Bar Ilan University

Agent Behavior Modeling by Learning from Action Logs

Author:
Elad Say Mintzer

Supervisor:
Prof. Gal A. Kaminka

Computer Science Department
Bar Ilan University
Ramat Gan, Israel 52900

May 7, 2023

Abstract

The development of AI agents is a challenging task that is also time-consuming. ML techniques try to tackle this problem, allowing by interacting with the agent to its environment. However in our test case, there is no ready access to the environment, and the only data available is recorded data - logs. The data may not contain information about how the agent perceived its environment before taking action. Moreover, the data will often be missing information on the agent's internal processes because the logs record only globally observable information. This thesis tackles two challenges when modeling the behavior from logs recorded in continuous environments with continuous actions. In the first part of the thesis, we focus on the semi-automated learning of continuous action parameters. The method relies on guidance from a human domain expert but uses machine learning algorithms to carry out the actual learning. In the second part of the thesis, we focus on mining sequences of complex actions that appear in the logs. We build on earlier work in hierarchical sequence mining to introduce a novel method for mining action sequences where actions are complex and have discretized parameters. We hope the combination of the techniques from the two parts will lead towards the capacity for building fuller agent behavior models from logs of actions and environment settings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificially-intelligent agents are increasingly being developed for sophisticated applications such as computer games [44, 54, 66], simulations for training, planning, and entertainment [16, 21, 38, 77, 79], alongside autonomous service robots [84], personal assistants, and many others. Such agents carry out complex decision-making processes, often mixing both reactive and proactive goal-oriented reasoning. They are capable of incorporating dynamically-changing information from the environment in which the agents are situated while taking into consideration other agents with which they can collaborate or compete.

The development of such agents is both challenging and time-consuming. An entire area of research, called *Agent-Oriented Software Engineering*, is devoted to the investigation of engineering methods, specialized programming languages, and control architectures to simplify the development process [10,11,17,18,55,69,73,74,83]. This challenge is further exacerbated when the goal is to develop agents capable of human-level decision-making, such as that required in defense simulations [77] or crowd simulations [21, 38, 79].

One alternative approach developing the decision-making process relies on *machine learning* techniques that generate the decision-making procedure by allowing the agent to interact with its environment. Indeed, much of *reinforcement learning*, a popular area of machine learning, is devoted to allowing a learning agent to interact with its environment, all the while learning to improve its responses. In recent years,

the use of reinforcement learning has led to impressive successes, including learning to fly a helicopter [1,41] or playing Go at the champion level [70].

However, it is not always feasible to allow an agent to interact freely with the environment, for example, because of the time and resources such interactions require or due to the lack of ready access to the environment. When we wish to model human decision-making, it is generally not possible to build a model of an existing agent without having access to its internal environment.

In these cases, the data available to the machine learning model may consist solely of recordings, such as *logs* of the agent's actions and responses to its environment. The data may not contain information about how the agent has perceived its environment before taking action. Moreover, the data will often be missing information on the agent's internal processes because the logs record only globally observable information.

We focus on *behavior modeling* that learns the decision-making procedure π of an agent from logs documenting its observable interactions with an environment. The behavior modeling system is given multiple logs as input, each consisting of multiple pairs $\langle \overrightarrow{E_t}, \overrightarrow{A_t} \rangle$, indexed by time t. Here, $\overrightarrow{E_t}$ is a collection of environment features that describe the environment state at time t, while $\overrightarrow{A_t}$ refers to the agent's actions at the same time t[1]. The task of the system is to induce $\pi^i : \mathcal{E} \rightarrow \mathcal{A}$, where $\mathcal{E}$ denotes the space of all possible environment states reached through all possible trajectories over time, and $\mathcal{A}$ denotes all possible actions.

Behavior modeling includes or overlaps with *grammar induction* from text [45,52, 71,72], *data-* and *sequence- mining* from logs [13,37,46,60], *behavior cloning* [7,67], *programming by demonstration* [26,75], and other machine learning techniques. However, existing techniques for behavior modeling generally do not support learning from continuous data, where the environment feature values are continuous, the actions are continuous, or both. Also, they typically do not account for complex values (e.g., a tuple $\langle x_o, y_o, \theta_o \rangle$ as a specific value for a location feature) or an action with multiple continuous parameters. See Chapter 2 for a detailed discussion.

[1]The null action (no action) is considered a possible value.

This thesis tackles two challenges when modeling the behavior from logs recorded in continuous environments with continuous actions. In the first part of the thesis, we focus on the *semi-automated* learning of continuous action parameters. The method relies on guidance from a human domain expert but uses machine learning algorithms to carry out the actual learning. In the second part of the thesis, we focus on mining sequences of complex actions that appear in the logs. We introduce the two parts below.

1.1 Part I: Expert-Guided Learning of Actions from Logs

In Part I of the thesis, we focus on learning continuous complex actions with one or more parameters that can take on continuous values. We first establish a baseline for such learning, using a variety of state-of-the-art classification and regression machine learning algorithms. After demonstrating the difficulty of the task, we take a series of steps in which guidance from a domain expert is used to incrementally augment the learning task, resulting in significant improvements to the learning success.

This systematic semi-automated approach to learning the agent's decision-making procedure π uses the domain expert's knowledge to isolate important structural elements in π. Specifically, the process identifies limits on the agent's perception, its internal interpretations of the data, and its potential task-dependent goals.

The first step, discussed in Chapter 3, involves pre-processing information on the environment's state $\overrightarrow{E_t}$ to generate features S_t that describe the environment features from the point of view of the agent. This transformation requires knowledge of the sensory limitations of the agent, which the domain expert needs to provide. For example, in the domain of simulated soccer, the transformation is from features describing the location of all agents (environment state), to features describing the location of agents that a specific agent a is able to sense. S_t, for instance, will include the location of only those agents within range and in front of a's simulated cameras.

In Chapter 4, we take additional expert-guided steps to improve the learning out-

comes. These are key to enriching the description of the agent's state by using domain knowledge regarding its beliefs (i.e., presumptions). In Chapter 5 we enriched the description of the agent's state by adding the agent's goals using domain knowledge. First, we describe a technique in which a visualization of the parameter values is used to recognize clusters of action parameter values. These clusters help us distinguish between different decision-making contexts of the agents. We also describe a technique by which, given a set of possible agent goals, the logs are searched forward in time to determine whether an action at time t_i served to achieve a goal at some time t_j in the future, when $j > i$. This allows us to recognize actions in the service of different goals, and further improve learning outcomes.

Finally, Chapter 6 presents experiments that evaluate the techniques introduced in the previous chapters, and compares their results to the baseline. These experiments demonstrate that our approach offers significant improvements over the baseline outcomes.

1.2 Part II: Mining Complex Action Sequences from Logs

In Part II of the thesis, we focus on different behavior modeling challenges arising from the logs of agent interactions in continuous settings. Specifically, we look at *data mining* to identify frequently occurring sequences of actions, without guidance. The ability to identify the frequent repetition of specific sequences (i.e., actions in a specific order) is an important part of behavior modeling from logs, and can be used towards various analysis goals [32,34,37].

There exist known techniques for mining sequences, an area of research within data-mining called *sequential pattern mining*. These assume a discrete and finite alphabet from which patterns are formed. In our case, every action, is denoted by a single, distinct, symbol of the alphabet, including its parameters which is continuous. This raises a difficult challenge for existing sequence mining techniques.

Actions in continuous spaces have infinite instances. A common method for ad-

dressing this is to quantize (i.e., discretize) the action parameter space. For example, the action for turning accepts a parameter that determines the degrees by which the agent turns. Assuming a half-degree finite quantization, there would be 720 different symbols just to describe the turning action in all of its instances.

However, this prohibits current mining algorithms from grouping together sequences that contain different instances of the same action. For example, a sequence containing a turn by 1 degree (denoted here, for simplicity, as $(..., turn - 1 - degree, ...)$) is just as different from the same sequence with a 1.5 degree turn $(..., turn - 1.5 - degrees, ...)$ as it is from an otherwise identical sequence with a 180-degree turn $(..., turn - 180 - degrees, ...)$.

As each such turn becomes a single distinct symbol, we transform a metric variable into a categorical one, and lose semantic information about the distance between the different symbols. Now, each combination of parameter values becomes a separate symbol for the mining algorithm. This problem is exacerbated when the number of parameters or dimensions in which the action operates increases.

We build on earlier work in hierarchical sequence mining to introduce a novel method for mining action sequences where actions are complex and have discretized parameters. In Chapter 7, we explain how to build a hierarchy of actions, based on multiple discretization resolutions. In Chapter 8 we describe our different measures for the quality of a hierarchical sequence mining model. In Chapter 9 we defined a sequence mining problem that seeks a given k number of sequences that maximize the model accuracy. We propose a solution to the problem, by reducing it to a known algorithmic problem, which can be approximately solved.

