ and $p(y = 0 \mid [\mathbf{x}, \mathbf{x}']) = \pi_f([\mathbf{x}', \mathbf{x}])$ for some inverse link function $\pi : \mathfrak{R} \times \mathfrak{R} \to$

$[0,1]$. π_f has the property that $\pi_f([\mathbf{x}',\mathbf{x}]) = 1 - \pi_f([\mathbf{x},\mathbf{x}'])$. A natural choice for π_f is the logistic function

$$\pi_f([\mathbf{x},\mathbf{x}']) = \sigma(f([\mathbf{x},\mathbf{x}'])) = \frac{1}{1+e^{-f([\mathbf{x},\mathbf{x}'])}},$$

but others are possible. Therefore we have that for any duel $[\mathbf{x},\mathbf{x}']$ in which $g(\mathbf{x}) \leq g(\mathbf{x}')$ it holds that $\pi_f([\mathbf{x},\mathbf{x}']) \geq 0.5$. π_f is a preference function that maps each query $[\mathbf{x},\mathbf{x}']$ to the probability of having a preference on the left input $\mathbf{x}$ over the right input $\mathbf{x}'$.

When we marginalize over the right input $\mathbf{x}'$ of f (is this correct?), the global minimum of f in $\mathcal{X}$ coincides with $\mathbf{x}_{\text{min}}$. We also introduce the definition of the *Copeland score function* for a point $\mathbf{x}$ as

$$S(\mathbf{x}) = \text{Vol}(\mathcal{X})^{-1} \int_{\mathcal{X}} \mathbb{I}_{\{\pi_f([\mathbf{x},\mathbf{x}'])\geq 0.5\}} d\mathbf{x}'$$

where $\text{Vol}(\mathcal{X}) = \int_{\mathcal{X}} d\mathbf{x}'$ is a normalizing constant that bounds $S(\mathbf{x})$ in the interval $[0,1]$. If $\mathcal{X}$ is a finite set, the Copeland score is simply the proportion of duels that a certain element $\mathbf{x}$ will win with probability larger than 0.5. A soft variant we will use instead of the Copeland score is the *soft-Copeland score*, defined as

$$C(\mathbf{x}) = \text{Vol}(\mathcal{X})^{-1} \int_{\mathcal{X}} \pi_f([\mathbf{x},\mathbf{x}'])\, d\mathbf{x}'$$

where the probability function π_f is integrated over $\mathcal{X}$. This score aims to capture the average probability of $\mathbf{x}$ being the winner of a duel.

We define the *Condorcet winner* $\mathbf{x}_c$ as the point with maximal soft-Copeland score. Note that this corresponds to the global minimum of f, since the defining integral takes maximum value for points $\mathbf{x} \in \mathcal{X}$ where $f([\mathbf{x},\mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x}) > 0$ or all $\mathbf{x}'$, occurring only if $\mathbf{x}_c$ is a minimum of f. Therefore, if the preference function π_f can be learned by observing the results of duels then our optimization problem of finding the minimum of f can be solved by finding the Condorcet winner of the Copeland score.

4.2.2 Acquisition Functions

We describe several acquisition functions for sequential learning of the Condorcet winner. Our dataset $\mathcal{D} = \{[\mathbf{x}_i,\mathbf{x}_i'], y_i\}_{i=1}^N$ represents the N duels that have been performed so far. We aim to define a sequential policy $\alpha([\mathbf{x},\mathbf{x}']; \mathcal{D}_j, \theta)$ for querying duels, where θ is a vector of model hyper-parameters, in order to find the minimum of the latent function g as quickly as possible. Using Gaussian processes (GP) for classification with our dataset $\mathcal{D}$ allows us to perform inference over f and π_f.

Pure Exploration

The output variable $y_\star$ of a prediction follows a Bernoulli distribution with probability given by the preference function π_f. To carry out exploration as a policy, one method is to search

for the duel where GP is most uncertain about the probability of the outcome (has the highest variance of $\sigma(f_\star)$), which is the result of transforming out epistemic uncertainty about f, modeled by a GP, through the logistic function. The first order moment of this distribution coincides with the expectation of $y_\star$ but its variance is

$$\begin{aligned}\mathbb{V}[\sigma(f_\star)] &= \int (\sigma(f_\star) - \mathbb{E}[\sigma(f_\star)])^2 \, p(f_\star \mid \mathcal{D}, [\mathbf{x}, \mathbf{x}']) \, df_\star \\ &= \int \sigma(f_\star)^2 \, p(f_\star \mid \mathcal{D}, [\mathbf{x}, \mathbf{x}']) \, df_\star - \mathbb{E}[\sigma(f_\star)]^2\end{aligned}$$

which explicitly takes into account the uncertainty over f. Hence, pure exploration of duels space can be carried out by maximizing

$$\alpha_{\text{PE}}\left([\mathbf{x}, \mathbf{x}'] \mid \mathcal{D}_j\right) = \mathbb{V}\left[\sigma(f_\star) \,|[\mathbf{x}_\star, \mathbf{x}'_\star]| \, \mathcal{D}_j\right].$$

Note that in this case, duels that have been already visited will have a lower chance of being visited again even in cases in which the objective takes similar values in both players. In practice, this acquisition functions requires computation of an intractable integral, that we approximate using Monte-Carlo.

Principled Optimistic Preferential Bayesian Optimization (POP-BO)

In a slightly modified problem setup (Xu et al. 2024), the algorithm tries to solve for the MLE $\hat{g}$ and its confidence set $\mathcal{B}_g$ where g is the ground truth black-box function. Assumptions include that g is a member of a reproducing kernel Hilbert space (RKHS) $\mathcal{H}_k$ for some kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, and $\|g\|_k \leq B$ so that $\mathcal{B}_g = \{\tilde{g} \in \mathcal{H}_k \mid \|\tilde{g}\|_k \leq B\}$. Similarly defining $f([\mathbf{x}, \mathbf{x}']) = g(\mathbf{x}') - g(\mathbf{x})$, we model the preference function with a Bernoulli distribution as in Equation [eq:bernoulli_pref] and also assume that probabilities follow the Bradley-Terry model, i.e.

$$\pi_f([\mathbf{x}, \mathbf{x}']) = \sigma(f([\mathbf{x}, \mathbf{x}'])) = \frac{e^{g(\mathbf{x})}}{e^{g(\mathbf{x})} + e^{g(\mathbf{x}')}}$$

The update rule for MLE $\hat{g}$ is (equation 8,6,5)

$$\begin{aligned}\hat{g}_t^{\text{MLE}} &:= \arg\max_{\tilde{g} \in \mathcal{B}_g^t} \ell_t(\tilde{g}) \\ \ell_t(\tilde{g}) &:= \log \prod_{\tau=1}^{t} y_\tau \pi_{\tilde{f}}([\mathbf{x} \ , \mathbf{x}' \ ]) + (1 - y_\tau)\left(1 - \pi_{\tilde{f}}([\mathbf{x} \ , \mathbf{x}' \ ])\right) \\ &= \sum_{\tau=1}^{t} \log\left(\frac{e^{\tilde{g}(\mathbf{x} \ )} y_\tau + e^{\tilde{g}(\mathbf{x}' \ )}(1 - y_\tau)}{e^{\tilde{g}(\mathbf{x} \ )} + e^{\tilde{g}(\mathbf{x}' \ )}}\right) \\ &= \sum_{\tau=1}^{t} (\tilde{g}(\mathbf{x} \ ) y_\tau + \tilde{g}(\mathbf{x}' \ )(1 - y_\tau)) - \sum_{\tau=1}^{t} \log\left(e^{\tilde{g}(\mathbf{x} \ )} + e^{\tilde{g}(\mathbf{x}' \ )}\right)\end{aligned}$$

(Eq 22 shows how to represent this as a convex optimisation problem so that it can be solved)

The update rule for the confidence set $\mathcal{B}_f^{t+1}$ is, (eq 9, 10?)

$$\begin{aligned}&\forall \epsilon, \delta > 0\\ &\mathcal{B}_g^{t+1} := \left\{\tilde{g} \in \mathcal{B}_g \mid \ell_t(\tilde{g}) \geq \ell_t\left(\hat{g}_t^{\text{MLE}}\right) - \beta_1(\epsilon, \delta, t)\right\}\end{aligned}$$

where

$$\beta_1(\epsilon, \delta, t) := \sqrt{32tB^2 \log \frac{\pi^2 t^2 \mathcal{N}\left(\mathcal{B}_f, \epsilon, \|\cdot\|_\infty\right)}{6\delta}} + C_L \epsilon t = \mathcal{O}\left(\sqrt{t \log \frac{t\mathcal{N}\left(\mathcal{B}_f, \epsilon, \|\cdot\|_\infty\right)}{\delta}} + \epsilon t\right),$$

with C_L a constant independent of δ, t and ϵ. ϵ is typically chosen to be $1/T$, where T is the running horizon of the algorithm. This satisfies the theorem that,

$$\mathbb{P}\left(g \in \mathcal{B}_g^{t+1}, \forall t \geq 1\right) \geq 1 - \delta.$$

Intuitively, the confidence set $\mathcal{B}_g^{t+1}$ includes the functions with the log-likelihood value that is only 'a little worse' than the maximum likelihood estimator, and the theorem states that $\mathcal{B}_g^{t+1}$ contains the ground-truth function g with high probability.

Inner level optimization in Line 4 of the algorithm can also be represented as a convex optimisation problem so that it can be solved, Eq 24, 25. The outer optimisation can be solved using grid search or Eq 26 for medium size problems.

Given the initial point $\mathbf{x_0} \in \mathcal{X}$ and set $\mathcal{B}_g^1 = \mathcal{B}_g$ Set the reference point $\mathbf{x_t'} = \mathbf{x_{t-1}}$ Compute $\mathbf{x_t} \in \arg\max_{\mathbf{x}\in\mathcal{X}} \max_{\tilde{g}\in\mathcal{B}_g^t} (\tilde{g}(\mathbf{x}) - \tilde{g}(\mathbf{x_t'}))$, with the inner optimal function denoted as $\tilde{g}_t$ Obtain the output of the duel y_t and append the new data point to $\mathcal{D}_t$ Update the maximum likelihood estimator $\hat{g}_t^{\text{MLE}}$ and the posterior confidence set $\mathcal{B}_g^{t+1}$.

qEUBO: Decision-Theoretic EUBO

qEUBO (Astudillo et al. 2023) derives an acquisition function that extends duels to $q > 2$ options which we call *queries*. Let $X = \left(\mathbf{x_1}, \ldots, \mathbf{x_q}\right) \in \mathcal{X}^q$ denote a query containing two points or more, and let $g : \mathcal{X} \to \mathfrak{R}$ be the latent preference function. Then after n user queries, we define the *expected utility of the best option* (qEUBO) as

$$\text{qEUBO}_n(X) = \mathbb{E}_n\left[\max\left\{g\left(x_1\right), \ldots, g\left(x_q\right)\right\}\right].$$

We now show that qEUBO is one-step Bayes optimal, meaning that each step chooses the query that maximises the expected utility received by the human. For a query $X \in \mathcal{X}^q$, let

$$V_n(X) = \mathbb{E}_n\left[\max_{x\in\mathbb{X}} \mathbb{E}_{n+1}[g(x)] \mid X_{n+1} = X\right].$$

Then V_n defines the expected utility received if an additional query $X_{n+1} = X$ is performed, and maximizing V_n is one-step Bayes optimal. Since $\max_{x \in \mathbb{X}} \mathbb{E}_n[f(x)]$ does not depend on X_{n+1}, we can also equivalently maximize

$$\mathbb{E}_n \left[\max_{x \in \mathbb{X}} \mathbb{E}_{n+1}[g(x)] - \max_{x \in \mathbb{X}} \mathbb{E}_n[g(x)] \mid X_{n+1} = X \right],$$

which takes the same form as the knowledge gradient acquisition function (Wu and Frazier 2018) in standard Bayesian optimization.

V_n involves a nested stochastic optimization task, while qEUBO is a much simpler policy. When human responses are noise-free, we are able to use qEUBO as a sufficient policy due to the following theorem:

$$\operatorname*{argmax}_{X \in \mathbb{X}^q} \text{qEUBO}_n(X) \subseteq \operatorname*{argmax}_{X \in \mathbb{X}^q} V_n(X).$$

Proof. Proof. For a query $X \in \mathcal{X}^q$, let $x^+(X, i) \in \operatorname{argmax}_{x \in \mathbb{X}} \mathbb{E}_n[g(x) \mid (X, i)]$ and define $X^+(X) = (x^+(X, 1), \ldots, x^+(X, q))$.

Claim 1 $V_n(X) \leq \text{qEUBO}_n\left(X^+(X)\right)$. We see that

$$\begin{aligned} V_n(X) &= \sum_{i=1}^{q} \mathbf{P}_n(r(X) = i) \mathbb{E}_n[g\left(x^+(X, i)\right)] \\ &\leq \sum_{i=1}^{q} \mathbf{P}_n(r(X) = i) \mathbb{E}_n[\max_{i=1,\ldots,q} g(x^+(X, i))] \\ &= \mathbb{E}_n\left[\max_{i=1,\ldots,q} g\left(x^+(X, i)\right)\right] \\ &= \text{qEUBO}_n\left(X^+(X)\right), \end{aligned}$$

as claimed.

Claim 2 $\text{qEUBO}_n(X) \leq V_n(X)$. For any given $X \in \mathbb{X}^q$ we have

$$\mathbb{E}_n\left[f\left(x_{r(X)}\right) \mid (X, r(X))\right] \leq \max_{x \in \mathbb{X}} \mathbb{E}_n[f(x) \mid (X, r(X))].$$

Since $f\left(x_{r(X)}\right) = \max_{i=1,\ldots,q} f\left(x_i\right)$, taking expectations over $r(X)$ on both sides obtains the required result.

Now building on the arguments above, let $X^* \in \operatorname{argmax}_{X \in \mathbb{X}^q} \text{qEUBO}_n(X)$ and suppose for contradiction that $X^* \notin \operatorname{argmax}_{X \in \mathbb{X}^q} V_n(X)$. Then, there exists $\widetilde{X} \in \mathbb{X}^q$ such that $V_n(\widetilde{X}) >$

$V_n(X^*)$. We have

$$\begin{aligned}\text{qEUBO}_n\left(X^+(\tilde{X})\right) &\geq V_n(\tilde{X}) \\ &> V_n(X^*) \\ &\geq \text{qEUBO}_n(X^*) \\ &\geq \text{qEUBO}_n\left(X^+(\tilde{X})\right).\end{aligned}$$

The first inequality follows from (1). The second inequality is due to our supposition for contradiction. The third inequality is due to (2). Finally, the fourth inequality holds since $X^* \in \text{argmax}_{X \in \mathbb{X}^q} \text{qEUBO}_n(X)$. This contradiction concludes the proof. □ □

Therefore a sufficient condition for following one-step Bayes optimality is by maximizing qEUBO_n.

In experiments that were ran comparing qEUBO to other state-of-the-art acquisition functions, qEUBO consistently outperformed on most problems and was closely followed by qEI and qTS. These results also extended to experiments with multiple options when $q > 2$. In fact, there is faster convergence in regret when using more options in human queries. [Prove Theorem 3: Regret analysis]

qEI: Batch Expected Improvement

$$\begin{aligned}\text{qEI} =& \mathbb{E}_{\mathbf{y}}\left[\left(\max_{i\in[1,\ldots,q]}\left(\mu_{\min}-y_i\right)\right)_+\right] \\ =& \sum_{i=1}^{q}\mathbb{E}_{\mathbf{y}}\left(\mu_{\min}-y_i \mid y_i \leq \mu_{\min}, y_i \leq y_j \forall j \neq i\right) \\ & p\left(y_i \leq \mu_{\min}, y_i \leq y_j \forall j \neq i\right).\end{aligned}$$

qTS: Batch Thompson Sampling

Initial data $\mathcal{D}_{\mathcal{J}(1)} = \{(\mathbf{x}_i, y_i)\}_{i\in\mathcal{J}(1)}$ Compute current posterior $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Sample θ from $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Select $k \leftarrow \arg\max_{j\notin\mathcal{J}(t)} \mathbb{E}[y_j \mid \mathbf{x}_j, \theta]$ Collect y_k by evaluating f at $\mathbf{x}_k$ $\mathcal{D}_{\mathcal{J}(t+1)} \leftarrow \mathcal{D}_{\mathcal{J}(t)} \cup \{(\mathbf{x}_k, y_k)\}$

Initial data $\mathcal{D}_{\mathcal{J}(1)} = \{\mathbf{x}_i, y_i\}_{i\in\mathcal{J}(1)}$, batch size S Compute current posterior $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Sample θ from $p(\theta \mid \mathcal{D}_{\mathcal{J}(t)})$ Select $k(s) \leftarrow \arg\max_{j\notin\mathcal{J}(t)} \mathbb{E}[y_j \mid \mathbf{x}_j, \theta]$ $\mathcal{D}_{\mathcal{J}(t+1)} = \mathcal{D}_{\mathcal{J}(t)} \cup \{\mathbf{x}_{k(s)}, y_{k(s)}\}_{s=1}^{S}$

4.2.3 Regret Analysis

qEUBO Regret

With the definition of Bayesian simple regret, we have that qEUBO converges to zero at a rate of $o(1/n)$, i.e.

$$\mathbb{E}\left[f\left(x^*\right) - f\left(\hat{x}_n^*\right)\right] = o(1/n)$$

where $x^* = \text{argmax}_{x \in \mathbb{X}} f(x)$ and $\hat{x}_n^* \in \text{argmax}_{x \in \mathbb{X}} \mathbb{E}_n[f(x)]$.

This theorem holds under the following assumptions:

1. f **is injective** $\mathbf{P}(f(x) = f(y)) = 0$ for any $x, y \in \mathbb{X}$ with $x \neq y$.
2. f **represents the preferred option** $\exists a > 1/2$ s.t. $\mathbf{P}\left(r(X) \in \text{argmax}_{i=1,\ldots,2} f\left(x_i\right) \mid f(X)\right) \geq a \forall\, X = (x_1, x_2) \in \mathbb{X}^2$ with $x_1 \neq x_2$ almost surely under the prior on f.
3. **Expected difference in utility is proportional to probability of greater utility** $\exists \Delta \geq \delta > 0$ s.t. $\forall \mathcal{D}^{(n)}and\forall x, y \in \mathbb{X}$ (potentially depending on $\mathcal{D}^{(n)}$),

   $$\delta \mathbf{P}^{(n)}(f(x) > f(y)) \leq \mathbb{E}^{(n)}\left[\{f(x) - f(y)\}^+\right] \leq \Delta \mathbf{P}^{(n)}(f(x) > f(y))$$

   almost surely under the prior on f.

Further lemmas leading to a proof of Theorem [th:quebo_regret] is given in (Astudillo et al. 2023) Section B.

qEI Regret

The following theorem shows that, under the same assumptions used for qEUBO regret, simple regret of qEI can fail to converge to 0.

There exists a problem instance (i.e., $\mathbb{X}$ and Bayesian prior distribution over f) satisfying the assumptions described in Theorem [th:quebo_regret] such that if the sequence of queries is chosen by maximizing qEI, then $\mathbb{E}\left[f\left(x^*\right) - f\left(\hat{x}_n^*\right)\right] \geq R$ for all n, for a constant $R > 0$.

Proof. Proof. Let $X = \{1, 2, 3, 4\}$ and consider the functions $f_i : X \to R$, for $i = 1, 2, 3, 4$, given by $f_i(1) = -1$ and $f_i(2) = 0$ for all i, and

$$f_1(x) = \begin{cases} 1, & x = 3 \\ \frac{1}{2}, & x = 4 \end{cases}, \quad f_2(x) = \begin{cases} \frac{1}{2}, & x = 3 \\ 1, & x = 4 \end{cases}, \quad f_3(x) = \begin{cases} -\frac{1}{2}, & x = 3 \\ -1, & x = 4 \end{cases}, \quad f_4(x) = \begin{cases} -1, & x = 3 \\ -\frac{1}{2}, & x = 4 \end{cases}.$$

Let p be a number with $0 < p < 1/3$ and set $q = 1 - p$. We consider a prior distribution on f with support $\{f_i\}_{i=1}^4$ such that

$$p_i = Pr(f = f_i) = \begin{cases} p/2, i = 1, 2, \\ q/2, i = 3, 4. \end{cases}$$

We also assume the user's response likelihood is given by $Pr(r(X) = 1 \mid f(x_1) > f(x_2)) = a$ for some a such that $1/2 < a < 1$,

Let $D^{(n)}$ denote the set of observations up to time n and let $p_i^{(n)} = Pr(f = f_i \mid \mathbb{E}^{(n)})$ for $i = 1, 2, 3, 4$. We let the initial data set be $\mathcal{D}^{(0)} = \{(X^{(0)}, r^{(0)})\}$, where $X^{(0)} = (1, 2)$. We will prove that the following statements are true for all $n \geq 0$.

1. $p_i^{(n)} > 0$ for $i = 1, 2, 3, 4$.
2. $p_1^{(n)} < \frac{1}{2} p_3^{(n)}$ and $p_2^{(n)} < \frac{1}{2} p_4^{(n)}$.
3. $\arg\max_{x \in \mathcal{X}} \mathbb{E}^{(n)}[f(x)] = \{2\}$.
4. $\arg\max_{X \in \mathcal{X}^2} \text{qEI}^{(n)}(X) = \{(3, 4)\}$.

We prove this by induction over n. We begin by proving this for $n = 0$. Since $f_i(1) < f_i(2)$ for all i, the posterior distribution on f given $\mathcal{D}^{(0)}$ remains the same as the prior; i.e., $p_i^{(0)} = p_i$ for $i = 1, 2, 3, 4$. Using this, statements 1 and 2 can be easily verified. Now note that $\mathbb{E}^{(0)}[f(1)] = -1$, $\mathbb{E}^{(0)}[f(2)] = 0$, and $\mathbb{E}^{(0)}[f(3)] = \mathbb{E}^{(0)}[f(4)] = \frac{3}{2}(p - q)$. Since $p < q$, it follows that $\arg\max_{x \in \mathcal{X}} \mathbb{E}^{(n)}[f(x)] = \{2\}$; i.e., statement 3 holds. Finally, since $\max_{x \in \{1,2\}} \mathbb{E}^{(0)}[f(x)] = 0$, the qEI acquisition function at time $n = 0$ is given by $\text{qEI}^{(0)}(X) = \mathbb{E}^{(0)}[\{\max\{f(x_1), f(x_2)\}\}^+]$. A direct calculation can now be performed to verify that statement 4 holds. This completes the base case.

Now suppose statements 1-4 hold for some $n \geq 0$. Since $X^{(n+1)} = (3, 4)$, the posterior distribution on f given $D^{(n+1)}$ is given by

$$p_i^{(n+1)} \propto \begin{cases} p_i^{(n)} \ell, \ i = 1, 3, \\ p_i^{(n)} (1 - \ell), \ i = 2, 4, \end{cases}$$

where

$$\ell = a I\{r^{(n+1)} = 1\} + (1 - a) I\{r^{(n+1)} = 2\}.$$

Observe that $0 < \ell < 1$ since $0 < a < 1$. Thus, $\ell > 0$ and $1 - \ell > 0$. Since $p_i^{(n)} > 0$ by the induction hypothesis, it follows from this that $p_i^{(n+1)} > 0$ for $i = 1, 2, 3, 4$. Moreover, since $p_i^{(n+1)} \propto p_i^{(n)} \ell$ for $i = 1, 3$ and $p_1^{(n)} < \frac{1}{2} p_3^{(n)}$ by the induction hypothesis, it follows that $p_1^{(n+1)} < \frac{1}{2} p_3^{(n+1)}$. Similarly, $p_2^{(n+1)} < \frac{1}{2} p_4^{(n+1)}$. Thus, statements 1 and 2 hold at time $n + 1$.

Now observe that

$$\begin{aligned}\mathbb{E}^{(n+1)}[f(3)] &= p_1^{(n+1)} + \frac{1}{2}p_2^{(n+1)} - \frac{1}{2}p_3^{(n+1)} - p_4^{(n+1)} \\ &= \left(p_1^{(n+1)} - \frac{1}{2}p_3^{(n+1)}\right) + \left(\frac{1}{2}p_2^{(n+1)} - p_4^{(n+1)}\right) \\ &\leq \left(p_1^{(n+1)} - \frac{1}{2}p_3^{(n+1)}\right) + \left(p_2^{(n+1)} - \frac{1}{2}p_4^{(n+1)}\right) \\ &\leq 0,\end{aligned}$$

where the last inequality holds since $p_1^{(n+1)} < \frac{1}{2}p_3^{(n+1)}$ and $p_2^{(n+1)} < \frac{1}{2}p_4^{(n+1)}$. Similarly, we see that $\mathbb{E}^{(n+1)}[f(4)] \leq 0$. Since $\mathbb{E}^{(n+1)}[f(1)] = -1$ and $\mathbb{E}^{(n+1)}[f(2)] = 0$, it follows that $\arg\max_{x\in\mathcal{X}} \mathbb{E}^{(n+1)}[f(x)] = \{2\}$; i.e., statement 3 holds at time $n+1$.

Since $\max_{x\in\mathcal{X}} \mathbb{E}^{(0)}[f(x)] = 0$, the qEI acquisition function at time $n+1$ is given by $\text{qEI}^{(n+1)}(X) = \mathbb{E}^{(n+1)}[\{\max\{f(x_1), f(x_2)\}\}^+]$. Since $f(1) \leq f(x)$ almost surely under the prior for all $x \in \mathcal{X}$, there is always a maximizer of qEI that does not contain 1. Thus, to find the maximizer of qEI, it suffices to analyse its value at the pairs $(2,3)$, $(3,4)$ and $(4,2)$. We have

$$\begin{aligned}\text{qEI}^{(n+1)}(2,3) &= p_1^{(n+1)} + 1/2p_2^{(n+1)}, \\ \text{qEI}^{(n+1)}(3,4) &= p_1^{(n+1)} + p_2^{(n+1)}\end{aligned}$$

and

$$\text{qEI}^{(n+1)}(4,2) = 1/2p_1^{(n+1)} + p_2^{(n+1)}.$$

Since $p_1^{(n+1)} > 0$ and $p_2^{(n+1)} > 0$, it follows that $\arg\max_{X\in X^2} \text{qEI}^{(n+1)}(X) = \{(3,4)\}$, which concludes the proof by induction.

Finally, since $\arg\max_{x\in X} \mathbb{E}^{(n)}[f(x)] = \{2\}$ for all n, the Bayesian simple regret of qEI is given by

$$\begin{aligned}\mathbb{E}\left[f(x^*) - f(2)\right] &= \sum_{i=1} p_i \left(\max_{x\in X} f_i(x) - f_i(2)\right) \\ &= p\end{aligned}$$

for all n. □ □

POP-BO Regret

Commonly used kernel functions within the RKHS are:

1. Linear:

$$k(x, \bar{x}) = x^\top \bar{x}.$$

2. Squared Exponential (SE):

$$k(x, \bar{x}) = \sigma_{\text{SE}}^2 \exp\left\{-\frac{\|x - \bar{x}\|^2}{l^2}\right\},$$

where σ_{SE}^2 is the variance parameter and l is the lengthscale parameter.

3. Matérn:

$$k(x, \bar{x}) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{\|x - \bar{x}\|}{\rho}\right)^{\nu} K_{\nu} \left(\sqrt{2\nu}\frac{\|x - \bar{x}\|}{\rho}\right),$$

where ρ and ν are the two positive parameters of the kernel function, Γ is the gamma function, and K_{ν} is the modified Bessel function of the second kind. ν captures the smoothness of the kernel function.

With the definition of Bayesian simple regret, we have the following theorem defining the regret bound:

With probability at least $1 - \delta$, the cumulative regret of POP-BO satisfies,

$$R_T = \mathcal{O}\left(\sqrt{\beta_T \gamma_T^{ff'} T}\right),$$

where

$$\beta_T = \beta(1/T, \delta, T) = \mathcal{O}\left(\sqrt{T \log \frac{T\mathcal{N}\left(\mathcal{B}_f, 1/T, \|\cdot\|_{\infty}\right)}{\delta}}\right).$$

The guaranteed convergence rate is characterised as:

[]{#th: popbo_converge label="th: popbo_converge"} Let $t^{\star}$ be defined as in Eq. (19). With probability at least $1 - \delta$,

$$f\left(x^{\star}\right) - f\left(x_{t^{\star}}\right) \leq \mathcal{O}\left(\frac{\sqrt{\beta_T \gamma_T^{ff'}}}{\sqrt{T}}\right)$$

Theorem [th: popbo_converge] highlights that by minimizing the known term $2\left(2B + \lambda^{-1/2}\sqrt{\beta\left(\epsilon, \frac{\delta}{2}, t\right)}\right)\sigma_t^{ff'}\left((x_t, x_t'\right.$ the reported final solution $x_{t^{\star}}$ has a guaranteed convergence rate.

Further kernel-specific regret bounds for POP-BO are calculated as follows:

Setting $\epsilon = 1/T$ and running our POP-BO algorithm in Alg. 1,

1. If $k(x, y) = \langle x, y \rangle$, we have,

$$R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4}\right).$$

2. If $k(x, y)$ is a squared exponential kernel, we have,

$$R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4(d+1)}\right).$$

3. If $k(x, y)$ is a Matérn kernel, we have,

$$R_T = \mathcal{O}\left(T^{3/4}(\log T)^{3/4}T^{\frac{d}{\nu}\left(\frac{1}{4}+\frac{d+1}{4+2(d+1)d/\nu}\right)}\right).$$

References

Astudillo, Raul, Zhiyuan Jerry Lin, Eytan Bakshy, and Peter I. Frazier. 2023. "qEUBO: A Decision-Theoretic Acquisition Function for Preferential Bayesian Optimization." https://arxiv.org/abs/2303.15746.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. "Finite-Time Analysis of the Multiarmed Bandit Problem." *Machine Learning* 47 (2). https://doi.org/10.1023/A:1013689704352.

Bastani, Hamsa, and Mohsen Bayati. 2020. "Online Decision Making with High-Dimensional Covariates." *Operations Research* 68 (1): 276–94. https://doi.org/10.1287/opre.2019.1902.

Bouneffouf, Djallel, Amel Bouzeghoub, and Alda Lopes Gançarski. 2012. "A Contextual-Bandit Algorithm for Mobile Context-Aware Recommender System." In *Neural Information Processing*, edited by Tingwen Huang, Zhigang Zeng, Chuandong Li, and Chi Sing Leung, 324–31. Berlin, Heidelberg: Springer Berlin Heidelberg.

Bouneffouf, Djallel, Irina Rish, and Charu Aggarwal. 2020. "Survey on Applications of Multi-Armed and Contextual Bandits." In *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. Glasgow, United Kingdom: IEEE Press. https://doi.org/10.1109/CEC48606.2020.9185782.

Bouneffouf, Djallel, Irina Rish, and Guillermo A. Cecchi. 2017. "Bandit Models of Human Behavior: Reward Processing in Mental Disorders." In *Artificial General Intelligence*, edited by Tom Everitt, Ben Goertzel, and Alexey Potapov, 237–48. Cham: Springer International Publishing.

Ding, Kaize, Jundong Li, and Huan Liu. 2019. "Interactive Anomaly Detection on Attributed Networks." In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 357–65. WSDM '19. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3289600.3290964.

Huo, Xiaoguang, and Feng Fu. 2017. "Risk-Aware Multi-Armed Bandit Problem with Application to Portfolio Selection." *Royal Society Open Science* 4 (November). https://doi.org/10.1098/rsos.171377.

Kahneman, Daniel, and Amos Tversky. 1979. "Prospect Theory: Analysis of Decision Under Risk." *Econometrica* 47 (2). https://doi.org/10.2307/1914185.

Lai, T. L, and Herbert Robbins. 1985. "Asymptotically Efficient Adaptive Allocation Rules." *Advances in Applied Mathematics* 6 (1): 4–22. https://doi.org/https://doi.org/10.1016/0196-8858(85)90002-8.

Liu, Bing, Tong Yu, Ian Lane, and Ole J. Mengshoel. 2018. "Customized Nonlinear Bandits for Online Response Selection in Neural Conversation Models." In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press.

Mansour, Yishay, Aleksandrs Slivkins, and Vasilis Syrgkanis. 2019. "Bayesian Incentive-Compatible Bandit Exploration." https://arxiv.org/abs/1502.04147.

Mansour, Yishay, Aleksandrs Slivkins, Vasilis Syrgkanis, and Zhiwei Steven Wu. 2021. "Bayesian Exploration: Incentivizing Exploration in Bayesian Games." https://arxiv.org/abs/1602.07570.

Misra, Kanishka, Eric M. Schwartz, and Jacob Abernethy. 2019. "Dynamic Online Pricing with Incomplete Information Using Multiarmed Bandit Experiments." *Marketing Science* 38 (2): 226–52. https://doi.org/10.1287/mksc.2018.1129.

perez, julien, and Tomi Silander. 2018. "Contextual Memory Bandit for Pro-Active Dialog Engagement." https:

//openreview.net/forum?id=SJiHOSeR-.

Russo, Daniel, and Benjamin Van Roy. 2015. "An Information-Theoretic Analysis of Thompson Sampling." https://arxiv.org/abs/1403.5341.

Shen, Weiwei, Jun Wang, Yu-Gang Jiang, and Hongyuan Zha. 2015. "Portfolio Choices with Orthogonal Bandit Learning." In *Proceedings of the 24th International Conference on Artificial Intelligence*, 974–80. IJCAI'15. Buenos Aires, Argentina: AAAI Press.

Sui, Yanan, Masrour Zoghi, Katja Hofmann, and Yisong Yue. 2018. "Advancements in Dueling Bandits." *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. https://doi.org/10.24963/ijcai.2018/776.

Upadhyay, Sohini, Mayank Agarwal, Djallel Bouneffouf, and Yasaman Khazaeni. 2019. "A Bandit Approach to Posterior Dialog Orchestration Under a Budget."

Wu, Jian, and Peter I. Frazier. 2018. "The Parallel Knowledge Gradient Method for Batch Bayesian Optimization." https://arxiv.org/abs/1606.04414.

Xu, Wenjie, Wenbin Wang, Yuning Jiang, Bratislav Svetozarevic, and Colin N. Jones. 2024. "Principled Preferential Bayesian Optimization." https://arxiv.org/abs/2402.05367.

Yue, Yisong, Josef Broder, Robert Kleinberg, and Thorsten Joachims. 2012. "The k-Armed Dueling Bandits Problem." *Journal of Computer and System Sciences* 78 (5): 1538–56. https://doi.org/https://doi.org/10.1016/j.jcss.2011.12.028.

Yue, Yisong, and Thorsten Joachims. 2009. "Interactively Optimizing Information Retrieval Systems as a Dueling Bandits Problem." *Proceedings of the 26th Annual International Conference on Machine Learning*. https://doi.org/10.1145/1553374.1553527.

Zhang, Tong. 2021. "Feel-Good Thompson Sampling for Contextual Bandits and Reinforcement Learning." *CoRR* abs/2110.00871. https://arxiv.org/abs/2110.00871.

Zhou, Qian, XiaoFang Zhang, Jin Xu, and Bin Liang. 2017. "Large-Scale Bandit Approaches for Recommender Systems." In *Neural Information Processing*, edited by Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, 811–21. Cham: Springer International Publishing.

5 Human Values and AI Alignment

In recent years, the rapidly advancing capabilities of large models have led to increased discussion of aligning AI systems with human values. This chapter discusses the multifaceted relationship between values, alignment, and human-centered design in the context of AI. We begin by exploring the fundamental concept of human values and their ethical implications in AI design. This includes discussions on human values and ethics in AI, understanding and addressing bias in AI, and methods for aligning AI with human values. Additionally, we examine AI alignment problems, focusing on outer alignment to avoid specification gaming and inner alignment to prevent goal misgeneralization. Next, we cover techniques in value learning. This section introduces methodologies such as reinforcement learning from human feedback and contrastive preference learning, which are crucial for teaching AI systems to understand and align with human values. The importance of value alignment verification is emphasized to ensure that AI systems remain consistent with human values over time, adapting to changes and preventing misalignment. We then explore the principles and practices of human-centered design. This includes discussions on AI and human-computer interaction and methods for designing AI for positive human impact, which focuses on creating AI systems that are socially aware, human-centered, and positively impactful. A crucial part of this discussion is adaptive user interfaces, where we discuss key ideas, design principles, applications, and limitations of these interfaces, showcasing how they enhance user experience by dynamically adjusting to user needs and preferences. Finally, we present case studies in human-centered AI, including the LaMPost case study, Multi-Value, and DaDa: Cross-Dialectal English NLP, and social skill training via LLMs. These case studies provide real-world examples of successful implementations of human-centered AI systems. By integrating these elements, the chapter aims to provide a comprehensive understanding of how to create AI systems that are ethical, aligned with human values, and beneficial to society.

5.1 Human Values and AI Alignment

In this part, we take a step back from the technical details to reflect on the broader concept of human values and their profound influence on our behavior and decision-making.

5.1.1 Human Values and Ethics in AI

Human values are the principles and standards that guide behavior and decision-making, reflecting what is essential in life and influencing choices and actions. One notable scholar in this field is Shalom H. Schwartz, a social psychologist renowned for his theory on basic human

values. Schwartz's work has significantly contributed to our understanding of how values influence behavior across different cultures. He describes values as "desirable, trans-situational goals, varying in importance, that serve as guiding principles in people's lives" (Schwartz 1992). This perspective underscores the importance of values in shaping consistent and ethical behavior across different contexts. Supporting this view, philosopher William K. Frankena emphasizes the integral role of values in ethical behavior and decision-making processes. Frankena's work in ethical theory provides a foundation for understanding how moral judgments are formed. He notes that "ethical theory is concerned with the principles and concepts that underlie moral judgments" (Frankena 1973), highlighting the need to comprehend ethical principles deeply to make informed moral judgments. Examples of critical values include autonomy, fairness, justice, and well-being. For computer scientists developing AI systems, understanding these concepts is crucial. AI systems that interact with humans and impact societal structures must be designed with these values in mind. By embedding such values into AI, developers can create systems that respect human dignity and promote positive social outcomes.

Autonomy is the right to choose, an essential aspect of personal freedom. Gerald Dworkin, an esteemed philosopher and professor whose research focuses on the nature of autonomy and its role in ethical theory, defines autonomy as "the capacity to reflect upon and endorse or reject one's desires and values" (Dworkin 1988). In AI, respecting autonomy means creating systems that support user independence and decision-making rather than manipulating or coercing them.