1.3 Thesis Overview

This dissertation comprises ten chapters (see Figure 1-1 on page 6), with the core chapters of the thesis organized into two main parts as described above. This chapter constitutes the introduction to this thesis, while the next chapter surveys the related work. Chapters 3 to 6 contain the first part. Chapters 7 through 9

constitute the second part. Chapter 10 presents the final remarks to this thesis, providing a summary of the research, conclusions, and possible directions for future research. It also discuss the combination of the techniques discussed in Parts I and II.

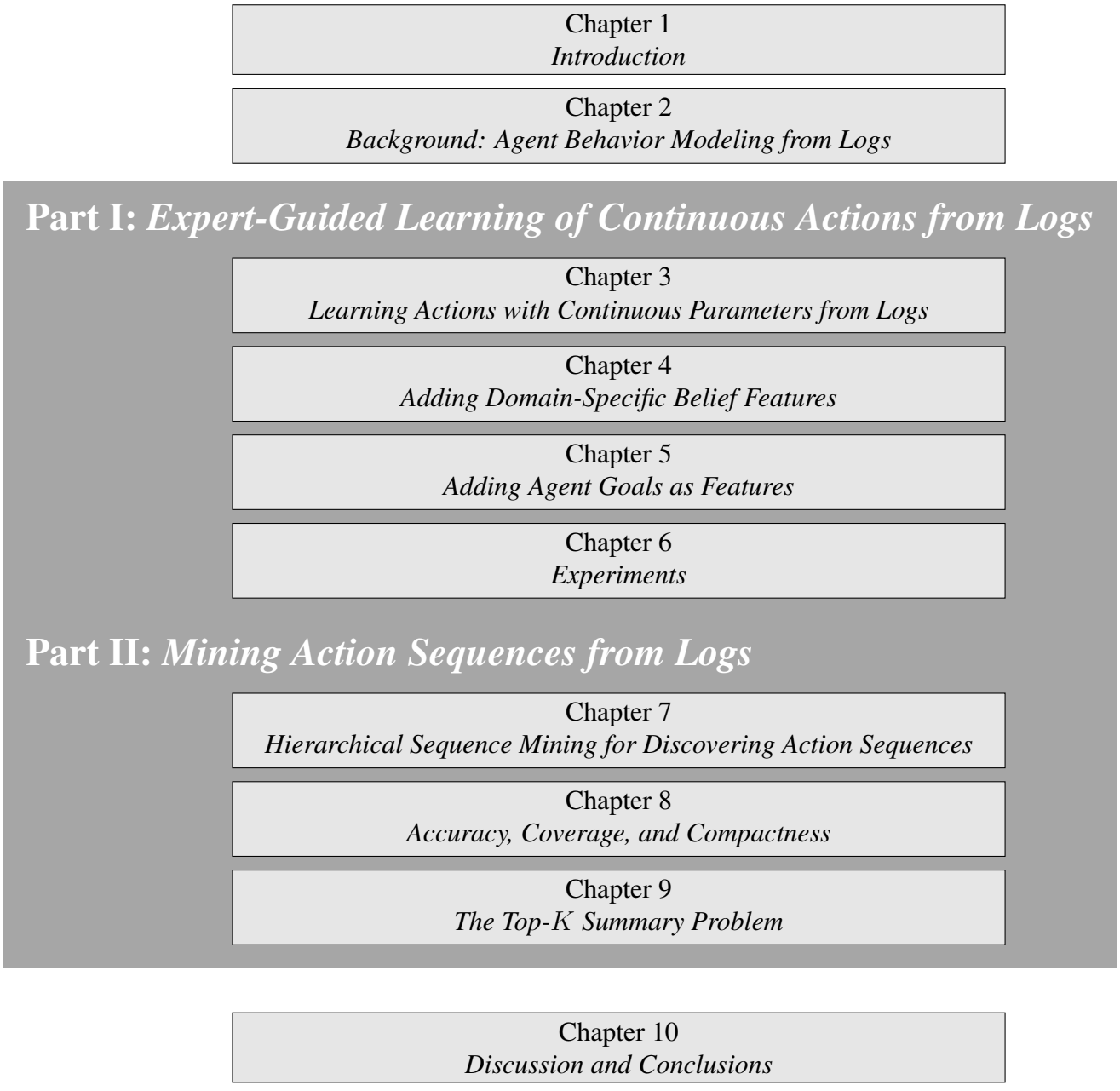


Figure 1-1: Thesis Structure

Chapter 2

Background: Agent Behavior Modeling from Logs

The development of artificially intelligent agents is a challenging task [10,11,17,18,55, 69,73,74,83]. So much so, that there are several conferences and scientific communities devoted to the challenges inherent to this task, including: Agent-Oriented Software Engineering (AOSE)[1], the Game-On[2] series of conferences on AI in Computer Games, the International Conference on Social Computing, Behavioral-Cultural Modeling, Prediction, and Behavior Representation in Modeling and Simulation (SBP-BRiMS[3]), the Intelligent Virtual Agents (IVA[4]) conference, the Multi-Agent-Based Simulation workshop (MABS[5]), and more.

As an alternative to the labor-intensive development of sophisticated agents, an agent can learn and improve from its own interactions with the target environment, without human supervision. For example, using *evolutionary computation* methods or *reinforcement learning*, the agent can improve itself until it is able to carry out its task successfully. In recent years, the use of reinforcement learning has led to impressive successes such as learning to fly a helicopter [1, 41] or playing Go at the

[1] See `http://winf.in.tu-clausthal.de/events/aose12/` for the AOSE conference, and `https://www.inderscience.com/jhome.php?jcode=ijaose` for the affiliated journal.

[2] `https://www.eurosis.org/cms/?q=taxonomy/term/257`

[3] `http://sbp-brims.org/`

[4] `https://dl.acm.org/conference/iva`

[5] `https://easychair.org/cfp/MABS2021`

champion level [70]. The learning process itself can become extremely expensive in terms of the amount of data required and the computational resources needed.

However, the use of machine learning to automate agent development misses a fundamental point. Many applications are required to develop agents that display human-level decision-making, including human biases and mistakes. For example, these include agents for air combat training simulations [77], emergency crowd evacuations [79], or pedestrian modeling for urban planning [21,38]. Likewise, when developing agents for computer games, the developer may not wish to have the agents excel too much at the game, since their objective is to match the human adversary for her entertainment, not to win the game. In all of these applications and many more, the agents must act as humanly as possible—including making human-like mistakes—but not beyond.

Our overall goal in this work is to build training agents that behave similar to humans. The first phase towards this goal focuses on learning a model of the agent—rather than a complete agent—from logs. The work we report on here is therefore related to machine learning techniques that can be or are being used with the observations of expert agent interactions, as the basis for learning.

A variety of methods, which we generally refer to as *imitation learning*, have been proposed to speed up and focus the learning towards specific behaviors. This is done by providing examples that offer traces of human expert-level decision-making or observations of expert-level interactions to a machine learning process. The machine learning then uses the examples to guide and focus its interactions with the task. Research into *programming by demonstration* or *programming by example*, *behavior cloning*, and *inverse reinforcement learning* are all areas where observations about an agent already interacting with the target environment are used to automatically build an agent capable of similar or even improved performance on the same target task. See Section 2.1 for a more detailed discussion of this area, and its differences with respect to the research reported in this thesis.

That said, it is not always possible to have the agent interact with the environment, nor is it always desirable. Sometimes the goal is develop a model of the agent for

analysis. For example, the model may be used to find weaknesses or strengths, or to find characteristic patterns that allow recognition of the agent when its actions are observed, etc.

We refer to this type of learning as *behavior modeling from logs*. In contrast to imitation learning, behavior modeling does not seek to produce an agent. Instead, its goal is to produce a model of an agent, by learning from its recorded interactions, i.e., *logs* of its actions. In Section 2.2 we discuss various behavior modeling methods in greater detail and contrast them with the research presented in this thesis.

2.1 Imitation Learning

In general, imitation learning involves inferring knowledge from observations, as well as experimenting and interacting with the environment, and then fine-tuning the knowledge to the agent. Such interactions with the real environment are not always possible or convenient because of cost and availability. For this reason, we decided to focus on the problem of learning from logs when there is no interaction with environments or simulated environments. The expert agent acts in an environment, and the observations along with the execution trace are recorded in logs. These actions are later imitated by the learning AI agent without accessing the environment.

The inference of knowledge from observations is a task that is also related to behavior modeling from logs. Thus, we present a brief review of imitation learning approaches, specifically *inverse reinforcement learning* (Section 2.1.1), *learning from demonstration* (Section 2.1.2), and *behavior cloning* (Section 2.1.3).