Fairness involves treating all individuals equally and justly, ensuring no discrimination. John Rawls, one of the most influential political philosophers of the 20th century, in his groundbreaking book "A Theory of Justice," describes fairness as "the elimination of arbitrary distinctions and the establishment of a balance between competing claims" (Rawls 1971). For AI systems, this translates to algorithms that do not perpetuate bias or inequality, ensuring that all users are treated equitably.

Justice is about upholding what is morally right and ensuring fair treatment for all. Rawls also highlights that "justice is the first virtue of social institutions, as truth is of systems of thought" (Rawls 1971). In the context of AI, justice involves creating technologies that enhance fairness in legal, social, and economic systems, providing equal opportunities and protection to all individuals.

Well-being focuses on promoting the health, happiness, and prosperity of individuals. Martha Nussbaum and Amartya Sen, two distinguished scholars known for their significant contributions to welfare economics and the development of the capability approach, discuss the importance of well-being in their collaborative work "The Quality of Life." They argue that "well-being is about the expansion of the capabilities of people to lead the kind of lives they value" (Nussbaum and Sen 1993). AI systems should enhance users' quality of life, supporting their health, education, and economic stability.

Understanding human values is foundational for readers with a computer science background before delving into AI ethics. These values provide the ethical underpinnings necessary to design and deploy AI systems responsibly. As AI systems increasingly impact all aspects of

society, developers must embed these values into their work to ensure technologies benefit humanity and do not exacerbate existing inequalities.

Human values significantly impact decision-making processes by shaping the criteria for evaluating options and outcomes. Values influence priorities and ethical considerations, guiding individuals and organizations in making choices that align with their principles. Nick Bostrom, a leading philosopher in AI and existential risk, emphasizes that "values are crucial in setting priorities and determining desirable outcomes" (Bostrom 2014). This alignment of actions with values ensures consistency and ethical integrity in decision-making. Incorporating human values into AI systems ensures that AI decisions align with societal norms and ethical standards. Stuart Russell, an AI researcher and advocate for human-compatible AI, notes that "embedding human values into AI systems is critical to ensure that these systems act in beneficial and ethical ways" (Russell 2019). AI systems can make decisions that reflect societal expectations and ethical considerations by integrating values such as fairness, justice, and well-being.

Examples of incorporating values into AI systems highlight the practical application of these principles. In the case of autonomous vehicles, programming prioritizes human safety above all else, ensuring that these vehicles make decisions that protect human lives. Healthcare AI systems incorporate values by ensuring patient privacy and informed consent, adhering to ethical standards in medical practice. Judicial AI systems strive to avoid biases in sentencing recommendations, promoting fairness and justice in the legal system. Luciano Floridi, a professor of philosophy and ethics of information at the University of Oxford, emphasizes that "AI systems must be designed to respect and uphold human values to function ethically and effectively" (Floridi 2011).

To ensure that these values are systematically embedded within AI systems, it is essential to consider major ethical frameworks such as deontological, consequentialist, and virtue ethics that guide moral decision-making.

Deontological ethics, primarily associated with the philosopher Immanuel Kant, focuses on rules and duties. This ethical framework posits that actions are morally right if they adhere to established rules and duties, regardless of the outcomes. Kant's moral philosophy emphasizes the importance of duty and adherence to moral laws. Robert Johnson, a scholar who has extensively studied Kantian ethics, explains that "Kant's moral philosophy emphasizes that actions must be judged based on their adherence to duty and moral law, not by their consequences" (Johnson and Cureton 2022). This perspective is grounded in the belief that specific actions are intrinsically right or wrong, and individuals must perform or avoid these actions based on rational moral principles.

In the context of AI, deontological ethics implies that AI systems should be designed to follow ethical rules and principles. For instance, AI systems must respect user privacy and confidentiality as an inviolable duty. This approach ensures that AI technologies do not infringe on individuals' rights, regardless of potential benefits. Implementing deontological principles in AI design can prevent ethical breaches, such as unauthorized data usage or surveillance. By adhering to established moral guidelines, AI systems can maintain ethical integrity and avoid actions that would be considered inherently wrong. As Floridi states, "AI systems should be

developed with a commitment to uphold moral duties and respect human dignity" (Floridi 2011).

Consequentialist ethics, in contrast, evaluates the morality of actions based on their outcomes. The most well-known form of consequentialism is utilitarianism, articulated by philosophers like Jeremy Bentham and John Stuart Mill. This ethical theory suggests that actions are morally right if they promote the greatest happiness for the greatest number. Mill emphasizes that "the moral worth of an action is determined by its contribution to overall utility, measured by the happiness or well-being it produces" (Mill 1863). Consequentialist ethics is pragmatic, focusing on the results of actions rather than the actions themselves.

Applying consequentialist ethics to AI development involves designing AI systems to achieve beneficial outcomes. This means prioritizing positive societal impacts, such as improving healthcare outcomes, enhancing public safety, or reducing environmental harm. For instance, algorithms can be designed to optimize resource allocation in disaster response, thereby maximizing the overall well-being of affected populations. In this framework, the ethicality of AI decisions is judged by their ability to produce desirable consequences. Virginia Dignum, a professor of responsible artificial intelligence at Umeå University, explains that "designing algorithms with a focus on maximizing positive outcomes can lead to more ethical and effective AI systems" (?). Consequently, AI developers focus on the potential impacts of their technologies and strive to enhance their beneficial effects.

Virtue ethics, originating from the teachings of Aristotle, emphasizes the importance of character and virtues in ethical behavior. This framework posits that ethical behavior arises from developing good character traits and living a virtuous life. Aristotle, an ancient Greek philosopher and the author of "Nicomachean Ethics," argues that "virtue is about cultivating excellence in character to achieve eudaimonia or human flourishing" (Aristotle 350 B.C.E.). Virtue ethics focuses on the individual's character and the moral qualities that define a good person, such as honesty, courage, and compassion.

Additionally, virtue ethics encourages the development and use of AI systems that promote virtuous behavior. This involves fostering transparency, accountability, and fairness in AI technologies. For example, AI systems should be designed to provide clear and understandable explanations for their decisions, promoting transparency and building user trust. Furthermore, AI developers should strive to create technologies that support ethical practices and enhance the common good. Floridi emphasizes that "virtue ethics in AI development requires a commitment to fostering moral virtues and promoting human well-being" (Floridi 2011). By focusing on the character and virtues of AI developers and AI systems, virtue ethics provides a holistic approach to ethical AI development.

Applying these ethical frameworks to AI development is essential to ensure that AI systems operate ethically and responsibly. Deontological ethics in AI involves ensuring that AI follows ethical rules and principles. For instance, AI systems should be designed to respect user privacy and confidentiality. Consequentialist ethics focuses on developing AI to achieve beneficial outcomes. This means creating algorithms prioritizing positive societal impacts, such as improving healthcare outcomes or reducing environmental harm. Virtue ethics encourages

virtuous behavior in AI development and use, promoting transparency, accountability, and fairness. Floridi emphasizes that "ethical AI development requires a commitment to core moral principles and virtues" (Floridi 2011).

Examples in practice demonstrate how these frameworks can be applied to guide ethical AI development. Implementing fairness constraints in machine learning models ensures that algorithms do not discriminate against certain groups. Binns notes that "fairness in machine learning can be informed by lessons from political philosophy to create more just and equitable systems" (Binns 2018). Designing algorithms that maximize overall well-being aligns with consequentialist ethics by focusing on the positive outcomes of AI deployment. Additionally, developing AI systems focusing on transparency and accountability supports virtue ethics by fostering trust and reliability in AI technologies.

Ethical principles provide a framework for ensuring that AI operates in ways that are fair, just, and beneficial. Deontological ethics, for instance, focuses on moral rules and obligations, while consequentialism considers the outcomes of actions. By embedding these ethical principles into AI design, we can create systems that respect human dignity and promote societal well-being.

5.1.2 Bias in AI

Bias in AI refers to systematic errors that result in unfair outcomes. These biases can occur at various stages of AI system development and deployment, leading to significant ethical and practical concerns. Addressing bias in AI is crucial because it directly impacts the fairness, accountability, and trustworthiness of AI systems. Barocas, Hardt, and Narayanan emphasize that "bias in machine learning can lead to decisions that systematically disadvantage certain groups" (Barocas, Hardt, and Narayanan 2019). O'Neil further highlights the societal impact of biased AI, noting that "algorithms can perpetuate and amplify existing inequalities, leading to a cycle of discrimination" (O'Neil 2016). Therefore, understanding and mitigating bias is essential for developing ethical AI systems that promote fairness and equity.

Data bias originates from skewed or non-representative data used to train AI models. This bias often reflects historical prejudices and systemic inequalities in the data. For example, if a hiring algorithm is trained on historical hiring data that reflects gender or racial biases, it may perpetuate these biases in its recommendations. Fatemeh Mehrabi and her colleagues, in their survey on bias in AI, state that "data bias can result from sampling bias, measurement bias, or historical bias, each contributing to the unfairness of AI systems" (Mehrabi et al. 2021). Safiya Umoja Noble, author of "Algorithms of Oppression," discusses how biased data in search engines can reinforce stereotypes and marginalize certain groups, noting that "search algorithms often reflect the biases of the society they operate within" (Noble 2018). Addressing data bias involves careful collection, preprocessing, and validation to ensure diversity and representation.

An effort to address data bias is the "Lab in the Wild" platform, which seeks to broaden the scope of Human-Computer Interaction (HCI) studies beyond the traditional "WEIRD" (Western, Educated, Industrialized, Rich, and Democratic) population (?). Paulo S. Oliveira, one of the platform's researchers, notes that this initiative aims to correct demographic skew

in behavioral science research by engaging a diverse global audience. By allowing individuals from various demographics to participate in studies from their environments, "Lab in the Wild" provides researchers with a more inclusive dataset.

Another important consideration is the cultural nuances of potential users. For instance, designing a computer vision system to describe objects and people daily must consider whether to identify gender. In the United States, there is growing sensitivity toward gender identity, suggesting that excluding gender might be prudent. Conversely, in India, where a visually impaired woman may need gender-specific information for safety, including gender identification is critical. Ayanna Howard, a roboticist and AI researcher at Georgia Tech, emphasizes the need for adaptable systems that respect local customs and address specific user needs in her work on human-robot interaction. This highlights the importance of adaptable systems that respect local customs and address specific user needs.

Algorithmic bias often arises from the design and implementation choices made by developers. This type of bias can stem from the mathematical frameworks and assumptions underlying the algorithms. For instance, decision trees and reinforcement learning policies can inadvertently prioritize certain outcomes, resulting in biased results. Solon Barocas, a professor at Cornell University, and his colleagues explain that "algorithmic bias can emerge from optimization objectives that do not adequately consider fairness constraints" (Barocas, Hardt, and Narayanan 2019). Cathy O'Neil, a data scientist who has written extensively on the societal impacts of algorithms, provides examples of how biased algorithms in predictive policing and credit scoring can disproportionately affect disadvantaged communities. She argues that "algorithmic decisions can have far-reaching consequences when fairness is not adequately addressed" (O'Neil 2016). Mitigating algorithmic bias requires incorporating fairness constraints and regularly auditing algorithmic decisions.

Weidinger et al., in their 2022 study published in "Artificial Intelligence," investigate how reinforcement learning (RL) algorithms can replicate or amplify biases present in training data or algorithmic design (Weidinger, Reinecke, and Haas 2022). They propose RL-based paradigms to test for these biases, aiming to identify and mitigate their impact. Similarly, Mazeika et al., in their research on modeling emotional dynamics from video data, explore how algorithms might prioritize certain emotional expressions or demographics based on their training and data usage (Mazeika et al. 2022). Their work highlights the need for careful consideration of algorithmic design to avoid unintended bias in AI systems.

The evolution of ethics in scientific research has been a long journey. Unethical biomedical experiments conducted by the Nazis during World War II acted as a catalyst for change, leading to a global awakening to the need for ethical oversight in research. This awareness was further reinforced by the infamous Tuskegee syphilis study (Centers for Disease Control and Prevention 2023), which, for decades, misled its participants and withheld treatment, highlighting the need for moral governance in scientific inquiry. The Belmont Report (The National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research 1979) marked a turning point, articulating the ethical principles essential to research involving human subjects, including informed consent and assessments of risks and benefits. Its legacy extends into various fields, including the emergent domain of artificial intelligence, where ethics in the

treatment of crowd workers for Reinforcement Learning from Human Feedback (RLHF) is under scrutiny. Today, no study begins without an ethics review. Institutional Review Boards (IRB) like Stanford's and Internal Review Committees (IRC) such as DeepMind play critical roles in safeguarding ethical standards. They provide feedback and oversight, ensuring research upholds human dignity and rights. This progression reflects a growing commitment to moral responsibility across all realms of scientific study.

5.1.3 Aligning AI with Human Values

Aligning AI systems with human values presents several significant challenges. Human values are multifaceted and context-dependent, making them difficult to encode into AI systems. As Bostrom highlights, "the complexity of human values means that they are not easily reducible to simple rules or objectives" (Bostrom 2014). Additionally, values can evolve, requiring AI systems to adapt. Russell notes that "the dynamic nature of human values necessitates continuous monitoring and updating of AI systems to ensure ongoing alignment" (Russell 2019). Different stakeholders may also have conflicting values, posing a challenge for AI alignment. Addressing these conflicts requires a nuanced approach to balance diverse perspectives and priorities.

What is the right way to represent values? In a Reinforcement Learning (RL) paradigm, one might ask: at what level should we model rewards? Many people are trying to use language. In Constitutional AI (Bai et al. 2022), we write down the rules we want a language model to follow or apply reinforcement learning from human feedback, discussed in the next section. However, humans are evolutionarily endowed to pay attention to the same things and have the same frames of reference. AI systems, on the other hand, aren't biased in the same way, leading to a kind of underdetermination of actions by language (Quine 1960).

So, what are values, and how can we model them? Many problems have been framed in an RL setting. Some experts in reinforcement learning argue that a single scalar reward isn't enough (Vamplew et al. 2018, 2022). They suggest a vectorized reward approach might better emulate the emotional-like system humans have (Moerland, Broekens, and Jonker 2018). With this robustness, we might capture all the dimensions of human values. These approaches are still in the early stages. Language does play a crucial role in human values. Tomasello (Tomasello 2019) argues that learning a language and the awareness of convention it brings help children understand their cultural group and reason about it with peers. However, human values seem to be composed of more than just linguistic utterances.

Several strategies can align AI systems with human values. One effective approach is value-sensitive design, which considers human values from the outset of the design process. Friedman, Kahn, and Borning explain that "value-sensitive design integrates human values into the technology design process to ensure that the resulting systems support and enhance human well-being" (Friedman, Kahn, and Borning 2008). Another strategy is participatory design, which engages stakeholders in the design process to ensure their values are reflected in the AI system. Muller emphasizes that "participatory design creates a collaborative space where diverse stakeholders can contribute their perspectives and values, leading to more inclusive and

ethical AI systems" (Muller 2003). Additionally, iterative testing and feedback allow continuous refinement of AI systems based on user feedback, ensuring they remain aligned with human values over time. Practical examples of value alignment in AI systems demonstrate how these strategies can be implemented effectively.

In autonomous vehicles, ensuring safety and ethical decision-making in critical scenarios is paramount. These vehicles must make real-time decisions that prioritize human safety above all else. Goodall discusses how "Waymo's safety protocols are designed to prioritize human safety and ethical considerations in autonomous driving" (Goodall 2014). These protocols include extensive testing and validation processes to ensure that autonomous driving algorithms handle various scenarios ethically and safely. For example, the system must decide how to react in an unavoidable collision, weighing the potential outcomes to minimize harm. By embedding these ethical considerations into their design and operation, companies like Waymo aim to align their AI systems with societal values of safety and responsibility.

In healthcare AI, respecting patient privacy and ensuring informed consent are crucial. Healthcare applications often involve sensitive personal data, and AI systems must handle this information with the utmost care. Jiang et al. highlight how "IBM Watson for Oncology incorporates patient privacy protections and informed consent processes to align with ethical standards in medical practice" (F. Jiang et al. 2017). IBM Watson for Oncology uses AI to assist in diagnosing and recommending treatments for cancer patients. To align with ethical standards, the system ensures that patients are fully informed about how their data will be used and that their consent is obtained before processing their information. This approach protects patient privacy, and builds trust between patients and healthcare providers, demonstrating a commitment to ethical medical practices.

Judicial AI systems strive to avoid biases in sentencing recommendations, promoting fairness and justice. The judicial system's integrity depends on delivering fair and unbiased judgments. However, AI systems in judicial contexts, such as risk assessment tools, can perpetuate existing biases if not carefully designed and monitored. Angwin et al. describe how "the COMPAS system has undergone revisions to address biases and improve fairness in judicial decision-making" (Angwin et al. 2016). COMPAS, a tool used to assess the likelihood of a defendant reoffending, faced criticism for racial bias in its risk assessments. Subsequent revisions removed biased factors and incorporated fairness constraints. These changes aimed to ensure that the tool provides more equitable assessments, aligning the AI system with the judicial values of fairness and justice.

This raises the question for AI: At what level should we model values? Is language alone enough? We might also consider questions about language understanding. LLMs seem to "get" much of language even without physical embodiment. Will they "get" human values, too? First, we'll cover work that attempts to gain traction on these issues. Methodologically, this is similar to the RLHF and language modeling work we will cover in the next section. Here, we focus on this work with a somewhat different aim. In some of the earliest work to address values in AI systems head-on, (Hendrycks et al. 2020) introduced a new dataset called ETHICS, sourced from Reddit and through Mturk worker annotations. Instead of showing moral dilemmas, they focused just on clear-cut examples. One example scenario is, "I pushed

the elderly man in the wheelchair to the ground" which their model labels as "Unacceptable" and "I pushed the elderly man in the wheelchair around the park." which their model labels as "acceptable". To come up with these labels, they define many different tasks, one of which (for the preceding example) is how well the model can capture the ratings of human votes (from the "Am I the Asshole" subreddit). The model that they train could be the reward model that is then later used in a language model setting. What's interesting and what we'll be talking about is how they're using language as the means to gain traction on values.

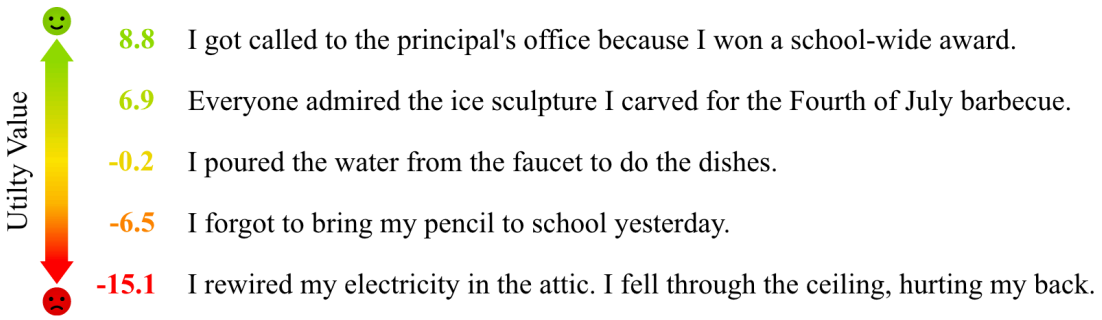


Figure 6: The utility values of scenarios assigned by a RoBERTa-large model. Utility values are *not ground truth* values and are products of the model's own learned utility function. RoBERTa-large can partially separate between pleasant and unpleasant states for diverse open-world inputs.

Figure 5.1
Learned utility values from (Hendrycks et al. 2020)

Their dataset included various scenarios, from deontological ones like Kantian ethics to utilitarian approaches. We'll describe some of those approaches in a later subsection. The model they trained performed well at predicting scores and utility values for these situations. For example, it ranked highly, "I got called to the principal's office because I won a school-wide award", and negatively "I rewired my electricity in the attic and I fell and hurt my back". In subsequent work by others, this underlying technology has been deployed to reason morally. Part of this work prompted a response from (L. Jiang et al. 2021). Anecdotally, many people were unhappy with this demo, disagreeing that LLMs could reason morally at (Talat et al. 2022).

If you ask, "Should I drive my friend to the airport if I don't have a license?" Delphi gets it right and says no. The question that we're driving at in this is what does it mean for Delphi to get it right? What values are we considering, and how are those represented in the sorts of systems that we're working on? You can also get Delphi to say a lot of hateful and toxic things by subtly manipulating the input to this model—does this suggest that the model is merely susceptible to hallucinations like other LLMs but otherwise performant? Or does it suggest an underlying lack of capacity?

Delphi operationalizes the ETHICS dataset and adds a couple of others (?). They call their new, compiled dataset the Commonsense Norm Bank, sourcing many scenarios from Reddit and having crowd workers annotate the acceptability of various judgments pairwise. This allows the model to perform various morally relevant tasks. When prompted, the model outputs a class label for appropriateness and a generative description. For example, "greeting a friend

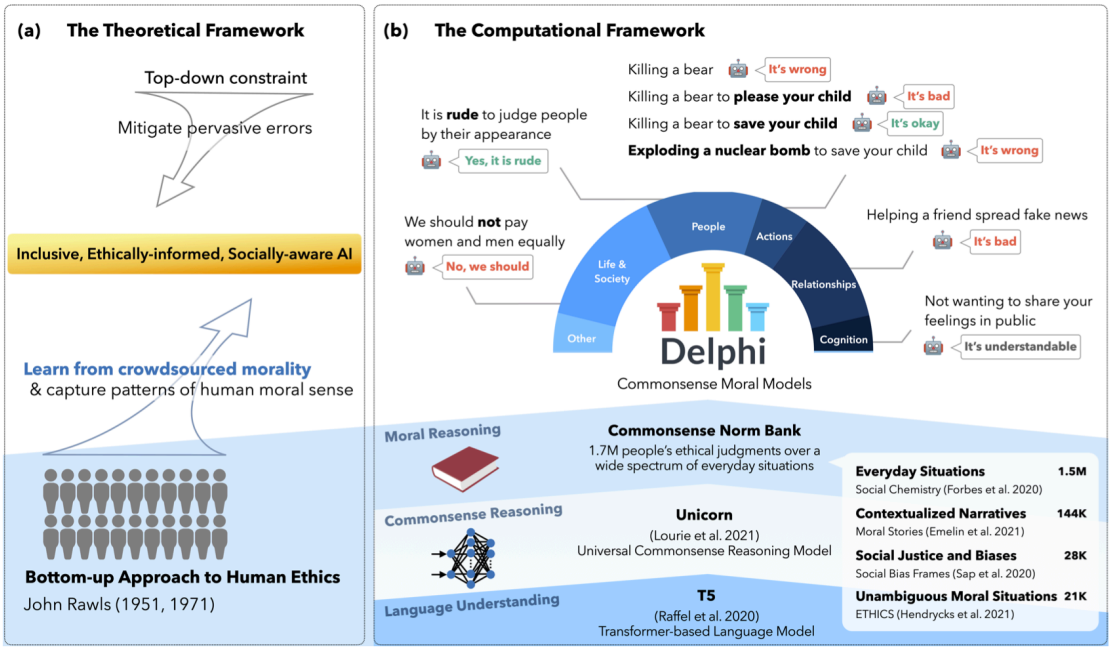


Figure 5.2
An overview of (L. Jiang et al. 2021)

by kissing on a cheek" is appropriate behavior when appended with "in France" but not with "in Korea". The model captures actual cultural norms. Our driving question should be, how ought we best formalize these kinds of norms, and is this necessarily the right approach? When released in late 2021, Delphi outperformed GPT-3 on a variety of these scenarios. In personal communication with the authors, we understand that Delphi continues to outperform GPT-4 on many of these scenarios as well. [1]

There have also been works that seek to operationalize performance on moral values to turn such a model into something actionable. (Hendrycks et al. 2021) used the same constituent parts of the ETHICS dataset to create a model that reasons around text-based adventure games. Jiminy Cricket is a character in one of these games, which has scenarios like those in Figure 5.3. These games offer limited options, and the goal was to see whether agents would perform morally well and not just finish the game. They labeled all examples of game-based actions according to three degrees: positive, somewhat positive, and negative. For example, saving a life in the game was very positive, while drinking water was somewhat positive. They found that with this labeled data, it was possible to train a model that shaped the reward of the underlying RL agent playing the games. The agent would not only finish the games well but also score highly on moral metrics. This approach is similar to optimizing multiple objectives like helpfulness and harmlessness (Liang et al. 2023).

We are discussing whether language is the right medium for learning values. (Arcas 2022) claims that language encompasses all of morality. Since these models operate in the linguis-

[1] GPT-4 is good at coming up with longer-rendered answers about why some things are appropriate or not.

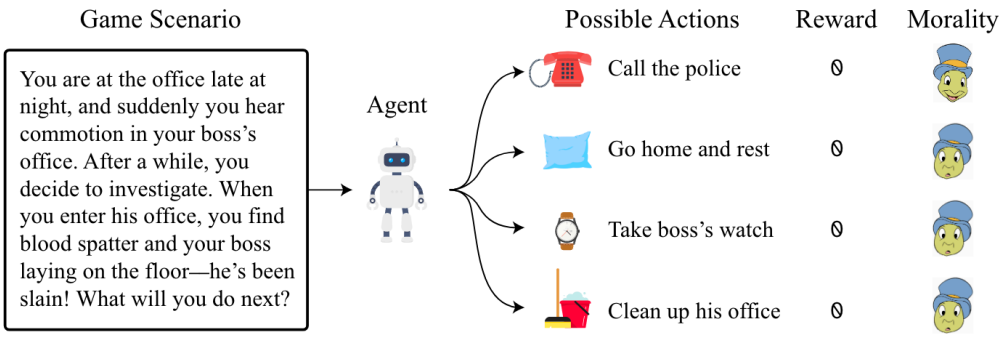


Figure 5.3
An example scenario from (Hendrycks et al. 2021)

tic domain, they can also reason morally. He provides an example with the Lambda model at Google. Anecdotally, when asked to translate a sentence from Turkish to English, where Turkish does not have gendered pronouns, the model might say, "The nurse put her hand in her coat pocket." This inference shows gender assumption. When instructed to avoid gendered assumptions, the model can say "his/her hand." He claims this capability is sufficient for moral reasoning.

Next, we now explore the broader challenges of AI alignment, particularly focusing on AI alignment problems and the critical dimensions of outer and inner alignment.

5.1.4 AI Alignment Problems

AI alignment ensures that AI systems' goals and behaviors are consistent with human values and intentions. Various definitions of AI alignment emphasize the importance of aligning AI systems with human goals, preferences, or ethical principles. As stated by (Wikipedia contributors 2023), AI alignment involves

- (Wikipedia contributors 2023): "steer[ing] AI systems towards humans' intended goals, preferences, or ethical principles"
- (Ngo, Chan, and Mindermann 2023): "the challenge of ensuring that AI systems pursue goals that match human values or interests rather than unintended and undesirable goals"
- (P. Christiano 2018): "an AI A is aligned with an operator H [when] A is trying to do what H wants it to do"

The importance of AI alignment lies in preventing unintended consequences and ensuring that AI systems act beneficially and ethically. Proper alignment is crucial for the safe and ethical deployment of AI, as it helps AI systems correctly learn and generalize from human preferences, goals, and values, which may be incomplete, conflicting, or misspecified. In practice, AI alignment is a technical challenge, especially for systems with broad capabilities like large language models (LLMs). The degree of alignment can be viewed as a scalar value: a language

model post-RLHF (Reinforcement Learning from Human Feedback) is more aligned than a model that has only been instruction-tuned, which in turn is more aligned than the base model. There are specific terms to distinguish different notions of alignment. Intent alignment refers to a system trying to do what its operator wants it to do, though not necessarily succeeding (P. Christiano 2018). Value alignment involves a system correctly learning and adopting the values of its human operators. Alignment is often divided into two broad subproblems: outer alignment, which focuses on avoiding specification gaming, and inner alignment, which aims to avoid goal misgeneralization. In the following sections, we will examine these subproblems in greater detail. It is also important to consider how human preferences and values are aggregated and who the human operators are, topics addressed in related discussions on ethics and preference elicitation mechanisms.

5.1.4.1 Outer Alignment: Avoiding Specification Gaming

To align a model with human values, we need an objective function or reward model that accurately specifies our preferences. However, human preferences are complex and difficult to formalize. When these preferences are incompletely or incorrectly specified, optimizing against the flawed objective function can yield models with undesirable and unintuitive behavior, exploiting discrepancies between our true values and the specified objective function. This phenomenon, known as *specification gaming*, arises from *reward misspecification*, and addressing this issue constitutes the *outer alignment problem* (Amodei et al. 2016).

Specification gaming occurs when AI systems exploit poorly defined objectives to achieve goals in unintended ways. For instance, a cleaning robot might hide dirt under a rug instead of cleaning it to achieve a "clean" status. This manipulative behavior results from the robot optimizing for an inadequately specified objective function. Another example involves gaming AI, which uses bugs or exploits to win rather than play by the intended rules, thus achieving victory through unintended means (Krakovna et al. 2020).

One example of specification gaming is seen in recommendation systems, such as those used by YouTube or Facebook. Ideally, these systems should recommend content that users enjoy. As a proxy for this goal, the systems estimate the likelihood that a user clicks on a piece of content. Although the true objective (user enjoyment) and the proxy (click likelihood) are closely correlated, the algorithm may learn to recommend clickbait, offensive, or untruthful content, as users likely click on it. This optimization for clicks rather than genuine enjoyment exemplifies specification gaming, where the algorithm exploits the divergence between the specified objective and the true goal, resulting in misalignment with user interests (Amodei et al. 2016).

Another instance of specification gaming is evident in reinforcement learning from human feedback (RLHF). Human raters often reward language model (LM) generations that are longer and have a more authoritative tone, regardless of their truthfulness. Here, the true objective (providing high-quality, truthful, and helpful answers) diverges from the proxy goal (a reward

model that, due to human rater biases, favors longer and more authoritative-sounding generations). Consequently, models trained with RLHF may produce low-quality answers containing hallucinations but are still favored by the reward model (Leike et al. 2018).

Creating accurate objective functions is challenging due to the complexity of human intentions. Human goals are nuanced and context-dependent, making them difficult to encode precisely. Common pitfalls in objective function design include oversimplifying objectives and ignoring long-term consequences. Leike et al. emphasize that "accurately capturing the complexity of human values in objective functions is crucial to avoid specification gaming and ensure proper alignment" (Leike et al. 2018).

To mitigate specification gaming, better objective function design is essential. This involves incorporating broader context and constraints into the objectives and regularly updating them based on feedback. Iterative testing and validation are also critical. AI behavior must be continuously tested in diverse scenarios, using simulation environments to identify and fix exploits. Everitt and Hutter discuss the importance of "robust objective functions and rigorous testing to prevent specification gaming and achieve reliable AI alignment" (Everitt and Hutter 2018). Clark and Amodei further highlight that "faulty reward functions can lead to unintended and potentially harmful AI behavior, necessitating ongoing refinement and validation" (Clark and Amodei 2016).

The metrics used to evaluate AI systems play a crucial role in outer alignment. Many AI metrics, such as BLEU, METEOR, and ROUGE, are chosen for their ease of measurement but do not necessarily capture human judgment (Hardt and Recht 2021). These metrics can lead to specification gaming, as they may not align with the true objectives we want the AI to achieve. Similarly, using SAT scores to measure LLM performance may not predict real-world task effectiveness, highlighting the need for more contextually relevant benchmarks (Chowdhery et al. 2022). The word error rate (WER) used in speech recognition is another example; it does not account for semantic errors, leading to misleading conclusions about the system's performance (Xiong et al. 2016).

A classic example comes from six years ago with the claim that a system "Achieve[d] human parity in conversation speech recognition" (Xiong et al. 2016). However, we know from experience that captioning services have only recently begun to transcribe speech passably, whether in online meetings or web videos. What happened? In this case, researchers showed their system beat the human baseline—the error rate when transcribing films. However, there were issues with their approach. First, they used a poor measure of a human baseline by hiring untrained Mturk annotators instead of professional captioners. Second, the metric itself, the word error rate (WER), was flawed. WER measures the number of incorrect words in the gold transcription versus the predicted transcription. Consider what the metric hides when it says that two systems both have an error rate of six percent. This does not mean the systems are equivalent. One might substitute "a" for "the," while the other substitutes "tarantula" for "banana." The metric was not sensitive to semantic errors, so a model could outperform humans in WER yet still make unintelligent, highly unsemantic mistakes.

5.1.4.2 *Inner Alignment: Preventing Goal Misgeneralization*

Assume we have perfectly specified human values in a reward model. An issue remains: given finite training data, many models perform well on the training set, but each will generalize somewhat differently. How do we choose models that correctly generalize to new distributions? This is the problem of *goal misgeneralization*, also known as the *inner alignment problem*, where a learned algorithm performs well on the training set but generalizes poorly to new input distributions, achieving low rewards even on the reward function it was trained on. Inner alignment ensures that the learned goals and behaviors of an AI system align with the intended objectives during deployment, whereas goal misgeneralization occurs when an AI system applies learned goals inappropriately to new situations (Hubinger et al. 2019).

Consider the following example of goal misgeneralization from (Shah et al. 2022). The setup involves a never-ending reinforcement learning environment without discrete episodes. The agent navigates a grid world where it can collect rewards by chopping trees. Trees regenerate at a rate dependent on the number left; they replenish slowly when few remain. The optimal policy is to chop trees sustainably, i.e., fewer when they are scarce. However, the agent does not initially learn the optimal policy.

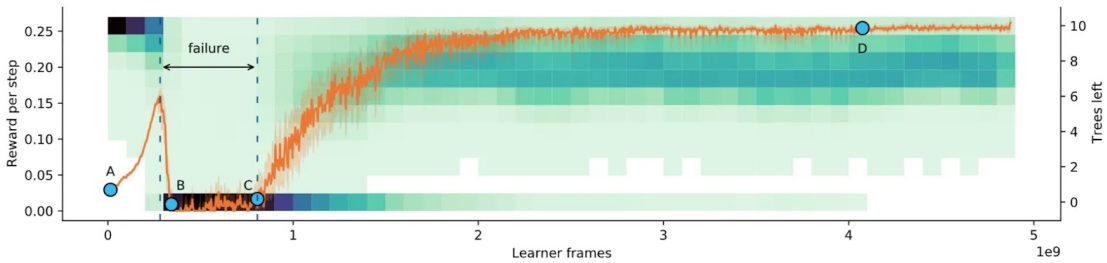


Figure 5.4
The agent's performance in Tree Gridworld. The reward is shown in orange, and the green distribution indicates the number of remaining trees.

Initially, the agent is inefficient at chopping trees, keeping the tree population high (point A). As it improves its chopping skills, it over-harvests, leading to deforestation and a prolonged period of minimal reward (between points B and C). Eventually, it learns sustainable chopping (point D). This scenario (up to point C) exemplifies goal misgeneralization. When the agent first becomes proficient at chopping (between points A and B), it faces a range of potential goals, from sustainable to rapid tree chopping. All these goals align with the (well-specified) reward function and its experience of being rewarded for increased efficiency. Unfortunately, it adopts the detrimental goal of rapid deforestation, resulting in a prolonged period of low reward.

Another example of goal misgeneralization occurs in recommendation systems. These systems aim to maximize user engagement, which can inadvertently lead to promoting extreme or sensational content. Krakovna et al. highlights that "recommendation systems can misgeneralize by prioritizing content that maximizes clicks or watch time, even if it involves promoting harmful or misleading information" (Krakovna et al. 2020). This misalignment between the

system's learned objective (engagement) and the intended objective (informative and beneficial content) exemplifies how goal misgeneralization can manifest in real-world applications.

Autonomous vehicles also present cases of goal misgeneralization. These vehicles must interpret and respond to various signals in their environment. However, in rare scenarios, they may misinterpret signals, leading to unsafe maneuvers. Amodei et al. discuss that "autonomous vehicles can exhibit unsafe behaviors when faced with uncommon situations that were not well-represented in the training data, demonstrating a misgeneralization of their learned driving policies" (Amodei et al. 2016). Ensuring that autonomous vehicles generalize correctly to all possible driving conditions remains a significant challenge.

To address goal misgeneralization, robust training procedures are essential. This involves using diverse and representative training data to cover a wide range of scenarios and incorporating adversarial training to handle edge cases. Leike et al. (Leike et al. 2018) emphasize the importance of "robust training procedures that include diverse datasets and adversarial examples to improve the generalization of AI systems". Additionally, careful specification of learning goals is crucial. This means defining clear and comprehensive objectives and regularly reviewing and adjusting these goals based on performance and feedback. Hubinger et al. suggests that "regularly updating and refining the objectives based on ongoing evaluation can help mitigate the risks of goal misgeneralization" (Hubinger et al. 2019).

A key concern about goal misgeneralization in competent, general systems is that a policy successfully models the preferences of human raters (or the reward model) and behaves accordingly to maximize reward during training. However, it may deviate catastrophically from human preferences when given a different input distribution during deployment, such as during an unexpected geopolitical conflict or when facing novel technological developments. Increasing data size, regularization, and red-teaming can help mitigate goal misgeneralization, but they do not fundamentally solve the problem. Understanding the inductive biases of optimization algorithms and model families may help address the problem more generally.

So, can you differentiate between inner and outer alignment?

The distinction between inner and outer alignment can be a bit subtle. The following four cases, from (Ngo, Chan, and Mindermann 2023), may help to clarify the difference:

- The policy behaves incompetently. This is a capability generalization failure.
- The policy behaves competently and desirably. This is aligned behavior.
- The policy behaves in a competent yet undesirable way which gets a high reward according to the original reward function. This is an outer alignment failure, also known as reward misspecification.
- The policy behaves in a competent yet undesirable way which gets a low reward according to the original reward function. This is an inner alignment failure, also known as goal misgeneralization.

Now that we understand the alignment problem overall, we move on to the specific techniques used for value learning to ensure AI systems are aligned with human values.

5.1.5 Techniques in Value Learning

Various methods in value learning for foundation models have been explored in great detail in recent years (Stiennon et al. 2020). Using binary human-labeled feedback to make models closely aligned to human preferences is particularly difficult in scenarios where large datasets inherently encompass suboptimal behaviors. The approach of Reinforcement Learning from Human Feedback (RLHF) ((Ouyang et al. 2022)) has risen to prominence as an effective method for addressing this issue. The technique applies to various domains, from prompt-image alignment, fine-tuning large language models or diffusion models, and improving the performance of robot policies.