2.1.1 Inverse Reinforcement Learning

In traditional *reinforcement learning*, the goal is to learn a decision process to produce behavior that maximizes some predefined reward function. *Inverse reinforcement learning* (IRL) [64], also referred to as *inverse optimal control* [47, 51], as described by Russell *et al.*, flips the problem. Instead it attempts to extract the reward function from the observed behavior of an agent. IRL works in two steps. During a

backward step, the learner attempts to recover a reward function $R(s, a)$ from an expert demonstrator's policy or by observing this expert's policy. Then, in a *forward step* the learner uses the recovered reward function to determine a policy that maximizes $R(s, a)$ using reinforcement learning techniques. During the *forward step*, the agent tests the learned policy in the target environment; this is a step we avoid in behavior modeling from logs.

IRL requires a significant amount of interaction data to work. A recent approach called *generative adversarial imitation learning* (GAIL) [31] combines *generative adversarial networks* [25] with IRL. GAIL learns a policy without attempting to recover the reward function. This is done by using adversarial training to discover it from the expert's demonstration data and the learning agent's data. Another approach, called *state aware imitation learning* (SAIL) accomplishes a similar objective by using temporal difference learning to directly estimate the gradient of the normalized state-action visitation frequency [68]. While these methods are productive with regards to the amount of expert data needed for training, they require significant interaction with the environment, which is not the case in our work.

2.1.2 Learning from Demonstration

Learning from demonstration is another closely-related area of research that also relies on allowing the learning agent to interact with the environment. Here, the training trajectories are assumed to have been selected for the purpose of teaching and therefore contain information that is meaningful to the learning agent.

For example, Sullivan *et al.* [75] use *hierarchical training of agent behaviors* (HiTAB) to learn, in real-time, complex behavior in the form of *hierarchical finite-state automata* (HFA). Each state corresponds to an agent's behavior that can be represented as atomic behavior already written or another HFA. They apply their approach to the *keepaway* problem in the *RoboCup2D* soccer simulator domain; this domain is related to the domain we use in this thesis but is much simpler. A key assumption in HiTAB is that the agent's internal structure is known in advance. Every atomic behavior (i.e., state in the HFA) is learned separately from demonstrations,

and then the hierarchy and transitions are put together. In contrast, we do not assume that the internal structure is known. However, we do encourage the domain expert to provide information about possible goals of the agent, which are used to categorize the training data to improve learning.

Grollman *et al.* [26] model a RoboCup team goal scorer policy on an AIBO robot with a finite state machine. The states and transitions are learned using an approach called *realtime overlapping Gaussian expert regression* (ROGER). ROGER did not learn the behavior transitions. A key assumption ROGER makes is that the input comes from the robot's sensors, so there is no uncertainty as to what the learning agent perceived. This is not the general case that we tackle here. More importantly. ROGER assumes the actions have no parameters; thus, the number of possible actions is greatly reduced compared to the learning scenarios in this thesis, especially in Part I).

2.1.3 Behavior Cloning

Behavior cloning (BC) methods learn a direct mapping from states to trajectories/actions, without recovering the reward function. Under some settings, this can be an efficient way to reproduce the demonstrated behavior. The training data for behavior cloning usually consists of a set of demonstrated state-action pairs $\{(s^i, a^i)\}_{i=1}^N$ of an expert agent, where s^i is the agent's inner state at time i and a^i is the action at time i. The learned agent's policy $\pi(s^t) \rightarrow a^t$ can be learned as the direct mapping from state to action.

This learning problem can be formulated as *supervised learning* and solved by off-the-shelf regression or classification algorithms [7, 67]. One crucial problem with this solution is the fact that *supervised learning* assumes $\{(a^i, s^i)\}_{i=1}^N$ are *i.i.d.* Because behavior cloning learns the optimal action from only a single state value, it is unaware of the future state distribution the current action will produce. Thus, errors are compounded in the future states, leading to undesirable agent behavior as shown by Ross *et al.* [63].

There are two general approaches to behavior cloning. Model-free methods avoid

explicit learning or estimating the environment dynamics [58]. For this reason, they often do not require interactive learning with the environment and are considered simple to implement compared to model-based methods. However, it is a problem to validate whether an action/state is feasible in the environment, since the learner has no model of the environment to use for such validation. In contrast, model-based methods learn the policy by using information about the environment's dynamics from the forward model [80]. This is not always straightforward, given the complexities of the environment. In general, model-based methods are more time-consuming than model-free BC methods.

Torabi *et al.* developed *behavioral cloning from observation* (BCO) [78], which first learns the environment's dynamics through self-supervised exploration where the action does not always appear and must be inferred from the data. In the next step, they imitate the expert's behavior using classic BC methods. The optional last step requires interaction with the environment, which is not part of our study requirements. In our study, the premise is that the demonstrated action is implicit.

One of the fundamental challenges in behavior cloning is generalizing the learned policy to work in unfamiliar settings such as state trajectories that do not appear in the original data [5]). This would be similar to teaching an autonomous car to drive by cloning expert behavior where the learner reaches a state that the expert never encountered [58]. Here the learner's behavior will be unpredictable. A common approach to this challenge is to introduce interactions within the tested environment [63]; however, this is not relevant to our case study. Calinon *et al.* [12,30] developed methods to generalize alternate goals within a given domain. This is complementary to our approach in Part I of the thesis, where we allow the domain expert to provide information about the agent's goals.

2.2 Data Mining from Logs

There are different areas of relevant research that focus on unsupervised data-mining from logs. We briefly discuss two such areas: *grammar induction* (Section 2.2.1) and

sequential pattern mining (Section 2.2.2). We refer the reader to [13] for a short survey of using data mining for agents.

2.2.1 Grammar Induction

Grammar induction addresses the automated learning of grammars from data. Assuming the model of the agent is a grammar of some type (e.g., a probabilistic finite-state machine, a context-free grammar), it may be possible to apply grammar induction to construct the model of the agent.

Ryoo *et al.* [65] defines a game activity representation using *context-free grammar* (CFG), which enables a system to recognize events and actively provide proper feedback to the human user when the user takes unexpected actions. Ivanov *et al.* [35] defines *stochastic context-free grammar* (SCFG) rules to recognize more complicated actions, e.g., music conducting gestures using hidden markov models (HMM) based [59] low-level action detectors. Lee *et al.* [45] study a robot that imitates human demonstrations to organize objects using SCFG-based task-independent action sequences. In early work, Nevill-Manning *et al.* [52] presented the SEQUITUR algorithm, which can discover hierarchical structures among symbols. Solan *et al.* [71] presented the ADIOS algorithm, which induces CFGs and context-sensitive grammars, with some restrictions (e.g., no recursions), using graphical representations. Ogale *et al.* [53] constructed an SCFG grammar based on the frequency of human posed pairs (i.e., bigrams), considering slightly varying viewpoints. Kitani *et al.* [42] presented a framework that discovers human activities from video sequences using an SCFG induction technique based on work by Stolcke and Omohundro [72].

While grammar induction handles discrete actions and perception, our challenge is to model an agent that interacts with a continuous environment. Indeed, in grammar induction, each action would be represented by a single symbol or a distinct sequence of symbols (word). But we cannot make this assumption. In Part II, we present a method for hierarchically organizing discretizations of the data, to bridge the continuous environment represented in the logs with a learning method intended for discrete data. Although we focus on sequence mining, we could have also used

grammar induction as a technique to solve the hierarchy problem, we leave this for future work.

2.2.2 Sequential Pattern Mining

Understanding how agents act over time is critical to modeling the agent's behavior. Sequential pattern mining, first introduced by Agrawal *et al.* [3], has become an active research area in *data mining*. A survey on *sequential pattern mining* techniques can be found at [48]. We focus here on reviewing work that is closely related to *behavior modeling*.

Based on the premise that past events influence the behavior of a player, Iglesias *et al.* [34] tried to classify the resulting behaviors across supervised learning (Chi-square test) sequences. They analyze the data from logs and create sequences. These sequences are composed of events such as pass, dribble, steal, and goal. The results are preliminary and the component responsible for correlating the acquired knowledge with the strategy to be adopted by the team presented several limitations. In the same context, Abreu *et al.* [2] divided the input space offline using clustering and associated a strategy with each cluster. However, these approaches can focus on clustering the strategies while focusing on finding the next action. Moreover, in Part I expert guidance serves as a necessary step for modeling behavior.

Time series and decision tree learning are used by Visser *et al.* [81] to induce rules that describe a team's behavior. The key idea of that research differs from ours. In their case, an object in a complex environment is seen as a time series. A qualitative abstraction of these time series is applied. Then, these time series are discretized to use the results for learning by C4.5, which cannot capture the temporal ordering of events. Instead, the temporal ordering is captured by the qualitative abstraction of the time-series. In contrast, our work directly tackles the temporal ordering of events.