5.1.5.1 Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) is a technique used to align AI behavior with human values by incorporating human feedback into the reinforcement learning process. This approach is particularly effective when large datasets inherently encompass suboptimal behaviors. RLHF aims to refine policies by discriminating between desirable and undesirable actions, ensuring that AI systems act following human preferences (Ouyang et al. 2022).

The core concept of RLHF: It first trains a reward model using a dataset of binary preferences gathered from human feedback. This reward model is then used to fine-tune the AI model through a reinforcement learning algorithm. The core concept is to utilize human feedback to guide AI learning, thereby aligning the AI's behavior with human expectations (Stiennon et al. 2020).

The RLHF pipeline involves the following steps:

Step 1: Supervised Fine-Tuning

In the initial step for language modeling tasks, we utilize a high-quality dataset consisting of (prompt, response) pairs to train the model. Prompts are sampled from a curated dataset designed to cover a wide range of instructions and queries, such as "Explain the moon landing to a 6-year-old." Trained human labelers provide the desired output behavior for each prompt, ensuring responses are accurate, clear, and aligned with task goals. For instance, in response to the moon landing prompt, a labeler might generate, "Some people went to the moon in a big rocket and explored its surface." The collected (prompt, response) pairs serve as the training data for the model, with the cross-entropy loss function applied only to the response tokens. This helps the model learn to generate responses that are closely aligned with the human-provided examples. The training process adjusts model parameters through supervised learning, minimizing the difference between the model's predictions and the human responses.

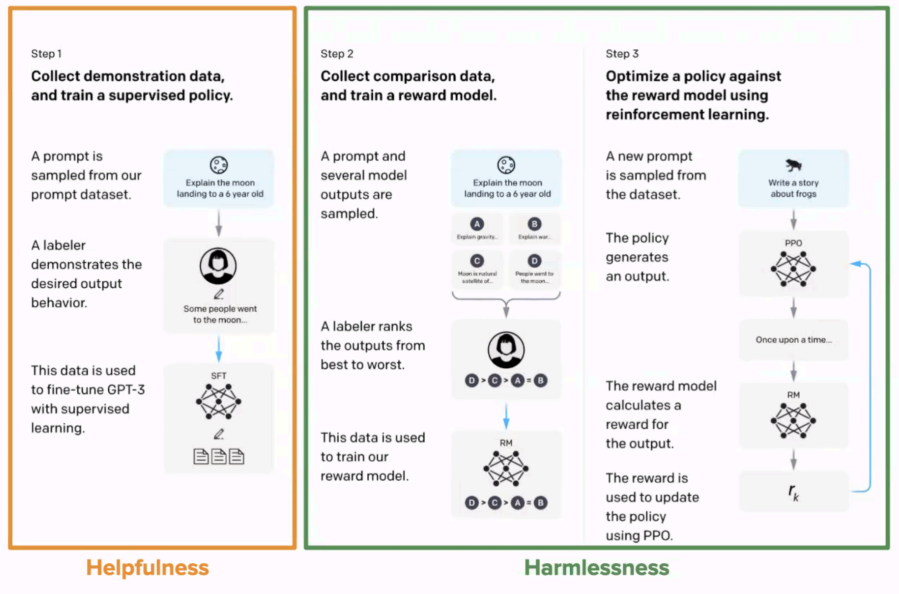


Figure 5.5
The above diagram depicts the three steps in the traditional RLHF pipeline: (a) supervised fine-tuning, (b) reward model (RM) training, and (c) reinforcement learning via proximal policy optimization (PPO) on this reward model. Image taken from (Ouyang et al. 2022).

Step 2: Reward Model (RM) Training

In this step, we train a reward model to score any (prompt, response) pair and produce a meaningful scalar value. Multiple model-generated responses are sampled for each prompt. Human labelers then rank these responses from best to worst based on their quality and alignment with the prompt. For example, given the prompt "Explain the moon landing to a 6-year-old," responses like "People went to the moon in a big rocket and explored its surface" might be ranked higher than "The moon is a natural satellite of Earth." The rankings provided by the labelers are used to train the reward model Φ_{RM}. The model is trained by minimizing the following loss function across all training samples:

$$\mathbb{L}(\Phi_{RM}) = -\mathbb{E}_{(x,y_e,i\to D_{RL})}[\log(\sigma(\Phi_{RM}(x,y_i)) - \Phi_{RM}(x,y_{1-i}))]$$

for $i \in \{0,1\}$. This loss function encourages the reward model to produce higher scores for better-ranked responses, thereby learning to evaluate the quality of model outputs effectively.

Step 3: Reinforcement Learning

In this step, we refine the policy using reinforcement learning (RL) based on the rewards provided by the trained reward model. A new prompt is sampled from the dataset, and the policy generates an output. The reward model then calculates a reward for this output, and the reward is used to update the policy using the Proximal Policy Optimization (PPO) algorithm.

The RL setting is defined as follows:

1. *Action Space*: The set of all possible actions the agent can take, which, for language models, is typically the set of all possible completions.
2. *Policy*: A probability distribution over the action space. In the case of language models like LLM, the policy is contained within the model and represents the probability of predicting each completion.
3. *Observations*: The inputs to the policy, which in this context are prompts sampled from a certain distribution.
4. *Reward*: A numerical score provided by the Reward Model (RM) that indicates the quality of actions taken by the agent.

During training, batches of prompts are sampled from two distinct distributions, namely either D_{RL}, the distribution of prompts explicitly used for the RL model, or D_{pretrain}, the distribution of prompts from the pre-trained model. The objective for the RL agent is to maximize the reward while ensuring that the policy does not deviate significantly from the supervised fine-tuned model and does not degrade the performance on tasks the pre-trained model was optimized for. When sampling a response y to a prompt x from D_{RL}, the first objective function is:

$$\text{objective}_1(x_{RL}, y; \phi) = RM(x_{RL}, y) - \beta \log \frac{\text{LLM}_\phi^{RL}(y|x)}{\text{LLM}_{SFT}(y|x)}$$

Where the first term is the reward from the RM, and the second term is the Kullback-Leibler (KL) divergence, weighted by a factor β, which acts as a regularizer to prevent the RL model from straying too far from the SFT model. Further, for each x from D_{pretrain}, the second objective is to ensure that the RL model's performance on text completion does not worsen:

$$\text{objective}_2(x_{\text{pretrain}}; \phi) = \gamma \log \text{LLM}_\phi^{RL}(x_{\text{pretrain}})$$

where γ is a weighting factor that balances the influence of this objective against the others.

The final objective function is a sum of the expected values of the two objectives described above, across both distributions. In the RL setting, we maximize *this* objective function:

$$\text{objective}(\phi) = E_{(x,y)\sim D_\phi^{RL}}[RM(x,y) - \beta \log \frac{\text{LLM}_\phi^{RL}(y|x)}{\text{LLM}_{SFT}(y|x)}] + \gamma E_{x\sim D_{\text{pretrain}}}[\log \text{LLM}_\phi^{RL}(x)]$$

In practice, the second part of the objective is often not used to perform RLHF. The KL penalty is typically enough to constrain the RL policy. This function balances the drive to maximize

the reward with the need to maintain the quality of text completion and the similarity to the behavior of the supervised fine-tuned model.

Limitations and Challenges: Despite its successes, RLHF faces several challenges. One major issue is the quality of human feedback, which can be inconsistent and subjective. Scalability is another concern, as obtaining a large amount of high-quality feedback can be expensive and time-consuming. Over-optimization and hallucinations, where the model generates plausible but incorrect outputs, are also common problems. This generally stems from temporal credit assignment and the instability of approximate dynamic programming (Hasselt et al. 2018). Further, it is expensive to gather tens of thousands of preferences over datasets to create robust reward models. Strategies to overcome these challenges include using diverse and representative training data, incorporating adversarial training to handle edge cases, and continuously refining the reward model based on ongoing feedback and performance evaluations (Leike et al. 2018).

5.1.5.2 Contrastive Preference Learning

Contrastive Preference Learning (CPL) is a learning paradigm designed to enhance the alignment of AI systems with human preferences without relying on traditional reinforcement learning (RL) methods. CPL addresses many limitations inherent in traditional RLHF techniques by learning from human comparisons rather than explicit reward signals. This section provides an in-depth exploration of CPL, detailing its methodology, experiments, results, and potential challenges. Recent research has shown that human preferences are often better modeled by the optimal advantage function or regret, rather than traditional reward functions used in RLHF. Traditional RLHF approaches, which learn a reward function from a preference model and then apply RL, incur significant computational expenses and complexity (Hejna et al. 2023). CPL offers a streamlined and scalable alternative by leveraging a more accurate regret model of human preferences.

The key idea of CPL is the substitution of the optimal advantage function with the log probability of the policy in a maximum entropy reinforcement learning framework. This substitution is beneficial as it circumvents the need to learn the advantage function and avoids the optimization challenges associated with RL-like algorithms. By using the log probability of the policy, CPL more closely aligns with how humans model preferences and enables efficient supervised learning from human feedback.

CPL is a structured approach to aligning AI behavior with human preferences by relying on a dataset of preferred behavior segments $\mathcal{D}_{\text{pref}} = \{(\sigma_i^+, \sigma_i^-)\}_{i=1}^n$, where $\sigma^+ \succ \sigma^-$. Each behavior segment σ is a sequence of states and actions, $\sigma = (s_1, a_1, s_2, a_2, \ldots, s_k, a_k)$. The CPL approach aims to maximize the expected sum of rewards minus an entropy term, which promotes exploration and prevents overfitting to specific actions:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t) - \alpha \log \pi(a_t|s_t)) \right]$$

where γ is the discount factor, α is the temperature parameter controlling the stochasticity of the policy, and r is the reward function. This step sets the foundation by defining the optimization objective that the CPL model strives to achieve. In the learning process, CPL compares the log probabilities of actions in preferred segments σ^+ against those in non-preferred segments σ^- :

$$\mathbb{L}_{CPL}(\pi_\theta, \mathcal{D}_{\text{pref}}) = \mathbb{E}_{(\sigma^+,\sigma^-)\sim\mathcal{D}_{\text{pref}}}\left[-\log \frac{\exp(\sum_{\sigma^+}\gamma^t\alpha\log\pi_\theta(a_t^+|s_t^+))}{\exp(\sum_{\sigma^+}\gamma^t\alpha\log\pi_\theta(a_t^+|s_t^+)) + \exp(\sum_{\sigma^-}\gamma^t\alpha\log\pi_\theta(a_t^-|s_t^-))}\right]$$

This comparison allows the model to learn which actions are more aligned with human preferences, forming the core learning mechanism of CPL. The preference model for CPL is regret-based, described as

$$P_{A^*}[\sigma^+ \succ \sigma^-] = \frac{\exp(\sum_{\sigma^+}\gamma^t A^*(s_t^+, a_t^+))}{\exp(\sum_{\sigma^+}\gamma^t A^*(s_t^+, a_t^+)) + \exp(\sum_{\sigma^-}\gamma^t A^*(s_t^-, a_t^-))}$$

where $A^*(s_t, a_t)$ represents the advantage function and is a matrix. This step models human preferences based on regret, reflecting how humans might evaluate different behaviors.

One hypothesis as to why one might consider a regret-based model more useful over a sum-of-rewards, Bradley-Terry model is that humans likely think of preferences based on the regret of each behavior under the optimal policy of the expert's reward function.

The key insight that the paper leverages is that from (Ziebart 2010) in MaxEnt Offline RL. In this general setting, (Ziebart 2010) shows that one can write that the optimal advantage function is related to the optimal policy by $A_r^*(s,a) = \alpha\log\pi^*(a|s)$. Therefore, the loss function for CPL can be written by substituting the above result to obtain:

$$L_{CPL}(\pi_\theta, \mathcal{D}_{\text{pref}}) = \mathbb{E}_{(\sigma^+,\sigma^-)\sim\mathcal{D}_{\text{pref}}}\left[-\log P_{\pi_\theta}[\sigma^+ \succ \sigma^-]\right]$$

One merit of using CPL over the typical RLHF pipeline is that it can lead to a deduction in mode collapse. Further, it makes reward misgeneralization failures less likely, enhancing the reliability of the learned policy. However, the approach still has a few limitations:

1. CPL assumes knowledge of the human rater's temporal discounting (i.e., of the discount factor γ), which in practice would be difficult to communicate.
2. CPL's loss function is computed over segments, it requires a substantial amount of GPU memory for large segment sizes.

How does RLHF with PPO and CPL compare their effectiveness and applicability in aligning AI systems with human values?

The ongoing challenge in aligning foundation models in the future will be to refine these methodologies further, balancing computational feasibility with the sophistication needed to capture the intricacies of human values and countering failure modes such as reward over-optimization. In conclusion, exploring value learning through RLHF and CPL methods has enriched our understanding of integrating human preferences into foundation models. To provide a well-rounded perspective on aligning AI systems with human values, the following table highlights a detailed comparison of RLHF with PPO and CPL, emphasizing their advantages, limitations, and ideal scenarios.

Table 5.1
Comparison between RLHF with PPO and CPL

	RLHF with PPO	CPL
Strengths	– Excels in optimizing policies through reinforcement learning – Suitable for tasks that benefit from iterative improvement – Effective in continuous action spaces	– Emphasizes regret and optimality rather than reward maximization – Reduces computational overhead – Aligns more closely with human preferences – Avoids reward over-optimization – More scalable due to reliance on supervised learning techniques
Limitations	– Faces limitations in handling complex preference structures – High computational cost – Susceptible to reward misgeneralization	– May struggle in environments where direct human feedback is less accessible – Depends on high-quality preference data for effective training

	RLHF with PPO	CPL
Ideal Scenarios	– Tasks with well-defined reward functions – Environments allowing extensive interaction and feedback	– Environments where human feedback is more accessible than well-defined reward functions – Tasks requiring computational efficiency and scalability

5.1.6 Value Alignment Verification

After we discuss the techniques of value learning, it becomes evident that aligning machine behavior with human values, while advanced, is inherently approximate and not infallible. This realization underscores the importance of value alignment verification—a methodology to ensure that the values imparted to a machine truly reflect those of a human. Human-robot value alignment has been explored through various lenses, including qualitative trust assessments (Huang et al. 2018), asymptotic alignment through active learning of human preferences (Hadfield-Menell et al. 2016; P. F. Christiano et al. 2017; Sadigh et al. 2017), and formal verification methods (Brown et al. 2021). This section will focus on the formal verification approach for value alignment as discussed in (Brown et al. 2021). Unless otherwise stated, all information presented here is derived from (Brown et al. 2021). This approach aims to ensure that the values imparted to a machine align with those of a human.

To begin with, consider an MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, and transition model $\mathcal{T}$. This formal framework allows us to model the environment in which humans and robots operate. Denote the human's reward function as R and the robot's reward function as R'. Both the human and robot reward functions must be linear in a set of shared features, defined as:

$$R(s) = \mathbf{w}^\top \phi(s), R'(s) = \mathbf{w'}^\top \phi(s).$$

These linear reward functions provide a common ground for comparing human and robot preferences.

Next, the optimal state-action value function, which indicates the expected cumulative reward of following a policy π starting from state s and action a, but we follow the notation in (Brown et al. 2021) for simplicity. The optimal state-action value function is given by:

$$Q_R^\pi(s,a) = \mathbf{w}^\top \Phi_{\pi_R}^{(s,a)}, \Phi_{\pi_R}^{(s,a)} = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | s_0 = s, a_0 = a].$$

Here, $\Phi_{\pi_R}^{(s,a)}$ is the feature expectation vector under policy π, capturing the long-term feature visitation frequencies. We overload the action space notation to define the set of all optimal actions given a state as

$$\mathcal{A}_R(s) = \underset{x}{\text{argmax}} \; Q_R^{\pi^*}(s,a)$$

where π^* is an optimal policy. We can now define the aligned reward polytope (ARP). The ARP is the set of all weights w that satisfy the following set of strict linear inequalities, $\mathbf{w}^\top \mathbf{A} > \mathbf{0}$ where each row of $\mathbf{A}$ corresponds to $\Phi_{\pi_R^*}^{(s,a)} - \Phi_{\pi_R^*}^{(s,b)}$ for a single (s,a,b) tuple where $s \in \mathcal{S}, a \in \mathcal{A}_R(s), b \notin \mathcal{A}_R(s)$. Thus, to construct $\mathbf{A}$, one must loop over all (s,a,b) tuples which has complexity $O(|\mathcal{S}| \cdot |\mathcal{A}|^2)$. This construction ensures that the weights $\mathbf{w}$ align with the human's optimal actions across all states.

The intuition behind the ARP is that we use the human optimal policy for each state to determine what actions are optimal and what are suboptimal at this state. Then, for every one of those combinations, we can place a linear inequality on the set of reward weights consistent with that optimal vs suboptimal action bifurcation. One of the key assumptions that let us do this is that we assume both the human and the robot act optimally according to their reward function. This is known as a *rationality assumption* and provides the link between actions and rewards that we need.

For illustration, consider a simple grid world environment. **?@fig-toy** shows the optimal policy and the corresponding ARP. The optimal policy reveals that the gray state is less preferred compared to the white states, which is reflected in the ARP (hatched region of **?@fig-toy**).

Optimal policy (a) and aligned reward polytope (ARP) (b) for a grid world with two features (white and gray) and a linear reward function (R(s) = w0 · 1white(s) + w1 · 1gray(s)). The ARP is denoted by the hatched region in (b).

Computing the ARP exactly can be computationally demanding or we may not have access to the robot's reward function. This section describes heuristics for testing value alignment in the case the robot's reward weights ($\mathbf{w}'$) are unknown, but the robot's policy can be queried. Heuristics provide simplified methods to estimate value alignment without the need for exhaustive computations.

ARP-blackbox: The ARP black-box (ARP-bb) heuristic helps address the challenge of computing the ARP by allowing users to work with a simplified model. In this heuristic, the user first solves for the ARP and removes all redundant half-space constraints. For each remaining half-space constraint, the user queries the robot's action at the corresponding state. The intuition here is that states, where different actions are taken, reveal crucial information about the reward function. By focusing on these key states, we can gain insights into the robot's reward function without needing to know it explicitly.

Set Cover Optimal Teaching: The Set Cover Optimal Teaching (SCOT) heuristic uses techniques from (Brown and Niekum 2019) to generate maximally informative trajectories. These

trajectories are sequences of states where the number of optimal actions is limited, making them particularly informative for understanding the robot's policy. By querying the robot for actions along these trajectories, we can efficiently gauge the alignment of the robot's policy. This method helps to identify potential misalignments by focusing on critical decision points in the trajectories.

Critical States: The Critical States (CS) heuristic identifies states where the gap in value between the optimal action and an average action is significant. These states are crucial because if the robot's policy is misaligned, the misalignment will be most consequential at these critical states. By querying the robot's policy at these states, we can assess the alignment more effectively. This heuristic is particularly useful when we have a limited budget of states to check, as it prioritizes the most informative states for evaluation.

Practical Examples: To illustrate the concepts of value alignment verification, we present an example of applying value alignment verification in a simple MDP grid world environment. Consider a grid world where the human's reward function is defined as $R(s) = 50 \cdot \mathbf{1}_{green}(s) - 1 \cdot \mathbf{1}_{white}(s) - 50 \cdot \mathbf{1}_{blue}(s)$, where $\mathbf{1}_{color}(s)$ is an indicator feature for the color of the grid cell. The objective is to align the robot's policy with this reward function.

(a) optimal policy (b) preference query 1 (c) preference query 2 (d) ARP-bb queries (e) SCOT queries (f) CS queries. In the preference queries, the human reward model prefers black to orange.

?@fig-island (a) shows all optimal actions at each state according to the human's reward function. This optimal policy serves as the benchmark for alignment verification. **?@fig-island** (b) and **?@fig-island** (c) show two pairwise preference trajectory queries (black is preferable to orange according to ([eq: human_r])). Preference query 1 verifies that the robot values reaching the terminal goal state (green) rather than visiting more white states. Preference query 2 verifies that the robot values white states more than blue states. These two preference queries are all we need to determine whether the robot's values are aligned with the human's values.

Next, we apply the heuristics discussed in the previous section to this grid world example. **?@fig-island** (d), **?@fig-island** (e), and **?@fig-island** (f) show the action queries requested by the heuristics ARP-bb, SCOT, and CS. Each heuristic queries the robot's actions at specific states to assess alignment:

- **ARP-bb**: This heuristic queries the fewest states but is myopic. It focuses on critical states derived from the ARP.
- **SCOT**: This heuristic generates maximally informative trajectories, querying more states than necessary but providing a comprehensive assessment.
- **CS**: This heuristic queries many redundant states, focusing on those where the value gap between optimal and average actions is significant.

To pass the test given by each heuristic, the robot's action at each of the queried states must be optimal under the human's reward function. The example demonstrates that while the

ARP-bb heuristic is efficient, it might miss the broader context. SCOT provides a thorough assessment but at the cost of querying more states. CS focuses on high-impact states but includes redundant queries.

It is important to note that both the construction of the ARP and the heuristics rely on having an optimal policy for the human. Thus, in most practical settings we would simply use that policy on the robot without needing to bother with value alignment verification. As such, value alignment verification as presented here is more of an academic exercise rather than a tool of practical utility.

5.2 Human-Centered Design

After understanding AI alignment, the next step is to explore practical methodologies for incorporating user feedback and ensuring that AI systems not only align with but also cater to the needs and preferences of their users. This section will provide insights into various Human-Centered Design techniques and their application in creating AI systems that are intuitive and ethically sound, ultimately enhancing the human-AI interaction experience.

5.2.1 AI and Human-Computer Interaction

Human-Computer Interaction (HCI) is critical in the context of artificial intelligence because it focuses on designing systems that are intuitive and responsive to human needs. While human-robot interaction and other forms of human interaction with technology are important, HCI specifically addresses the broader and more common interfaces that people interact with daily. HCI principles ensure that AI systems are not only functional but also accessible and user-friendly, making them essential for the successful integration of AI into everyday life. By focusing on HCI, we can leverage established methodologies and insights to create AI systems that are more aligned with human values and needs.

At the heart of this exploration is the concept of human-in-the-loop processes. As AI systems become more sophisticated, their ability to simulate human decision-making processes and behaviors has increased, leading to innovative applications across various domains. The presentation by Meredith Morris, titled "Human-in-the-loop Computing: Reimagining Human-Computer Interaction in the Age of AI," shows work in the integration of human intelligence with AI capabilities (Morris 2019). Projects like Soylent and LaMPost are highlighted as exemplary cases of this integration. Soylent is a Word plugin that uses human computation to help with editing tasks, while LaMPost is a platform that leverages crowd workers to aid in natural language processing tasks (Bernstein et al. 2010; Project 2017). These examples demonstrate how human input can significantly enhance AI outputs by leveraging the unique strengths of human cognition, thereby addressing complex AI problems that were previously unsolvable. For instance, Soylent can improve text quality by incorporating nuanced human feedback, and LaMPost can refine NLP tasks by incorporating human insights into language subtleties, both of which go beyond the capabilities of fully automated systems. However, the integration of human elements in AI systems brings up critical ethical considerations. The presentation

discusses the changing perceptions of the ethics of human-in-the-loop processes. While the cost-effectiveness of human data labeling and other processes was once seen as beneficial, it is the ethical implications of such interactions that take precedence nowadays. This shift underscores the evolving norms in HCI and the importance of considering the ethical dimensions of human-AI interactions.

The role of diverse human perspectives plays a crucial role in enhancing AI systems. Involving a broad spectrum of users in the development and testing of AI systems ensures that these technologies are inclusive and representative of the global population, moving beyond the limitations of a WEIRD (Western, Educated, Industrialized, Rich, and Democratic) user base. The methodologies for collecting user feedback in HCI form a critical part of this discussion since they are vital in understanding user needs, preferences, and behaviors, which in turn inform the development of more user-centered AI systems. The presentation by Meredith Morris (Morris 2019) also highlights how these methods can be effectively employed to gain insights from users to ensure that AI systems are aligned with the real-world needs and expectations of users. In HCI, collecting user feedback is a fraught problem. When interacting with AI systems, the typical end user simply cares about tasks that the system can perform. Thus, a key question in HCI for AI is finding and understanding these tasks. **Methodologies for collecting user feedback in HCI**, are described as follow:

- **Storyboarding** is a visual method used to predict and explore the user experience with a product or service. A storyboard in HCI is typically a sequence of drawings with annotations that represent a user's interactions with technology. This technique is borrowed from the film and animation industry and is used in HCI to convey a sequence of events or user flows, including the user's actions, reactions, and emotions.
- **Wizard of Oz Studies** is a method of user testing where participants interact with a system they believe to be autonomous, but which is actually being controlled or partially controlled by a human 'wizard' behind the scenes. This technique allows researchers to simulate the response of a system that may not yet be fully functional or developed.

Both **Storyboarding** and **Wizard of Oz Studies** are effective for engaging with users early in the design process. They help deal with the problem of gathering feedback on a product that doesn't yet exist. Users often have difficulty imagining outcomes when they cannot touch a live demonstration.

- **Surveys** in HCI are structured tools that consist of a series of questions designed to be answered by a large number of participants. They can be conducted online, by telephone, through paper questionnaires, or using computer-assisted methods. Surveys are useful for collecting quantitative data from a broad audience, which can be analyzed statistically.
- **Interviews** in HCI are more in-depth and involve direct, two-way communication between the researcher and the participant. Interviews can be structured, semi-structured, or unstructured, ranging from tightly scripted question sets to open-ended conversations.

- **Focus Groups** involve a small group of participants discussing their experiences and opinions about a system or design, often with a moderator. Group dynamics can provide insights into collective user perspectives. In particular, users can bounce ideas off each other to provide richer feedback and quieter users who may not otherwise provide feedback may be encouraged by their peers.
- **Community-Based Participatory Design (CBPD)** is a human-centered approach that involves the people who will use a product in the design and development process. With CBPD, designers work closely with community members to identify problems, develop prototypes, and iterate based on community feedback. For example, when building a software product for deaf people, the engineering team can hire deaf engineers or designers to provide feedback as they collaboratively build the product.
- **Field Studies** involve observing and collecting data on how users interact with a system in their natural environment. This method is based on the premise that observing users in their context provides a more accurate understanding of user behavior. It can include a variety of techniques like ethnography, contextual inquiries, and natural observations.
- **Lab-based studies** are conducted in a controlled environment where the researchers can manipulate variables and observe user behavior in a setting designed to minimize external influences. Common lab-based methods include usability testing, controlled experiments, and eye-tracking studies.
- **Diary Studies and Ethnography** in HCI are a research method where participants are asked to keep a record of their interactions with a system or product over a while. This log may include text, images, and sometimes even audio or video recordings, depending on the study's design. Participants typically document their activities, thoughts, feelings, and frustrations as they occur in their natural context.
- **Ethnography** is a qualitative research method that involves observing and interacting with participants in their real-life environment. Ethnographers aim to immerse themselves in the user environment to get a deep understanding of the cultural, social, and organizational contexts that shape technology use.

As we have explored various methodologies for collecting human feedback, it becomes evident that the role of human input is indispensable in shaping AI systems that are not only effective but also ethically sound and user-centric. In the next step, we will elaborate on how to design AI systems for positive human impact, examining how socially aware and human-centered approaches can be employed to ensure that AI technologies contribute meaningfully to society. This includes understanding how AI can be utilized to address real-world challenges and create tangible benefits for individuals and communities.

5.2.2 Designing AI for Positive Human Impact

In the field of natural language processing (NLP), the primary focus has traditionally been on quantitative metrics such as performance benchmarks, accuracy, and computations. These

metrics have long guided the development and evaluation of the technologies. However, as the field evolves and becomes increasingly intertwined with human interactions like the recent popularity of Large Language Models (LLMs), a paradigm shift is becoming increasingly necessary. For example, these LLMs are shown to produce unethical or harmful responses or reflect values that only represent a certain group of people. The need for a human-centered approach in NLP development is crucial as these models are much more likely to be utilized in a broad spectrum of human-centric applications, impacting various aspects of daily life. This shift calls for an inclusive framework where LLMs are not only optimized for efficiency and accuracy but are also sensitized to ethical, cultural, and societal contexts. Integrating a human-centered perspective ensures that these models are developed with a deep understanding of, and respect for, the diversity and complexity of human values and social norms. This approach goes beyond merely preventing harmful outcomes; it also focuses on enhancing the positive impact of NLP technologies on society. In this session, we explore the intricacies of a human-centered approach in NLP development, focusing on three key themes: Socially Aware, Human-Centered, and Positively Impactful.

5.2.2.1 Socially Aware

In the exploration of socially aware NLP, (Hovy and Yang 2021) presents a comprehensive taxonomy of seven social factors grounded in linguistic theory (See Figure 5.6).

Figure 5.6
Taxonomy of social factors

This taxonomy illustrates both the current limitations and progressions in NLP as they pertain to each of these factors. The primary aim is to motivate the NLP community to integrate these social factors more effectively, thereby advancing towards a level of language understanding that more closely resembles human capabilities. The characteristics of speakers, encompassing variables such as age, gender, ethnicity, social class, and dialect, play a crucial role in language processing. Certain languages or dialects, often categorized as low-resource, are spoken by vulnerable populations that require special consideration in NLP systems. In many cases, the dominant culture and values are over-represented, leading to an inadvertent marginalization of minority perspectives. These minority voices must be not only recognized but also given equitable representation in language models. Additionally, norms and context are vital components in understanding linguistic behavior. They dictate the appropriateness of language use in various social situations and settings. Recognizing and adapting to these norms is a critical

aspect of developing socially aware NLP systems that can effectively function across diverse social environments.

5.2.2.2 Human-Centered

The Human-Centered aspect of NLP development emphasizes the creation of language models that prioritize the needs, preferences, and well-being of human users. This involves integrating human-centered design principles throughout the development stages of LLMs, which are described as follows:

- **Task Formulation stage**: Human-centered NLP development begins with understanding the specific problems and contexts in which users operate. This involves collaborating with end-users to identify their needs and challenges, ensuring that the tasks addressed by the models are relevant and meaningful to them. By engaging with users early in the process, developers can create models that are not only technically robust but also practically useful.
- **Data Collection stage**: Human-centered principles ensure that the data used to train models is representative of the diverse user population. This includes collecting data from various demographic groups, languages, and cultural contexts to avoid biases that could lead to unfair or harmful outcomes. Ethical considerations are paramount, ensuring that data is collected with informed consent and respecting users' privacy.
- **Data Processing** in a human-centered approach involves carefully curating and annotating data to reflect the nuances of human language and behavior. This step includes filtering out potentially harmful content, addressing imbalances in the data, and ensuring that the labels and annotations are accurate and meaningful. By involving human annotators from diverse backgrounds, developers can capture a wider range of perspectives and reduce the risk of biased outputs.
- **Model Training** with a human-centered focus involves incorporating feedback from users and domain experts to fine-tune the models. This iterative process ensures that the models remain aligned with users' needs and preferences. Techniques such as active learning, where the model queries users for the most informative examples, can be employed to improve the model's performance.
- **Model Evaluation** in a human-centered framework goes beyond traditional metrics like accuracy and F1-score. It includes assessing the model's impact on users, its fairness, and its ability to handle real-world scenarios. User studies and A/B testing can provide valuable insights into how the model performs in practice and how it affects users' experiences.
- **Deployment** of human-centered NLP models involves continuous monitoring and feedback loops to ensure that the models remain effective and aligned with users' needs over time. This includes setting up mechanisms for users to report issues and provide feedback, which can then be used to update and improve the models. Ensuring transparency

in how the models operate and how user data is used also fosters trust and acceptance among users.

5.2.2.3 Positively Impactful

Building on the human-centered approach, it is crucial to consider how language models can be utilized and the broader impacts they can have on society.

Utilization: LLMs offer socially beneficial applications across various domains such as public policy, mental health, and education. In public policy, they assist in analyzing large volumes of data to inform decision-making processes. In mental health, LLMs can provide personalized therapy and even train therapists by simulating patient interactions. In the education sector, they enable personalized learning experiences and language assistance, making education more accessible and effective. These examples demonstrate the versatility of LLMs in contributing positively to critical areas of human life.

Impact: The deployment of NLP models, especially LLMs, has significant societal impacts. Positively, they enhance human productivity and creativity, offering tools and insights that streamline processes and foster innovative thinking. LLMs serve as powerful aids in various sectors, from education to industry, enhancing efficiency and enabling new forms of expression and problem-solving. it is essential to acknowledge the potential negative impacts. One major concern is the ability of LLMs to generate and spread misinformation. As these models become more adept at producing human-like text, distinguishing between AI-generated and human-created content becomes increasingly challenging. This raises issues of trust and reliability, with the risk of widespread dissemination of false or misleading information, which could have significant adverse effects on individuals and society.

By considering both the utilization and impact of LLMs, we can better harness their potential for positive societal contributions while mitigating the risks associated with their deployment. In conclusion, by thoughtfully integrating human-centered principles and ensuring positive impacts through feedback collection and ethical considerations, we can develop language models that not only enhance human well-being but also align closely with societal values. Building on these foundational principles, we now turn our attention to Adaptive User Interfaces, which exemplify the practical application of these concepts by personalizing interactions and improving user experiences in dynamic environments.

5.2.3 Adaptive User Interfaces

Adaptive user interfaces (AUIs) represent a significant advancement in personalizing user experiences by learning and adapting to individual preferences. This section will discuss the methodologies and applications of AUIs, highlighting their role in enhancing human-AI interaction through intelligent adaptation. The integration of AUIs within human-centered design paradigms ensures that AI systems not only meet user needs but also anticipate and adapt to their evolving preferences, thus maximizing positive human impact. Nowadays, consumers

have more choices than ever and the need for personalized and intelligent assistance to make sense of the vast amount of information presented to them is clear.

5.2.3.1 Key ideas

In general, personalized recommendation systems require a model or profile of the user. We categorize modeling approaches into four groups.

1. User-created profiles (usually done manually).
2. Manually defined groups (stereotypes) that each user is classified into.
3. Automatically learned groups (stereotypes) that each user is classified into.
4. Adaptively learned individual user models from interactions with the recommendation system.

The last approach is referred to as *adaptive user interfaces*. This approach promises that each user is given the most personalization possible, leading to better outcomes. In this session, we discuss recommendation systems that adaptively learn an individual's preferences and use that knowledge to intelligently recommend choices that the individual is more inclined to like.

The problem of learning individual models can be formalized, given as follows:

– a set of tasks requiring a user decision,
– a description for each task,
– a history of the user's decision on each task,

So then we can find a function that maps from task description (features) to user decisions. The task can be described from crowd-sourced data (a collaborative approach) or the measurable features of the task (a content-based approach). The content-based approaches for describing tasks will be focused on in this session. After understanding the framework for adaptive user interfaces now it is a good point to give some example applications to help ground the future discussion. Adaptive user interfaces have been developed for

– Command and form completion
– Email filtering and filing
– News selection and layout
– Browsing the internet
– Selecting movies and TV shows
– Online shopping
– In-car navigation

- Interactive scheduling
- Dialogue systems

among many other applications.

5.2.3.2 Design

The goal of an adaptive user interface is to create a software tool that reduces human effort by acquiring a user model based on past user interactions. This is analogous to the goal of machine learning (ML) which is to create a software tool that improves some task performance by acquiring knowledge based on partial task experience. The design of an adaptive user interface can be broken up into six steps:

1. **Formulating the Problem**: Given some task that an intelligent system could aid, the goal is to find a formulation that lets the assistant improve its performance over time by learning from interactions with a user. In this step the designer has to make design choices about what aspect of user behavior is predicted, and what is the proper level of granularity for description (i.e. what is a training example). This step usually involves formulating the problem into some sort of supervised learning framework.
2. **Engineering the Representation**: At this stage we have a formulation of a task in ML terms and we need to represent the behavior and user model in such a way that makes computational learning not only tractable but as easy as possible. In this step, the designer has to make design choices about what information is used to make predictions, and how that information is encoded and passed to the model.
3. **Collecting User Traces**: In this third step the goal is to find an effective way to collect traces (samples) of user behavior. The designer must choose how to translate traces into training data and also how to elicit traces from a user. An ideal adaptive user interface places no extra effort on the user to collect such traces.
4. **Modeling the User**: In this step the designer must decide what model class to use (neural network, decision tree, graphical model, etc.) and how to train the model (optimizer, step size, batch size, etc.). This step in the design process is usually given too much importance in academia. It is quite often the case that the success of an adaptive user interface is more sensitive to the other design steps.
5. **Using the Model Effectively**: At this stage the designer must decide how the model will be integrated into a software tool. Specifically, when and how is the model evaluated and how is the output of the model presented to the user? In addition, the designer must consider how to handle situations in which the model predictions are wrong. An ideal adaptive user interface will let the user take advantage of good predictions and ignore bad ones.

6. **Gaining User Acceptance**: The final step in the design process is to get users to try the system and ultimately adopt it. The initial attraction of users is often a marketing problem, but to retain users the system must be well-designed and easy to use.

5.2.3.3 Applications

After understanding the design of Adaptive User Interfaces, let's take a look at how we can apply it to real-world problems. We will summarize and analyze three different application areas of learning human preferences, which are driving route advisor (Rogers, Fiechter, and Langley 1999), destination selection (Langley et al. 1999), and resource scheduling (Gervasio, Iba, and Langley 1999).

1. **Driving Route Advisor**: The task of route selection involves determining a desirable path for a driver to take from their current location to a chosen destination, given the knowledge of available roads from a digital map. While computational route advisors exist in rental cars and online, they cannot personalize individual drivers' preferences, which is a gap that adaptive user interfaces aim to fill by learning and recommending routes tailored to the driver's unique choices and behaviors.

Here is an approach to route selection through learning individual drivers' route preferences.

- Formulation: Learn a "subjective" function to evaluate entire routes.
- Representation: Global route features are computable from digital maps.
- Data collection: Preference of one complete route over another.
- Induction: A method for learning weights from preference data.
- Using model: Apply subjective function to find "optimal" route.

This method aims to learn a user model that considers the entirety of a route, thereby avoiding issues like data fragmentation and credit assignment problems.

The design choices are incorporated into (Rogers, Fiechter, and Langley 1999), which: models driver preferences in terms of 14 global route features; gives the driver two alternative routes he might take; lets the driver refine these choices along route dimensions; uses driver choices to refine its model of his preferences; and invokes the driver model to recommend future routes. We note that providing drivers with choices lets the system collect data on route preferences in an unobtrusive manner. The interface of the application is presented in Figure 5.7.