Kaminka *et al.* [37] recognized basic actions based on descriptive predicates, and learned relevant sequences of actions using a statistical approach. Horman and Kaminka [32] expanded on this approach. A similar process is also used by Huang et al. [33] to find frequent patterns in dynamic scenes. Leece *et al.* [46] studied frequent

sequences occurring in *StarCraft: Brood War* replays to understand micro-level (e.g., unit movement) and macro-level behavior (e.g., build orders). However, like most related work in this area, these previous works assume that actions are discrete and finite. In this thesis, we focus on actions that have metric parameters and thus require discretization, which is a challenge when using the techniques above.

Sequential analysis is important in understanding the interactions between players and games. Kang *et al.* study [82] lag sequential analysis (LSA) to analyze the sequential behavior patterns of players. The sequential analysis is conducted on players' behavioral code to objectively analyze the quality of video games. Han *et al.* [28] learned sequences of game level elements; then sequences of player behavior are analyzed to find the connection between in-game content and player experiences, thereby helping improve the quality of automatically generated game content. In our work, we try to model the agent's behavior, based on the demonstration given by logs. Thus, we aim for a model capable of accounting for decisions, not user satisfaction or experience.

Our work in Part II builds on LASH [6], a hierarchical sequence-mining algorithm designed for large-scale sequence datasets. It is the first parallel algorithm to discover frequent sequences by considering hierarchies that are given to it as input. We discuss our use of LASH in Part II.

Part I

Expert-Guided Learning of Continuous Actions from Logs

Chapter 3

Learning Actions with Continuous Parameters from Logs

This chapter first introduces an example of the challenging domain that motivated us to develop and test the techniques presented in this thesis (Section 3.1). We then explain how to transform the global state information that appears in the logs into the egocentric state representation used by agents. In this way, we bring the log data closer to our learning goals (Section 3.2). Finally, we present the baseline results from applying various state-of-the-art classifiers to learn action parameters in this domain (Section 3.3), without expert guidance or intervention.

3.1 RoboCup 2D Simulation: A Challenging Domain

The *RoboCup2D* soccer simulation, shown in Figure 3-1 on page 18, is one of the most highly investigated robot competition domains [4, 50]. In the game, 11 autonomous agents play soccer on a simulated two dimensional (2D) field with 5-minute halves. The game lasts for 6,000 simulation steps, and each one takes 100 milliseconds.

In each of these steps, the agents receive sensory information such as the ball location, the locations of viewed agents, and field features such as lines or field corners. The agent's visual sensor (which is not recorded) reports the objects currently seen, in an egocentric state coordinate system: distances in meters and angles in degrees

Figure 3-1: *RoboCup2D* simulated soccer environment. Screenshot taken from [62].

relative to the current agent's pose, and head orientation for each step. Other agents may not be visible to the agent, or they may be completely or partially visible. For example, the agent may only know their team name but not the uniform number that identifies them. In this case of incomplete data, the respective piece of information is left out. The server can also provide information about an agent, the ball velocity, the agent's own stamina, and more.

After processing this information, agents select actions such as dashing (moving), kicking the ball, or turning, each of which may accept parameters. For example, the *turn* action accepts a numeric parameter indicating the change of the agent's body's angle. Similarly, the kick command has two parameters: the first parameter is the power [0, 100] and the second parameter is the relative direction [0, 360]. The *RoboCup2D* simulation server manual [14] provides a detailed description of the domain. It lists the main actions an agent can take and the information the agent perceives from the surrounding environment. Table 3.1 on page 19 lists a few of the actions and their parameters that can appear in $\overrightarrow{A_t}$.

Action Name	Parameter	Meaning	Parameter Range	Description
Kick	P1 P2	Power Direction(degrees)	[0, 100] [0, 360]	Kick the ball with power(p1) with relative direction(p2)
Tackle	P1 P2	Direction Foul	[0, 360] [True/False]	Contest the ball with direction(p1) while intend to foul(p2)
Dash	P1 P2(optional)	Power Direction(degrees)	[-100, 100] [0, 360]	Moves the agent with power(p1) and relative direction(p2)
Turn	P1	Direction(degrees)	[0, 360]	Change agent body angle.
Turn Neck	P1	Direction(degrees)	[0, 360]	Change agent head angle.

Table 3.1: *RoboCup2D* actions and parameter metadata.

We use the RoboCup 2D simulation here because of the complexity of the agents' state and the wealth of logs available from competitions over the years since 1997[1]. These logs represent a prime example of the behavior modeling task investigated in this thesis, and the challenges that arise in carrying it out.

Although the soccer simulator communicates with agents using their exact agent egocentric state, these individual messages are not recorded in the simulator log files. Each simulation generates several recorded files:

- A recording of the game, including the global environment state $\overrightarrow{E_t}$ in a global coordinate system, with files distinguished by suffix .rcg.
- The action from agents as received by the simulator ($\overrightarrow{A_t}$) in plain text, with files distinguished by suffix .rcl.

Separating a log file into individual logs for each agent gives us 22 logs. Each such individual logs is essentially composed of tuples (e_t, a_t), where t is a time stamp, E_t is a snapshot of the environment state at time t, and a_t is an action, with parameters selected by the agent. From this limited information, we must incrementally build a model of each agent's decision-making process π.

[1]Logs are available from: `http://chaosscripting.net/files/competitions/RoboCup/`

3.2 From World to Egocentric Coordinates

Our first step is to transform the log information into data from the agent's perspective. This is a key challenge in any domain, and the *RoboCup2D* simulation is no different. In general, the agent's egocentric state (S_t^i) is not recorded inside the logs, and must be inferred from $\overrightarrow{E_t}$. For example, the logs do not reveal which of the objects and agents on the simulated field are actually perceived by each agent. Moreover, perceived details about other agents or objects are not recorded in the logs.

3.2.1 Perceived Objects and Agents

The RoboCup simulation server manual [14] notes that the agents receive information using their own egocentric coordinate system. Thus, our first step is to transform $\overrightarrow{E_t}$ into the corresponding S_t^i, where positions, velocities, and angles are recorded from the agent's perspective. We used the manual as a guide to help us decide which objects or agents were perceived by each agent, and how to derive the distance, the view angle (field of view).

Table 3.2 on page 21 lists a few of the features of the agent's egocentric state (S_t^i). It also shows the feature type along with a short description.

The field-of-view angle, the agent's direction, and the neck angle are used to derive the egocentric coordinates of objects and other agents in the nearby surroundings. Table 3.3 on page 22 shows the features kept for each object (ball or goal) estimated to be visible to the agent. The last column shows a mark when we need expert help, based on the manual, to determine the value of a feature. We computed the *visible object id probability* as described in the manual, and considered an object or agent to be visible if the probability was greater than 0. Similarly, Table 3.4 on page 22 presents the features kept for each visible agent; these agents are considered visible if the calculated *visible agent team probability* is greater than 0.

Feature name	Type	Description
x agent pos	float	The agent's x-position on the field
y agent pos	float	The agent's y-position on the field
x agent velocity	float	The agent's x-velocity on the field
y agent velocity	float	The agent's y-velocity on the field
agent body angle	cyclic angle	The agent's body angle
agent neck angle	cyclic angle	The agent's neck angle
robocup agent type	int	The agent type (constants for each game)
view quality	char	The agent view quality: high(H), normal(N) or low(L)
view angle	cyclic angle	The agent view angle
agent stamina	float	Low stamina slows movement.
robocup focus side	char	Which team does the agent focus on: Teammate ('T') or Opponents ('O'))
robocup focus num	int	Which agent number (1-11) does the agent focus on
team name	str	The agent team name
agent id	int	The agent ID
playmode	str	The environment play mode (offside/out of bound/foul etc.)
perceived objects	list	List of perceived objects (see Table 3.3)
perceived teammates	list	List of perceived teammates (see Table 3.4)
Perceived Opponents	list	List of perceived opponents (see Table 3.4)

Table 3.2: Agent's belief and perceived state features.

Feature name	Type	Description	Using expert guide based on the manual
x object pos	float	The object's global x-coordinate on the field	
y object pos	float	The object's global y-coordinate on the field	
x object velocity	float	The viewed object's x-velocity on the field	
y object velocity	float	The viewed object's y-velocity on the field	
relative angle	cyclic angle	The bearing of the object offset from the agent's neck angle	√
distance	float	The distance to the object	√
object id	int	ball/own goal/opponent goal	
visible object id probability	float	The probability that the viewing agent will recognize the object ID	√

Table 3.3: Viewed object list of features.