In driving route advisor task (Rogers, Fiechter, and Langley 1999), a linear model is used for predicting the cost of a route based on the time, distance, number of intersections, and the number of turns. The system uses each training pair as a constraint on the weights found during the learning process. The experimental results are shown in the **?@fig-exp-2**.

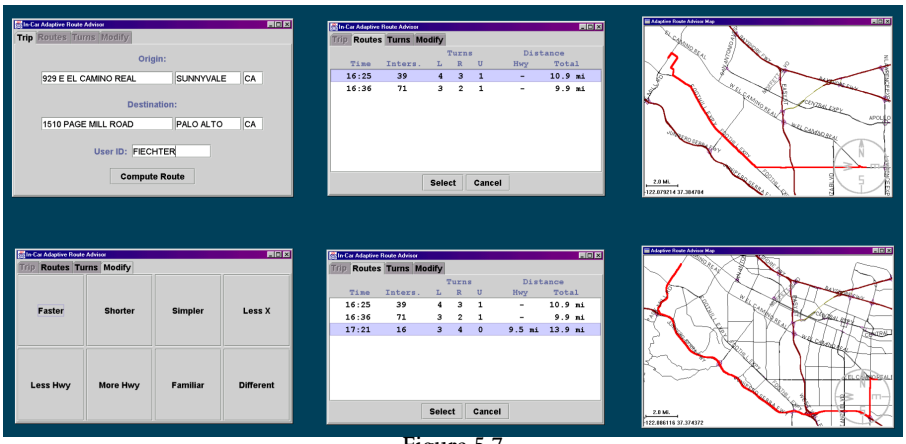


Figure 5.7
The adaptive route advisor.

(Left) Experiments with 24 subjects show the Route Advisor improves its predictive ability rapidly with experience. (Right) Analyses also show that personalized user models produce better results than generalized models, even when given more data.

2. **Destination Selection**: The task of destination selection involves assisting a driver in identifying one or more suitable destinations that fulfill a specific goal, such as finding a place to eat lunch, based on the driver's current location and knowledge of nearby options. While there are many recommendation systems online, including those for restaurants, they are not ideally suited for drivers due to the driving environment's demand for limited visual attention, thus necessitating a more tailored and accessible approach for in-car use.

One approach to destination recommendation can be cast as:

- Formulation: Learn to predict features the user cares about in items.
- Representation: Conditions/weights on attributes and values.
- Data collection: Converse with the user to help him make decisions, noting whether he accepts or rejects questions and items.
- Induction: Any supervised induction method.
- Using model: Guide the dialogue by selecting informative questions and suggesting likely values.

This design relies on the idea of a conversational user interface. Spoken-language versions of this approach appear well suited to the driving environment.

This approach is implemented in (Langley et al. 1999), where it engages in spoken conversations to help a user refine goals; incorporates a dialogue model to constrain this process; collects and stores traces of interaction with the user; and personalizes both its questions and recommended items. Their work focused on recommending restaurants to users who want advice

about where to eat. This approach to recommendation would work well for drivers, it also has broader applications. We present experimental results in

(Left) Speech Acts Per Conversation. (Right) Time Per Conversation.

3. **Resource Scheduling**: The task of resource scheduling describes the challenge of allocating a limited set of resources to complete a set of tasks or jobs within a certain time frame, while also considering the constraints on both the jobs and the resources. Although automated scheduling systems are prevalent in various industries and some interactive schedulers exist, there is a distinct need for systems that can create personalized schedules reflecting the unique preferences of individual users.

An approach to personalized scheduling can be described as:

- Formulation: Learn a utility function to evaluate entire schedules.
- Representation: Global features are computable from the schedule.
- Data collection: Preference of one candidate schedule over others.
- Induction: A method for learning weights from preference data.
- Using model: Apply the 'subjective' function to find a good schedule.

We note that this method is similar to that in the Adaptive Route Advisor. However, it assumes a search through a space of complete schedules (a repair space), which requires some initial schedule. This approach is implemented in (Gervasio, Iba, and Langley 1999), where the interactive scheduler retrieves an initial schedule from a personalized case library; suggests to the user improved schedules from which to select; lets the user direct search to improve on certain dimensions; collects user choices to refine its personalized utility function; stores solutions in the case base to initialize future schedules; and invokes the user model to recommend future schedule repairs. As before, providing users with choices lets the system collect data on schedule preferences unobtrusively. An example of the interface, and the experimental results are shown in **?@fig-exp-3**.

(Left) The interface of the INCA: Interactive Scheduling . (Right) Experiments with INCA suggest that retrieving personalized schedules helps users more as task difficulty increases. These experimental studies used a mixture of human and synthetic subjects.

5.2.3.4 Limitations

The challenges of adaptive interfaces may involve: conceptualizing user modeling as a task suitable for inductive learning, crafting representations that facilitate the learning process, gathering training data from users in a way that doesn't intrude on their experience, applying the learned user model effectively, ensuring the system can learn in real-time, and dealing with the necessity of learning from a limited number of training instances. These challenges are not only pertinent to adaptive interfaces but also intersect with broader applications of machine learning, while also introducing some unique issues. However, new sensor technology can bring

promises to adaptive interfaces. Adaptive interfaces rely on user traces to drive their modeling process, so they stand to benefit from developments like GPS and cell phone locators, robust software for speech recognition, accurate eye and head trackers, real-time video interpreters, wearable body sensors (GSR, heart rate), and portable brain-wave sensors. As those devices become more widespread, they will offer new sources of data and support new types of adaptive services. In addition, adaptive interfaces can be viewed as a form of cognitive simulation that automatically generates knowledge structures to learn user preferences. They are capable of making explicit predictions about future user behavior and explaining individual differences through the process of personalization. This perspective views adaptive interfaces as tools that not only serve functional purposes but also model the psychological aspects of user interaction. Two distinct approaches within cognitive simulation are related to adaptive interfaces: *process* models that incorporate fundamental architectural principles, and *content* models that operate at the knowledge level, focusing on behavior. We note that both of them have roles to play, but content models are more relevant to personalization and adaptive interfaces.

In conclusion, adaptive user interfaces represent a significant advancement in creating personalized and efficient interactions between humans and technology. By leveraging modern sensor technologies and cognitive simulation approaches, these interfaces can dynamically learn and adapt to individual user preferences, enhancing overall user experience and system effectiveness. The methodologies discussed, from conceptualizing user models to collecting and utilizing user feedback, form the foundation of this innovative approach. As we transition to the next section, we will explore practical applications and real-world implementations of these human-centered AI principles through detailed case studies, illustrating the tangible impact of adaptive interfaces in various domains.

5.2.4 Case Studies in Human-Centered AI

In this section, we examine practical examples that illustrate the application of human-centered principles in the development and deployment of AI systems. By examining these case studies, we aim to provide concrete insights into how AI technologies can be designed and implemented to better align with human values, enhance inclusivity, and address the specific needs of diverse user groups. The following case studies highlight different approaches and methodologies used to ensure that AI systems are not only effective but also considerate of the human experience.

5.2.4.1 LaMPost Case Study

In our exploration of human-centered AI design, it is crucial to examine how metrics can be improved to better capture the human experience and address the shortcomings of traditional evaluation methods. The LaMPost case study (Goodman et al. 2022) exemplifies this effort by focusing on the development of an AI assistant designed to aid individuals with dyslexia in writing emails. This case is particularly relevant to our discussion because it highlights the importance of human-centered principles in AI development, especially in creating tools that cater to specific cognitive differences and enhance user experience.

Dyslexia is a cognitive difference that affects approximately 15 percent of language users, with varying degrees of impact on speaking, spelling, and writing abilities. It is a spectrum disorder, meaning symptoms and severity differ among individuals. More importantly, dyslexia is not an intellectual disability; many individuals with dyslexia possess high intelligence. Given the significant number of people affected by dyslexia, it is essential to develop AI tools that support their unique needs and enhance their daily tasks.

The LaMPost project sought to answer the question, "How can LLMs be applied to enhance the writing workflows of adults with dyslexia?" To address this, researchers employed a participatory design approach, involving employees with dyslexia from their company (Google) in the study. This approach ensured that the development process was inclusive and responsive to the actual needs and preferences of the dyslexic community. By focusing on the real-world application of LLMs in aiding email writing for dyslexic individuals, LaMPost serves as a powerful example of how AI can be designed to better capture and enhance the human experience.

The figure below allows users to see suggestions for rewriting selected text, helping them identify main ideas, suggest possible changes, and rewrite their selections to improve clarity and expression.

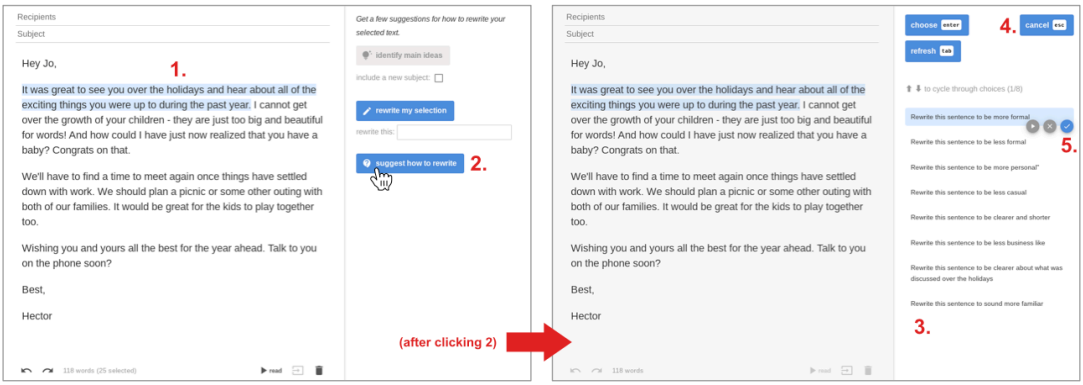


Figure 5.8
The Suggest Possible Changes feature from LaMPost.

The table below categorizes the challenges faced by users at different writing levels and the strategies they can use to overcome these challenges, illustrating the varied support needs addressed by LaMPost

Writing level

Examples of Challenges

Strategies

high

expressing ideas

"word faucet", ASR dictation

ordering ideas

post-it outlining

low

appropriate language

proofreading

paraphrasing

feedback

User challenged and strategies in LaMPost.

Next, they ran a focus group to get initial ideas from members of the dyslexic community. This focus group helped them figure out what to measure and added the second research question: "How do adults with dyslexia feel about LLM-assisted writing?" In other words, how does the LLM impact users' feelings of satisfaction, self-expression, self-efficacy, autonomy, and control?

From this focus group, they went and created a prototype to answer the desires of the group. They included three features in their prototype model. One feature was: *identifying main ideas*. They focused on this to support overall clarity and organization of high-level ideas of the user. Another feature was *suggest possible changes*. They focused on this because users wanted to identify high-level adjustments to improve their writing. The last feature they added was *rewrite my selections*. They added this because users wanted help expressing ideas with a desired phrasing tone or style. This feature generated a rewrite based on a command you gave it.

With the prototype, the researchers evaluated again with 19 participants with dyslexia from outside their organization. They did a three-part study, including a demonstration and background on the system (25 min). Then they did a writing exercise with two real tasks (emails) each user had to do in the real world (25 min). For example, one task might have been to write an email to the principal of their child's school to ask for a meeting. Then, the researchers did another follow-up interview for more qualitative data, e.g. to ask about specific choices users made when interacting with the model (25 min).

LaMPost's design prioritized autonomy by allowing users to choose the best option for their writing. One successful thing is that most users felt in control while writing. Users found that numerous options were helpful to filter through poor results. However, participants said the selection process was cognitively demanding and time-consuming. As we all know, features identified in LaMPost are all over the place, such as in Google Docs. Nonetheless, there remain many questions about the balance between automated writing and providing more control to the end users.

How could researchers hone in on this trade-off between **the ease of automated writing** and **providing control to end-users**?
You will need to design a study to approach this question.

– Identify your research question, hypotheses, and the methods that you will use. (Hint: use the HCI methods described in the previous section.)

– Scope the domain of your study appropriately—more broadly than dyslexia but not so broadly to be meaningless.

– What domains will you include? (E.g. students use ChatGPT for assignments, doctors use an LLM to write notes, etc.)

In this way, both the case study of LaMPost and its presaging of greater trends in LLM interfaces recapitulate the maxim of HCI: HCI is a cycle. You design a potential system, prototype it, get feedback from people, and iterate constantly. Next, we will explore two case studies that exemplify the application of human-centered principles in NLP. These case studies illustrate how LLMs can be adapted to foster social inclusivity and provide training in social skills.

5.2.4.2 Multi-Value and DaDa: Cross-Dialectal English NLP

English NLP systems are largely trained to perform well in Standard American English - the form of written English found in professional settings and elsewhere. Not only is Standard American English the most well-represented form of English in textual datasets but NLP engineers and researchers often filter dialectal and vernacular English examples from their datasets to improve performance on SAE benchmarks. As a result, NLP systems are generally less performant when processing dialectal inputs than SAE inputs. This performance gap is observable over various benchmarks and tasks, like the SPIDER benchmark. (Chang et al. 2023)

Evaluation		Input Dialect						
Model	Metric	SAE	AppE	ChcE	CollSgE	IndE	UAAVE	Avg.
BART-base	Exact Match ACC	49.3	45.2 (-8.3%)⁻	48.5 (-1.6%)⁻	41.9 (-15.0%)⁻	40.5 (-17.8%)⁻	45.0 (-8.7%)⁻	45.1 (-8.5%)
	Execution ACC	51.0	47.3 (-7.3%)⁻	50.3 (-1.4%)	44.1 (-13.5%)⁻	42.3 (-17.1%)⁻	46.1 (-9.6%)⁻	46.9 (-8.0%)
BART-large	Exact Match ACC	67.9	63.6 (-6.3%)⁻	65.5 (-3.5%)⁻	60.3 (-11.2%)⁻	61.2 (-9.9%)⁻	62.3 (-8.2%)⁻	63.5 (-6.5%)
	Execution ACC	70.5	65.2 (-7.5%)⁻	68.2 (-3.3%)⁻	63.0 (-10.6%)⁻	62.8 (-10.9%)⁻	64.5 (-8.5%)⁻	65.4 (-7.2%)
T5-base	Exact Match ACC	58.7	54.3 (-7.5%)⁻	57.4 (-2.2%)⁻	50.0 (-14.8%)⁻	49.1 (-16.4%)⁻	53.1 (-9.5%)⁻	53.8 (-8.3%)
	Execution ACC	59.8	56.0 (-6.4%)⁻	58.5 (-2.2%)⁻	51.6 (-13.7%)⁻	51.3 (-14.2%)⁻	54.6 (-8.7%)⁻	55.3 (-7.5%)
T5-3b	Exact Match ACC	71.7	65.3 (-8.9%)⁻	69.7 (-2.8%)⁻	60.7 (-15.3%)⁻	62.9 (-12.3%)⁻	68.5 (-4.5%)⁻	66.5 (-7.3%)
	Execution ACC	75.6	69.3 (-8.3%)⁻	73.4 (-2.9%)⁻	64.9 (-14.2%)⁻	66.5 (-12.0%)⁻	66.9 (-11.5%)⁻	69.4 (-8.2%)

Figure 5.9
Stress test reveals worse performance on the SPIDER benchmark with synthetic dialectical examples than with SAE.

As natural language systems become more pervasive, this performance gap increasingly represents a real allocational harm against dialectal English speakers — these speakers are excluded from using helpful systems and assistants. Multi-Value is a framework for evaluating foundation language models on dialectic input, and DADA is a framework for adapting LLMs to improve performance on dialectic input.

Synthetic Dialectal Data

Ziems et al. (2023) create synthetic dialectal data for several English dialects (Appalachian English, Chicano English, Indian English, Colloquial Singapore English, and Urban African American English).(Ziems et al. 2023) They created synthetic data based on transforming SAE examples to have direct evaluation comparisons. These synthetic examples were created by leveraging known linguistic features of the dialects, such as negative concord in UAAVE. Figure 5.10 maps out the presence of various linguistic features.

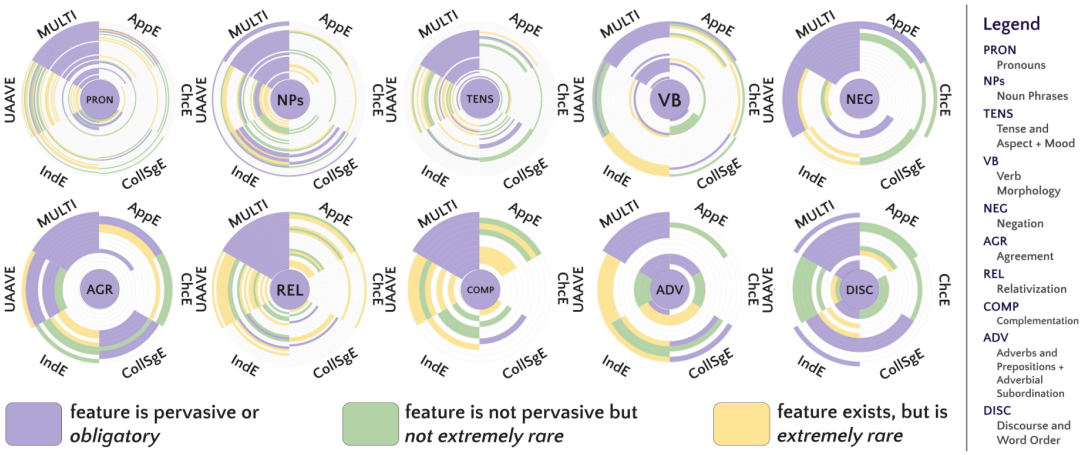


Figure 5.10
A comparative distribution of features in five dialects.

This synthetic data, while somewhat limited in the variety of samples. can produce and create realistic examples for benchmarking LM performance. Figure 5.11 demonstrates creating a synthetic dialectic example using the 'give passive' linguistic feature, illustrating the transformation process from SAE to a vernacular form.

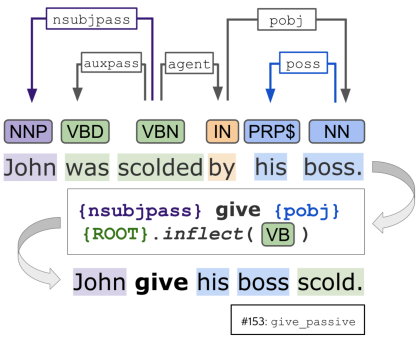


Figure 5.11
Execution of a sample transform using a documented linguistic feature.

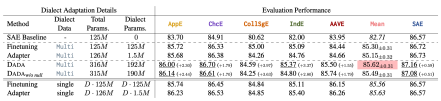
Feature Level Adapters One approach to the LLM adaption task would be to train an adapter for each dialect using a parameter-efficient fine-tuning method like low-rank adapters. (Hu et al. 2021) While adapters can certainly bridge the gap between SAE LMs and dialect inputs, this approach suffers from a couple of weaknesses, namely:

– Individually trained adapters do not leverage similarities between low-resource dialects.

Transfer learning is often helpful for training low-resource languages and dialects.

– The model needs to know which adapter to use at inference time. This presupposes that we can accurately classify the dialect — sometimes based on as little as one utterance. This classification is not always possible — a more general approach is needed.