Feature name	Type	Description	Using expert guide based on the manual
x agent pos	float	The agent's global x-position on the field	
y agent pos	float	The agent's global y-position on the field	
x agent velocity	float	The agent's x-velocity on the field	
y agent velocity	float	The agent's y-velocity on the field	
speed	float	The agent's speed (velocity vector magnitude)	√
heading	cyclic angle	The direction in which the agent is heading	√
relative angle	cyclic angle	The relative bearing to the agent, offset from the neck angle	√
distance	float	The distance to the agent	√
agent id	int	The agent's id (1-11)	
visible agent id probability	float	The probability that the viewing agent will recognize the agent id	√
visible agent team probability	float	The probability that the viewing agent will recognize the agent teams	√

Table 3.4: Viewed agent list of features.

3.2.2 Cyclic Features

The representation of angles is an issue common to transformations from a global coordinate system to an egocentric polar coordinate system (distance and bearing). For example, in the RoboCup 2D domain, the body angle is given in a global coordinate system, while the neck angle is given in angles offset relative to the agent's current heading. Similarly, the same representation is also used in action parameters ($\overrightarrow{A_t}$), such as the kick direction, the tackle direction, turn and turn neck directions, and so forth.

Angles are one example of what is referred to in machine learning as *cyclic* features. Cyclic features have numeric values that cycle from the maximum value back to the minimal value. For example, 360 degrees is the same as 0 degrees, and expanding a 359-degree angle by 2 degrees involves an angle of 1 degree, not a 361 degree angle. Time in hours and seconds is often a cyclic feature as well. Adding 20 minutes to 23:50 is given as 0:10 of the next day, not 24:10.

Cyclic features pose a challenge to machine learning algorithms because they have semantics that are different from ordinary numeric features. Thus, they must be marked explicitly for the machine learning algorithm or replaced by equivalent features that avoid the cycling. We transformed each angle to a tuple in the egocentric state (S_t^i): $deg \in [0, 360] \rightarrow \langle deg, sin(deg), cos(deg) \rangle$, where $sin()$ and cos refer to the familiar sine and cosine functions.

3.3 Initial Results

As a baseline for the research presented in this thesis, we focus on autonomous learning of agent action parameters without human guidance (other than interpreting the manual and accounting for cycle features). We show below the results for the *kick* action, as it serves to illustrate the magnitude of the challenge to the learning system.

To learn the agent's kick behavior, we want to predict the kick policy $\pi^t(S_t^i) \rightarrow (P1, P2)$ that will map from the agent egocentric state (S_t^i). The basic question is: Assuming that a decision is made to take a kick action, can the learning system

predict the kick's parameters: $P1$ and $P2$?

3.3.1 Experiment Setup

The learning process dataset we used is from 41 teams over 3 sessions of *RoboCup2D* WorldCup[2] (2013 - 2015). We used the logs for all teams, including the *Gliders (2013)*, whose logs contain more than a single kick at the same timestamp, which is illegal according to the manual rules.

We consider logs with the same team name over a number of years as different, based on the assumption that the agents' policies change throughout the years. However, to simplify and generalize the *supervised learning* of the parameters for the agent's actions ($\overrightarrow{A_t}$), we chose to merge all the observations of agents in each team, excluding the goalie, and treat a team as one agent when it comes to behavior. In other words, we assume that the all agents in the team use the same policy. Appendix A provides a high-level description of the system built to convert the logs mentioned above into one merged log with 6,000 records of $\langle \overrightarrow{E_t}, S_t^i, \overrightarrow{A_t} \rangle$.

We experimented with various learning algorithms, from the open-source package scikit-learn[3] and used the default values for all parameters. All experiments were run on a computer with an 8-core Intel i7-8550U CPU, under Ubuntu Linux (version 18.04.2 LTS), running at 1.80 GHz, with 16 GB of RAM.

3.3.2 Baseline Results

The consolidated logs were divided into training (75%) and testing (25%) sets. We experimented with both types of supervised learning algorithms: *regression* and *classification*. Regression algorithms predict the numeric values of parameters. Classification algorithms predict discrete labels. But, given a discretization of the parameter numeric values, they can be used to predict the discretized values.

Figures 3-2 to 3-4 show the results of running 33 different classification and regression algorithms using default parameters. These were used to predict the $P1$ and $P2$

[2] http://chaosscripting.net/files/competitions/RoboCup/WorldCup/

[3] `https://scikit-learn.org/` [56].

parameters of the kick command, based on the state of the agent as known at the time of the kick. In all of the figures, the horizontal axis marks the algorithm used. The vertical axis marks the mean squared error (MSE, Fig. 3-2), the root mean squared error (RMSE, Fig. 3-3), or the mean absolute error (MAE, Fig. 3-4), respectively. A lower result is better. The box-plot graph shows the distribution of the learning results from testing all teams. Each box plot aggregates 41 data points.

A matching view of the results is given in Figure 3-5, which measures the explained variance in the results. Here, a higher result is better.

For the classification algorithms, we experimented with various uniform discretizations. The name of the algorithm shows the discretization level. For example, the classifier *ExtraTreeClassifier[5, 30]* divides the $P1$ range by 5, and the $P2$ range by 30. Here are some additional examples, for further clarification:

- *kick* $P1 = 97, P2 = 303$ is given the label *kick[100, 300]* $round(97/5) \cdot 5 = 100$, $round(303/30) \cdot 30 = 10$
- *kick* $P1 = 103, P2 = 296$ is given the label *kick[100, 300]* $round(103/5) \cdot 5 = 100$, $round(296/30) \cdot 30 = 10$
- *kick* $P1 = 2, P2 = 10$ is given the label *kick[0, 0]* $round(2/5) \cdot 5 = 0$, $round(10/30) \cdot 30 = 0$

All graphs led to the same conclusion. We see that the *regression* algorithms are generally better than the *classification* algorithms. These regression algorithms are *ExtraTreesRegressor* [24], *RandomForestRegressor* [9], *AdaBoostRegressor* [20], *GradientBoostingRegressor* [22,23,29] and *HistGradientBoostingRegressor* [15,39,61].

The baseline experiment shows the kind of performance we can expect from providing the agent with relatively simple transformations of the environment state to its own egocentric state (S_t^i). These results are even more disappointing given that the settings for the learning task at hand were relatively easy: predict the two parameters of a single action. It demonstrates how daunting the full task of learning a model of an agent in this domain can be. The next chapters demonstrate how the judicious

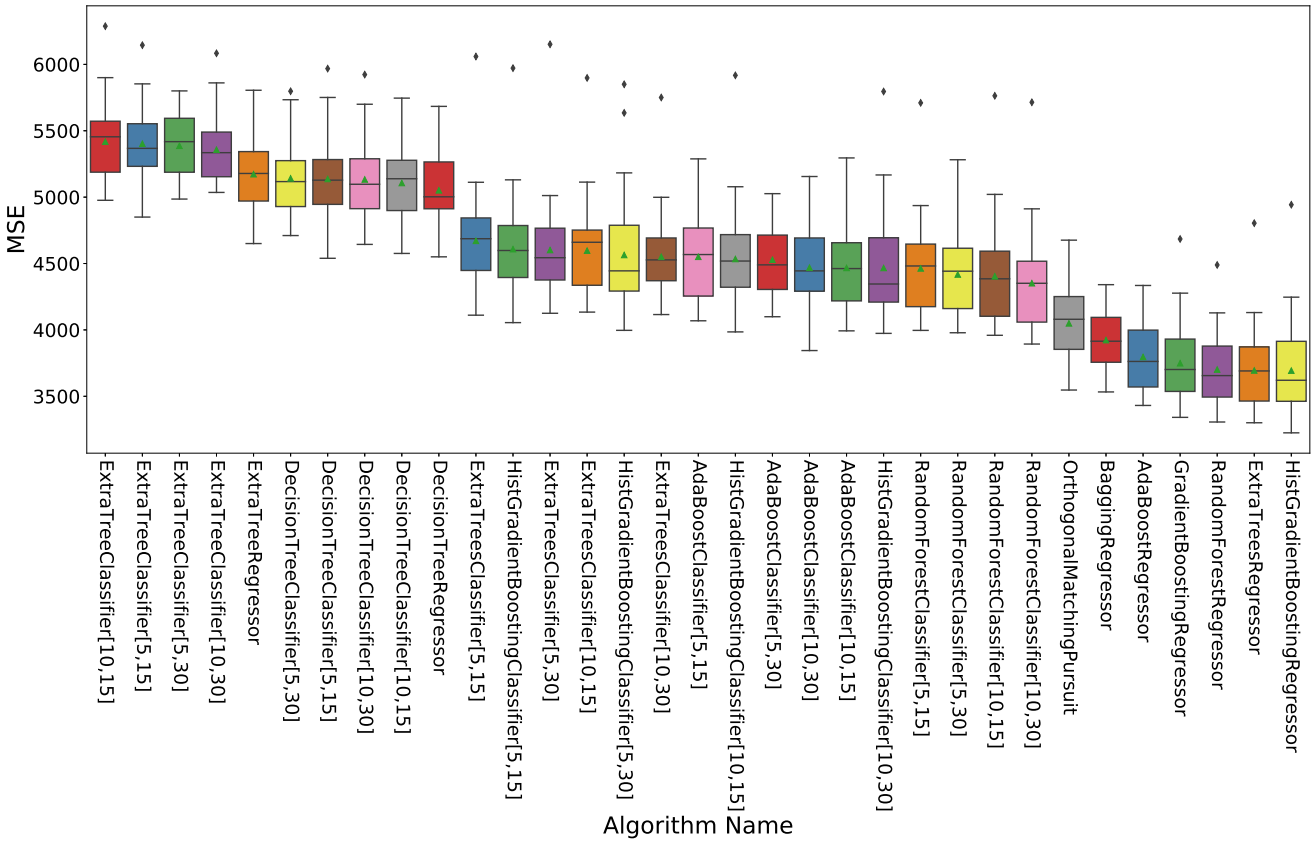


Figure 3-2: MSE score over 33 off-the-shelf supervised learning algorithms, lower results are better.

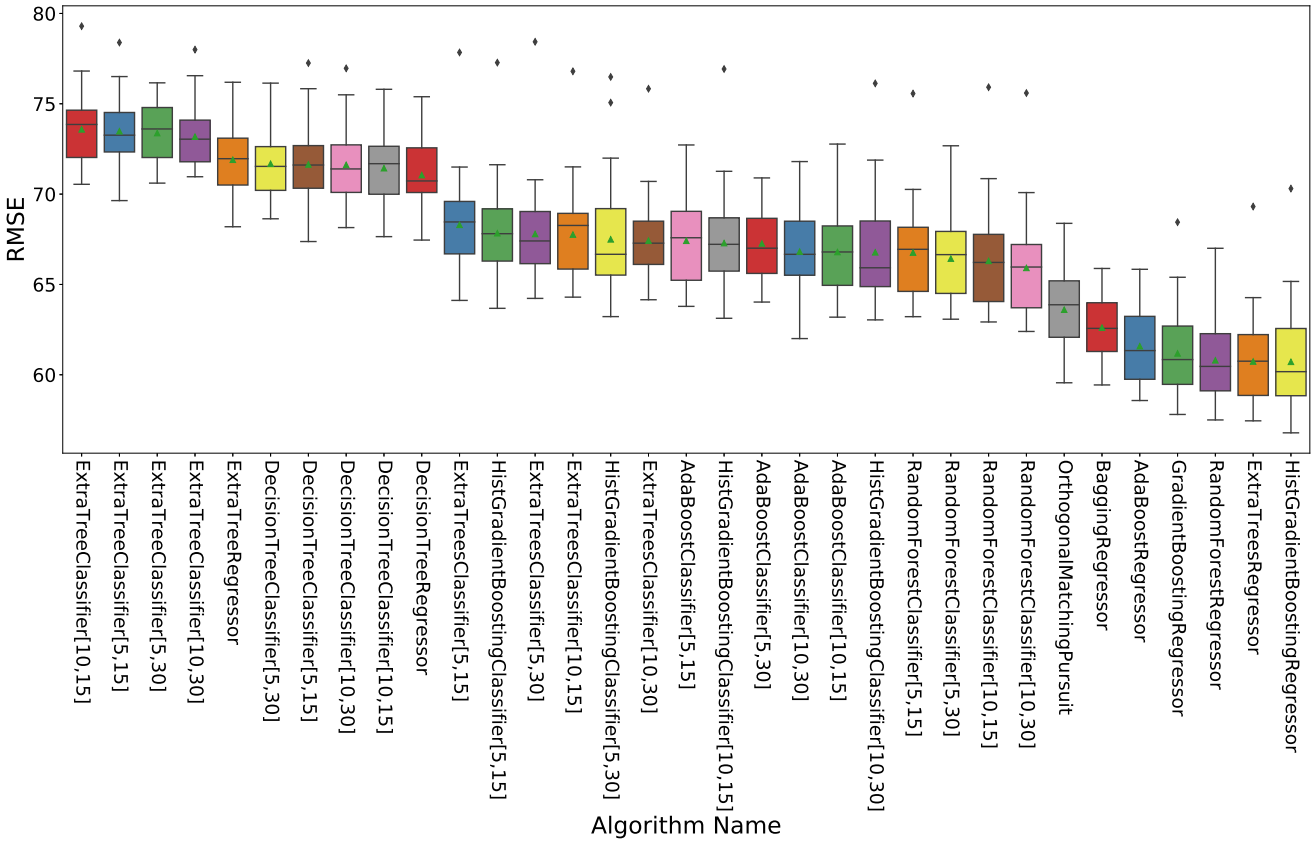


Figure 3-3: RMSE score over 33 off-the-shelf supervised learning algorithms, lower results are better.

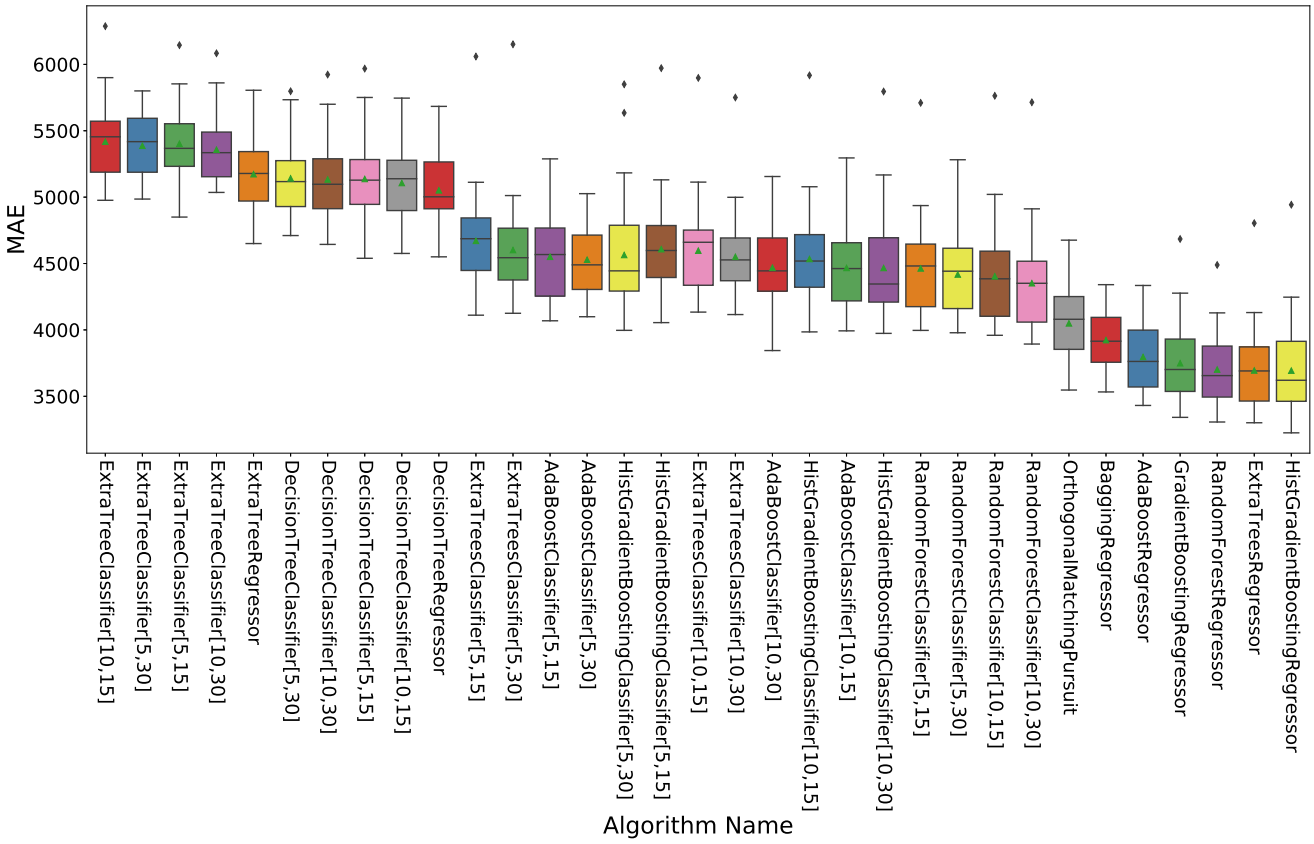


Figure 3-4: MAE score over 33 off-the-shelf supervised learning algorithms, lower results are better.