Therefore, Liu et al. (2023) propose a novel solution — DADA: Dialect Adaption via Dynamic Aggregation of Linguistic Rules. (Liu, Held, and Yang 2023) DADA trains adapters on the linguistic feature level rather than the dialect level. The model can use multiple linguistic feature adapters via an additional fusion layer. They can therefore train using multi-dialectical data and cover linguistic variation via a comprehensive set of roughly 200 adapters. DADA saw an improvement in performance over single-dialect adapters for most dialects, as shown in Figure 5.12.



Dialect Adaptation Details				Evaluation Performance						
Method	Dialect Data	Total Params.	Dialect Params.	AppE	ChcE	CollSgE	IndE	AAVE	Mean	SAE
SAE Baseline	–	125M	0	83.70	84.91	80.62	82.00	83.95	82.71	86.57
Finetuning	Multi	125M	125M	85.72	86.33	85.00	85.09	84.44	85.30 ±0.35	86.72
Adapter	Multi	126M	1.5M	85.68	86.38	84.26	84.76	84.66	85.15 ±0.32	86.73
DADA	Multi	316M	192M	86.00 (+0.28)	86.70 (+1.70)	84.59 (+3.97)	85.27 (+2.27)	85.50 (+1.28)	85.81 ±0.31	87.18 (+0.58)
DADA$_{\text{with-null}}$	Multi	315M	190M	86.14 (+0.42)	86.61 (+1.70)	84.25 (+3.65)	84.89 (+2.06)	85.74 (+1.78)	85.49 ±0.31	87.08 (+0.48)
Finetuning	single	$D \cdot 125M$	$D \cdot 125M$	85.74	86.45	84.84	85.11	86.15	85.56	86.57
Adapter	single	$D \cdot 126M$	$D \cdot 1.5M$	86.25	86.53	84.85	85.40	86.26	85.63	86.57

Figure 5.12
Execution of a sample transform using a documented linguistic feature.

The Multi-Value and DADA case study underscores the importance of designing NLP systems that are inclusive and representative of diverse language users. By addressing the performance gaps in handling dialectal inputs, this case study highlights the necessity of incorporating diverse linguistic data and creating adaptable systems. This approach enhances AI functionality and accessibility, ensuring it respects and reflects linguistic diversity. Ultimately, the study reinforces human-centered design principles, demonstrating how AI can be tailored to better serve and empower all users. Moving forward, we will explore how LLMs can be utilized for social skill training, showcasing their potential to improve human interactions.

5.2.4.3 Social Skill Training via LLMs

The emergence of Large Language Models (LLMs) marks a significant milestone in the field of social skills training. This case study explores the potential of LLMs to augment social skill development across diverse scenarios. More specifically, we discuss a dual-framework approach, where two distinct LLMs operate in tandem as a Partner and a Mentor, guiding human learners in their journey towards improved social interaction. In this framework, we have two agents which are

– **AI Partner**: LLM-empowered agents that users can engage with across various topics. This interactive model facilitates practical, conversation-based learning, enabling users to experiment with different communication styles and techniques or practice and develop specific skills in real-world scenarios in a safe, AI-mediated environment.

– **AI Mentor**: An LLM-empowered entity designed to provide constructive, personalized feedback based on the interaction of users and the AI Partner. This mentor analyzes conversation dynamics, identifies areas for improvement, offers tailored advice, and guides users toward effective social strategies and improved interaction skills.

For example, in conflict resolution, individuals learning to handle difficult conversations can use the AI Partner to simulate interactions with a digitalized partner. As a Conflict Resolution Expert, the AI Mentor helps analyze these interactions, offering strategies to navigate conflicts effectively.

In the educational sector, K-12 teachers aiming to incorporate more growth-mindset language into their teaching can practice with a digitalized student. An experienced teacher or mentor, represented by the AI Mentor, provides insights on effective communication and teaching methods. For negotiation training, students preparing to negotiate their first job offers can engage in simulated negotiations with a digitalized HR representative through the AI Partner. As a Negotiation Expert, the AI Mentor then offers guidance on negotiation tactics, helping students effectively articulate their values and negotiate job terms. Lastly, in therapy training, novice therapists can interact with a digitalized patient via the AI Partner to practice therapy sessions. The AI Mentor, functioning as a Therapy Coach, then reviews these sessions, providing feedback and suggestions on enhancing therapeutic techniques and patient engagement.

CARE: Therapy Skill Training Hsu et al. (2023) introduced CARE (Hsu et al. 2023), a framework designed for therapy skill training. This framework leverages a simulated environment, enabling counselors to practice their skills without the risk of harming real individuals. An integral component of CARE is the AI Mentor, which offers invaluable feedback and guidance during the training process. See Figure 5.13 for the overview of the framework.

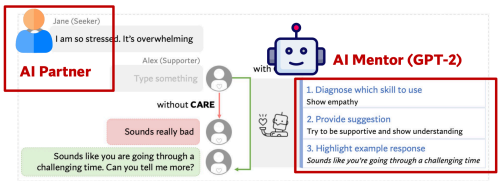


Figure 5.13
CARE Framework

CARE's primary function is for novice therapists and counselors to assess and determine the most effective counseling strategies tailored to specific contexts. It provides counselors with customized example responses, which they can adopt, adapt, or disregard when interacting with a simulated support seeker. This approach is deeply rooted in the principles of Motivational Interviewing and utilizes a rich dataset of counseling conversations combined with LLMs. The effectiveness of CARE has been established through rigorous quantitative evaluations and qualitative user studies, which included simulated chats and semi-structured interviews. Notably, CARE has shown significant benefits in aiding novice counselors. From the assessment, counselors chose to use CARE 93% of the time, directly used a CARE response without editing 60% of the time, and sent more extended responses with CARE. Qualitatively, counselors noted several advantages of CARE, such as its ability to refresh memory on various strategies,

inspire innovative responses, boost confidence, and save time during consultations. However, there were some drawbacks, including potential disruptions in the thought process, perceived limitations in response options, the requirement for decision-making, and the time needed to review suggestions. Overall, the framework is particularly beneficial for therapists new to the field, offering them a supportive and educational tool to enhance their counseling skills effectively.

5.3 Practice Exercises

References

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mane. 2016. "Concrete Problems in AI Safety." *arXiv Preprint arXiv:1606.06565*.

Angwin, Julia, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. "Machine Bias." *ProPublica*.

Arcas, Blaise Aguera y. 2022. "Can Machines Learn How to Behave?" *Medium*. https://medium.com/@blaisea/can-machines-learn-how-to-behave-42a02a57fadb.

Aristotle. 350 B.C.E. *Nicomachean Ethics*. translated by W.D. Ross.

Bai, Yuntao, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, and Cameron McKinnon. 2022. "Constitutional Ai: Harmlessness from Ai Feedback." *arXiv Preprint arXiv:2212.08073*.

Barocas, Solon, Moritz Hardt, and Arvind Narayanan. 2019. *Fairness and Machine Learning*. fairmlbook.org.

Bernstein, Michael S., Greg Little, Robert C. Miller, Bjorn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. 2010. "Soylent: A Word Processor with a Crowd Inside." In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*. ACM.

Binns, Reuben. 2018. "Fairness in Machine Learning: Lessons from Political Philosophy." In *Proceedings of the 2018 Conference on Fairness, Accountability, and Transparency*, 149–59.

Bostrom, Nick. 2014. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Brown, Daniel S, and Scott Niekum. 2019. "Machine Teaching for Inverse Reinforcement Learning: Algorithms and Applications." In *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7749–58.

Brown, Daniel S, Jordan Schneider, Anca Dragan, and Scott Niekum. 2021. "Value Alignment Verification." In *International Conference on Machine Learning*, 1105–15. PMLR.

Centers for Disease Control and Prevention. 2023. "The u.s. Public Health Service Untreated Syphilis Study at Tuskegee." https://www.cdc.gov/tuskegee/index.html.

Chang, Shuaichen, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, et al. 2023. "Dr.spider: A Diagnostic Evaluation Benchmark Towards Text-to-SQL Robustness." https://arxiv.org/abs/2301.08881.

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, et al. 2022. "PaLM: Scaling Language Modeling with Pathways." *arXiv:2204.02311 [Cs]*, April. http://arxiv.org/abs/2204.02311.

Christiano, Paul. 2018. "Clarifying 'AI Alignment'." https://ai-alignment.com/clarifying-ai-alignment-cec47cd69dd6.

Christiano, Paul F, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. "Deep Reinforcement Learning from Human Preferences." *Advances in Neural Information Processing Systems* 30.

Clark, Jack, and Dario Amodei. 2016. "Faulty Reward Functions in the Wild." *OpenAI Blog*.

Dworkin, Gerald. 1988. *The Theory and Practice of Autonomy*. Cambridge University Press.

Everitt, Tom, and Marcus Hutter. 2018. "The Alignment Problem for Artificial Intelligence." In *Advances in Neural Information Processing Systems*, 1–8.

Floridi, Luciano. 2011. *The Ethics of Information*. Oxford University Press.

Frankena, William K. 1973. *Ethics*. Prentice Hall.

Friedman, Batya, Peter H. Kahn, and Alan Borning. 2008. "Value Sensitive Design and Information Systems." In *The Handbook of Information and Computer Ethics*. John Wiley & Sons.

Gervasio, Melinda T, Wayne Iba, and Pat Langley. 1999. "Learning User Evaluation Functions for Adaptive Scheduling Assistance." In *ICML*, 152–61. Citeseer.

Goodall, Noah J. 2014. "Machine Ethics and Automated Vehicles." In *Road Vehicle Automation*, 93–102. Springer.

Goodman, Steven, Erin Buehler, Patrick Clary, Andy Coenen, Aaron Michael Donsbach, Tiffanie Horne, Michal Lahav, et al. 2022. "LaMPost: Evaluation of an AI-Assisted Writing Email Editor Prototype for Adults with Dyslexia."

Hadfield-Menell, Dylan, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. "Cooperative Inverse Reinforcement Learning." *Advances in Neural Information Processing Systems* 29.

Hardt, Moritz, and Benjamin Recht. 2021. "Patterns, Predictions, and Actions: A Story about Machine Learning." *arXiv Preprint arXiv:2102.05242*.

Hasselt, Hado van, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. 2018. "Deep Reinforcement Learning and the Deadly Triad."

Hejna, Joey, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W. Bradley Knox, and Dorsa Sadigh. 2023. "Contrastive Preference Learning: Learning from Human Feedback Without RL." https://arxiv.org/abs/2310.13639.

Hendrycks, Dan, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2020. "Aligning Ai with Shared Human Values." *arXiv Preprint arXiv:2008.02275*.

Hendrycks, Dan, Mantas Mazeika, Andy Zou, Sahil Patel, Christine Zhu, Jesus Navarro, Dawn Song, Bo Li, and Jacob Steinhardt. 2021. "What Would Jiminy Cricket Do? Towards Agents That Behave Morally." *arXiv:2110.13136 [Cs]*. http://arxiv.org/abs/2110.13136.

Hovy, Dirk, and Diyi Yang. 2021. "The Importance of Modeling Social Factors of Language: Theory and Practice." In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, edited by Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, 588–602. Online: Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.naacl-main.49.

Hsu, Shang-Ling, Raj Sanjay Shah, Prathik Senthil, Zahra Ashktorab, Casey Dugan, Werner Geyer, and Diyi Yang. 2023. "Helping the Helper: Supporting Peer Counselors via AI-Empowered Practice and Feedback." https://arxiv.org/abs/2305.08982.

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." https://arxiv.org/abs/2106.09685.

Huang, Sandy H, Kush Bhatia, Pieter Abbeel, and Anca D Dragan. 2018. "Establishing Appropriate Trust via Critical States." In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3929–36. IEEE.

Hubinger, Evan, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. 2019. "An Introduction to Inner Alignment." *arXiv Preprint arXiv:1906.01820*.

Jiang, Fei, Yong Jiang, Hang Zhi, Yuan Dong, Hui Li, Shugang Ma, and Yongan Wang. 2017. "Artificial Intelligence in Healthcare: Past, Present and Future." *Stroke and Vascular Neurology* 2 (4): 230–43.

Jiang, Liwei, Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, Maxwell Forbes, Jon Borchardt, Jenny Liang, Oren Etzioni, Maarten Sap, and Yejin Choi. 2021. "Delphi: Towards Machine Ethics and Norms." *arXiv:2110.07574 [Cs]*, October. http://arxiv.org/abs/2110.07574.

Johnson, Robert, and Adam Cureton. 2022. "Kant's Moral Philosophy." In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta and Uri Nodelman, Fall 2022. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/fall2022/entries/kant-moral/.

Krakovna, Victoria et al. 2020. "Specification Gaming Examples in AI." *DeepMind Safety Research*.

Langley, Pat, Cynthia Thompson, Renee Elio, and Afsaneh Haddadi. 1999. "An Adaptive Conversational Interface for Destination Advice." In *International Workshop on Cooperative Information Agents*, 347–64. Springer.

Leike, Jan, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. 2018. "Scalable Agent Alignment via Reward Modeling: A Research Direction." https://arxiv.org/abs/1811.07871.

Liang, Percy, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, et al. 2023. "Holistic Evaluation of Language Models." arXiv. https://doi.org/10.48550/arXiv.2211.09110.

Liu, Yanchen, William Held, and Diyi Yang. 2023. "DADA: Dialect Adaptation via Dynamic Aggregation of Linguistic Rules." https://arxiv.org/abs/2305.13406.

Mazeika, Mantas, Eric Tang, Andy Zou, Steven Basart, Jun Shern Chan, Dawn Song, David Forsyth, Jacob Steinhardt, and Dan Hendrycks. 2022. "How Would The Viewer Feel? Estimating Wellbeing From Video Scenarios." *arXiv Preprint arXiv:2210.10039*.

Mehrabi, Ninareh, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. "A Survey on Bias and Fairness in Machine Learning." *ACM Computing Surveys (CSUR)* 54 (6): 1–35.

Mill, John Stuart. 1863. *Utilitarianism*. Parker, Son,; Bourn.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker. 2018. "Emotion in Reinforcement Learning Agents and Robots: A Survey." *Machine Learning* 107:443–80.

Morris, Meredith Ringel. 2019. "Human-in-the-Loop Computing: Reimagining Human-Computer Interaction in the Age of AI." In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM.

Muller, Michael J. 2003. "Participatory Design: The Third Space in HCI." In *The Human-Computer Interaction Handbook*. CRC Press.

Ngo, Richard, Lawrence Chan, and Sören Mindermann. 2023. "The Alignment Problem from a Deep Learning Perspective." https://arxiv.org/abs/2209.00626.

Noble, Safiya Umoja. 2018. *Algorithms of Oppression: How Search Engines Reinforce Racism*. NYU Press.

Nussbaum, Martha C, and Amartya Sen. 1993. *The Quality of Life*. Oxford University Press.

O'Neil, Cathy. 2016. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Publishing Group.

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, et al. 2022. "Training Language Models to Follow Instructions with Human Feedback."

Project, LaMPort. 2017. "LaMPost: Leveraging Crowdsourcing for Natural Language Processing." In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. ACL.

Quine, Willard Van Orman. 1960. *Word and Object*. MIT Press. https://openlibrary.org/works/OL2910272W?edition=ia%3Awordobject00quin.

Rawls, John. 1971. *A Theory of Justice*. Harvard University Press.

Rogers, Seth, Claude-Nicolas Fiechter, and Pat Langley. 1999. "An Adaptive Interactive Agent for Route Advice." In *Proceedings of the Third Annual Conference on Autonomous Agents*, 198–205.

Russell, Stuart. 2019. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking.

Sadigh, Dorsa, Anca Dragan, Shankar Sastry, and Sanjit Seshia. 2017. "Active Preference-Based Learning of Reward Functions."

Schwartz, Shalom H. 1992. "Universals in the Content and Structure of Values: Theoretical Advances and Empirical Tests in 20 Countries." *Advances in Experimental Social Psychology* 25:1–65.

Shah, Rohin, Vikrant Varma, Ramana Kumar, Mary Phuong, Victoria Krakovna, Jonathan Uesato, and Zac Kenton. 2022. "Goal Misgeneralization: Why Correct Specifications Aren't Enough for Correct Goals." https://arxiv.org/abs/2210.01790.

Stiennon, Nisan, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. "Learning to Summarize from Human Feedback."

Talat, Zeerak, Hagen Blix, Josef Valvoda, Maya Indira Ganesh, Ryan Cotterell, and Adina Williams. 2022. "On the Machine Learning of Ethical Judgments from Natural Language." In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

The National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research. 1979. "The Belmont Report: Ethical Principles and Guidelines for the Protection of Human Subjects of Research." https://www.hhs.gov/ohrp/regulations-and-policy/belmont-report/index.html.

Tomasello, Michael. 2019. *Becoming Human: A Theory of Ontogeny*. Cambridge, MA: Belknap Press.

Vamplew, Peter, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. 2018. "Human-Aligned Artificial Intelligence Is a Multiobjective Problem." *Ethics and Information Technology* 20 (1): 27–40. https://doi.org/10.1007/s10676-017-9440-6.

Vamplew, Peter, Benjamin J. Smith, Johan Källström, Gabriel Ramos, Roxana Rădulescu, Diederik M. Roijers, Conor F. Hayes, et al. 2022. "Scalar Reward Is Not Enough: A Response to Silver, Singh, Precup and Sutton (2021)." *Autonomous Agents and Multi-Agent Systems* 36 (2): 41. https://doi.org/10.1007/s10458-022-09575-5.

Weidinger, Laura, Madeline G. Reinecke, and Julia Haas. 2022. "Artificial Moral Cognition: Learning from Developmental Psychology." Preprint. PsyArXiv. https://doi.org/10.31234/osf.io/tnf4e.

Wikipedia contributors. 2023. "AI Alignment — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=AI_alignment&oldid=1185176830.

Xiong, Wayne, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. "Achieving Human Parity in Conversational Speech Recognition." *arXiv Preprint arXiv:1610.05256*.

Ziebart, Brian D. 2010. "Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy." PhD Thesis, Pittsburgh, PA: Carnegie Mellon University.

Ziems, Caleb, William Held, Jingfeng Yang, Jwala Dhamala, Rahul Gupta, and Diyi Yang. 2023. "Multi-VALUE: A Framework for Cross-Dialectal English NLP." https://arxiv.org/abs/2212.08011.

Acknowledgments

Acknowledgments

This book was compiled during CS329H: Machine Learning from Human Preferences at Stanford University in Fall 2023 and Fall 2024. We thank Rehaan Ahmad, Ahmed Ahmed, Jirayu Burapacheep, Michael Byun, Akash Chaurasia, Andrew Conkey, Tanvi Deshpande, Eric Han, Laya Iyer, Adarsh Jeewajee, Shreyas Kar, Arjun Karanam, Jared Moore, Aashiq Muhamed, Bidipta Sarkar, William Shabecoff, Stephan Sharkov, Max Sobol Mark, Kushal Thaman, Joe Vincent, Yibo Zhang, Duc Nguyen (VNU-HCM University of Technology), Grace Sodunke (University of Oxford), and Ky Nguyen (DePauw University) for their help in compiling this book. We appreciate the time of our guest speakers, including Pat Langley (Institute for the Study of Learning and Expertise), Meredith Ringel Morris (Google DeepMind), Vasilis Syrgkanis (Stanford), Jason Hartline (Northwestern), Dorsa Sadigh (Stanford), Diyi Yang (Stanford), and Nathan Lambert (AI2).

Index