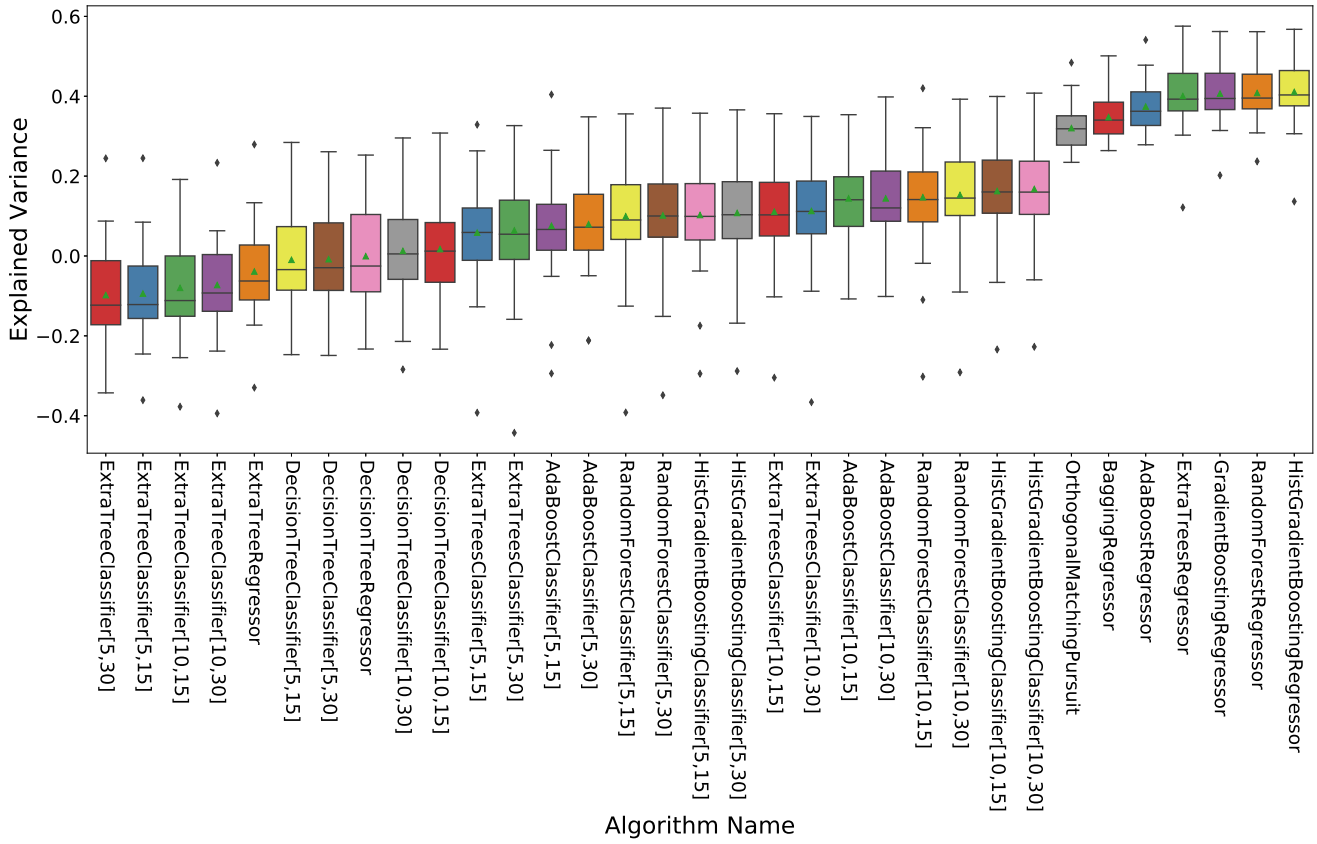


Figure 3-5: Explained variance score over 33 off-the-shelf supervised learning algorithms, higher results are better.

and focused use of domain knowledge, given by a domain expert, can significantly boost the performance of the learning system.

Chapter 4

Adding Domain-Specific Belief Features

In this chapter, we explore the effect of adding features whose significance to predicting the agent's behavior relies on knowledge of the domain at hand. Relying on a domain expert, we improve and add to the state of features in the egocentric agent state, while adding the agent's beliefs (B_t^i). Thus, the learning task is transformed into learning the mapping $(S_t^i, B_t^i) \rightarrow \overrightarrow{A_t}$.

The key is to add features whose values can be derived from the existing state features, and significantly affect the agent's decision-making process. For example, suppose the agent is close to the opponent's goal and has the ball. This fact has significance in a game of soccer, but requires domain expertise to identify. In this case, a domain expert may suggest adding another feature whose value is true when the agent is in this significant position, and false otherwise. The value of such a predicate can assist the machine learning algorithm in distinguishing between kicks intended for a shot on the goal, from, say, a pass to a teammate.

4.1 Features Affecting Decisions

Much of the agent's internal belief structure uses information derived from the raw perceptual data made available from the logs. Background information about the domain is also used here to drive the agent's decision-making process.

In the RoboCup 2D soccer domain, the log information does not carry any infor-

mation about the field structure. This information, shown as a map in Figure 4-1, is extremely important in understanding agent behavior. Much of the geometric information provided to the agent can be understood in relation to this map. Furthermore, the map, combined with domain knowledge of soccer, allows us to derive additional features from the available data.

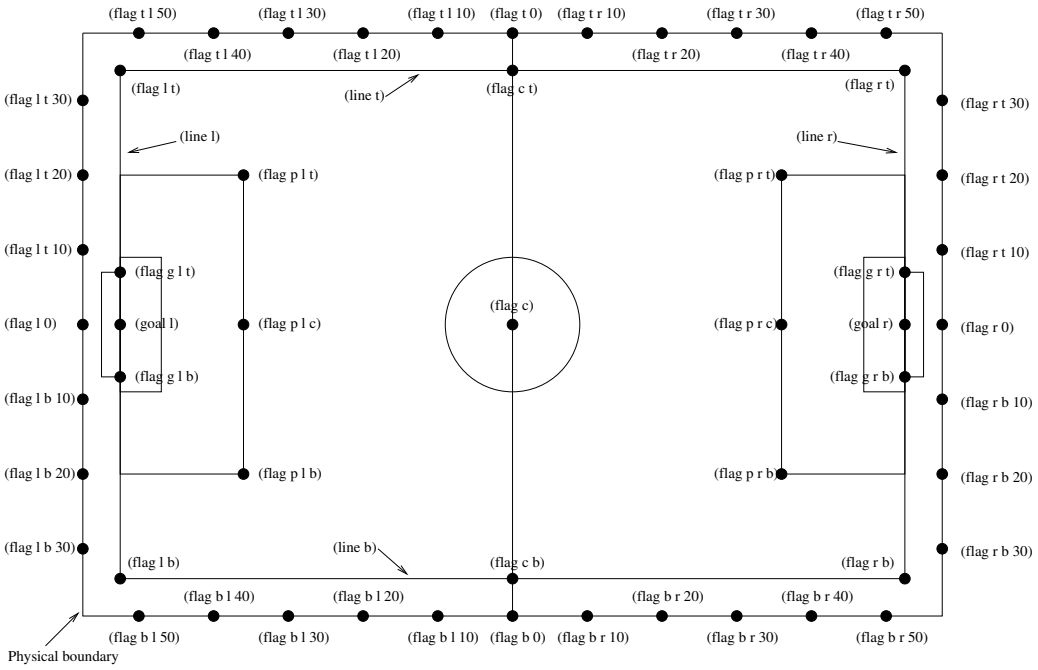


Figure 4-1: RoboCup 2D soccer field map, taken from [14].

For example, when the ball is on the agent's own team side of the field, the behavior of the agent is likely to be different than otherwise. Likewise, the distances of the agent from its own goal, and that of the opponent, and its angles to either, also affect its decision making. Another example considers whether the ball is in the opponent's penalty box or in the agent's own team area.

In some cases, continuous-valued features can make it more difficult for learning algorithms to build a good model for prediction. The machine learning algorithm cannot easily tell whether a distance of 40.1 meters to a goal is meaningfully different from a distance of 40.2, for example. To alleviate the learning task, it is advantageous to discretize such values, in such a way as to reflect their domain significance. We, again, look to rely on a domain expert in this task.

For example, consider the ball's distance from the opponent's goal. The expert can define that three meters should be interpreted as *near the goal*, distances within the penalty box can be denoted as *close to the goal*, distances outside of it will be considered *far*, and so on. These distinctions may ease the task of the learning algorithm in predicting the kicking power needed. Similar examples may include distinguishing whether a teammate is *open* for receiving a pass, whether the nearest opponents may be considered a threat to dribbling, etc.

Table 4.1 lists all the features that were derived from considering the background knowledge of the agent, and are regarded as important by the domain-expert.

4.2 Reordering Internal Beliefs

The agent's internal belief state is often complex, i.e., contains sub-structures. For example, it may contain lists of perceived objects and agents that may sometimes separate teammates and opponents. This is true in many environments, and also true of the RoboCup 2D simulation domain used in this thesis (see Table 3.2). Each of the elements in these lists is described by its own multiple features, such as distance, heading, and so forth.

Unfortunately, state-of-art machine learning algorithms are incapable of natively handling complex features. A common solution is to flatten the complex structure and impose an ordering on the list. For example, this may be done by using an index number for each element, and appending it to each of the sub-features of each element.

Unfortunately, the fixed arbitrary ordering of the lists can lead to the loss of important semantic information. For example, the list of teammates may be ordered by their increasing distance from the agent, or by their openness for passing, by their uniform number, or by their type. The choice of the ordering will therefore impact the results of the learning, as the learning is not aware of order semantics, and only considers the feature name conveying its index (e.g., "3rd agent in list").

One of the notable improvements in predicting the *kick* action was made by re-

Feature	Type	Description
ball distance - bottom outside	float	The distance of the ball from the bottom outside line (line b)
ball distance - top outside	float	The distance of the ball from the top outside line (line t)
distance from opponent goal middle point	float	The distance from the opponent's goal middle point
distance from own goal middle point	float	The distance from the agent's own goal middle point
angle from opponent goal middle point	cyclic angle	The cyclic relative angle from the opponent's goal middle point
angle from own goal middle point	cyclic angle	The cyclic relative angle from the agent's own goal middle point
distance from opponent goal top point	float	The distance from opponent's goal top point
distance from own goal middle top point	float	The distance from the agent's own goal top point
angle from opponent goal top point	cyclic angle	The cyclic relative angle from the opponent's goal top point
angle from own goal top point	cyclic angle	The cyclic relative angle from the agent's own goal top point
distance from opponent goal bottom point	float	The distance from the opponent's goal bottom point
distance from own goal bottom point	float	The distance from the agent's own goal bottom point
angle from opponent goal bottom point	cyclic angle	The cyclic relative angle from the opponent's goal bottom point
angle from own goal bottom point	cyclic angle	The cyclic relative angle from the agent's own goal bottom point
Ball in own *goal area*	bool	If the ball is inside the agent's team *goal area*
ball in opponent *goal area*	bool	If the ball is inside the agent's team *goal area*
ball in own box	bool	If the ball is inside the agent's own *penalty box*
ball in opponent box	bool	If the ball is inside the agent's opponent *penalty box*
offense direction to right side	bool	True if the offense direction is the right side, false for left side
ball in offense	bool	True if the ball is in the offense court side, false in case of defense court side
ball close to bottom outside	float	If the distance of the ball from the bottom outside line is less than 3 meters
ball close to top outside	float	If the distance of the ball from the top outside line is less than 3 meters
opponent is near by	bool	If the distance of the agent from any opponent agent is less than 3 meters

Table 4.1: List of added domain-specific features.

ordering the list of perceived agents. The initial ordering was simply based on the timing of their perception (i.e., how recently each agent was observed). Reordering the lists of teammates and opponents by increasing distance, led to significant improvements in the learning results, as is noted in later chapters. We were able to raise the r^2 scores from approximately 0.4 to just under 0.6 for the top regression algorithms by combining this re-ordering with the domain-specific features described in the earlier section.

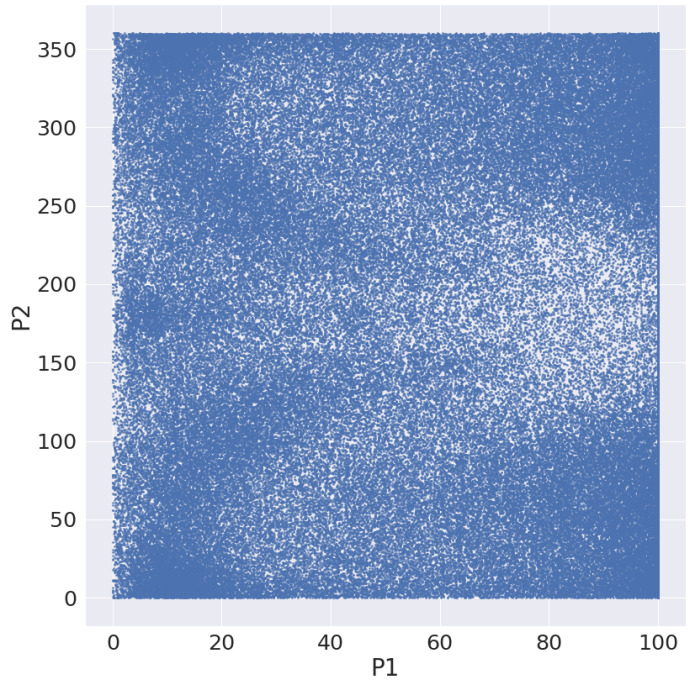
4.3 Mirroring Action's Parameters

Some internal belief state features are difficult to generalize for a machine, while an expert guide can simplify the task. For example, degrees may be symmetrical for right and left side, due to the fact that agent might not have side preferences (though a human might). This is true in many environments, and also true in the RoboCup 2D simulation domain used in this thesis.

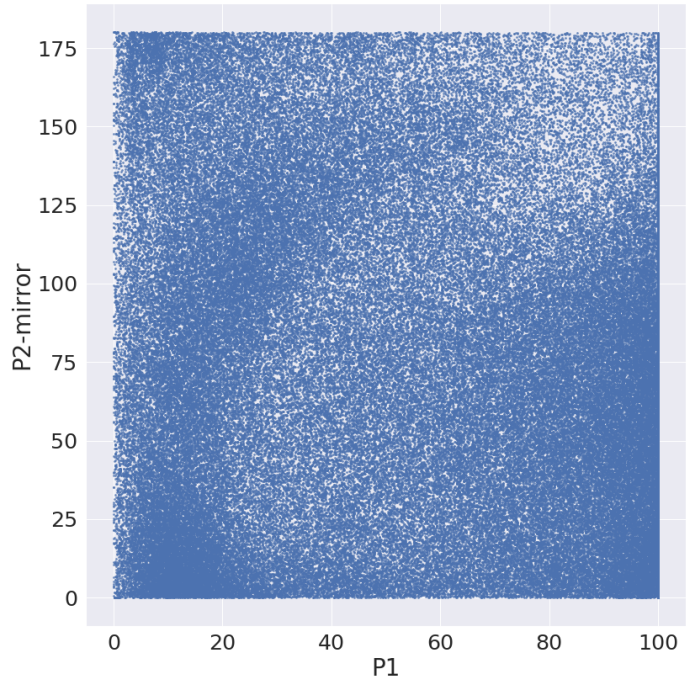
In kick action for example we plotted P1 and P2. We observed that there is a *mirror transform* with P2 over the 360 degrees, as shown in Figure 4-2a. As a result of this *mirror transform*, we chose to generalize the P2 parameter and ignore the right/left direction to generalize G_t^i, the plot with applying the *mirror transform* shown at 4-2b: degrees in [-180, 0] were transform to [0, 180].

In the tackle action we plot the histogram over P1 by seprate to true and false case of P2 in order to see the *mirror transform* shown at 4-3.

Each of the action parameter which used degrees in Table 3.1 on page 19 was "mirroring", and was similarly transformed.

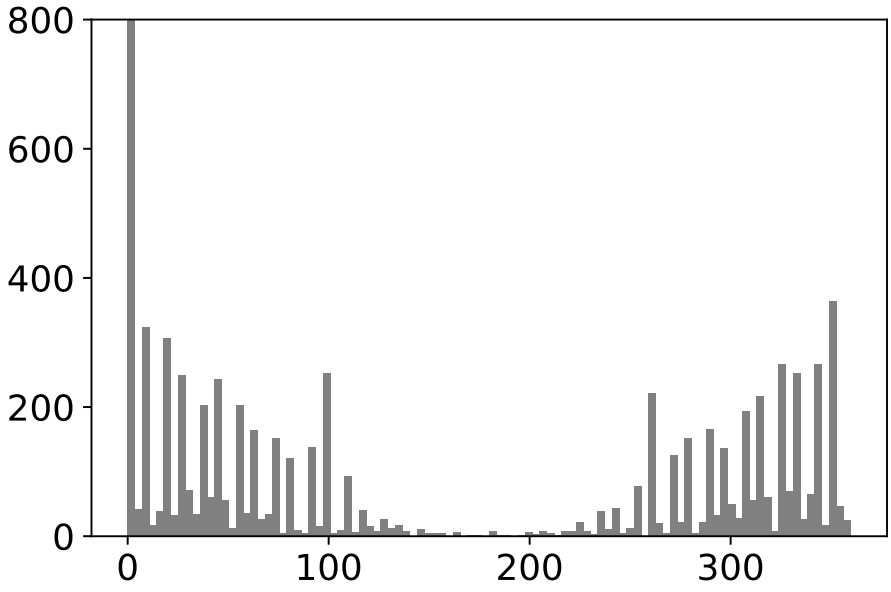


(a) P2 without applying mirroring right and left

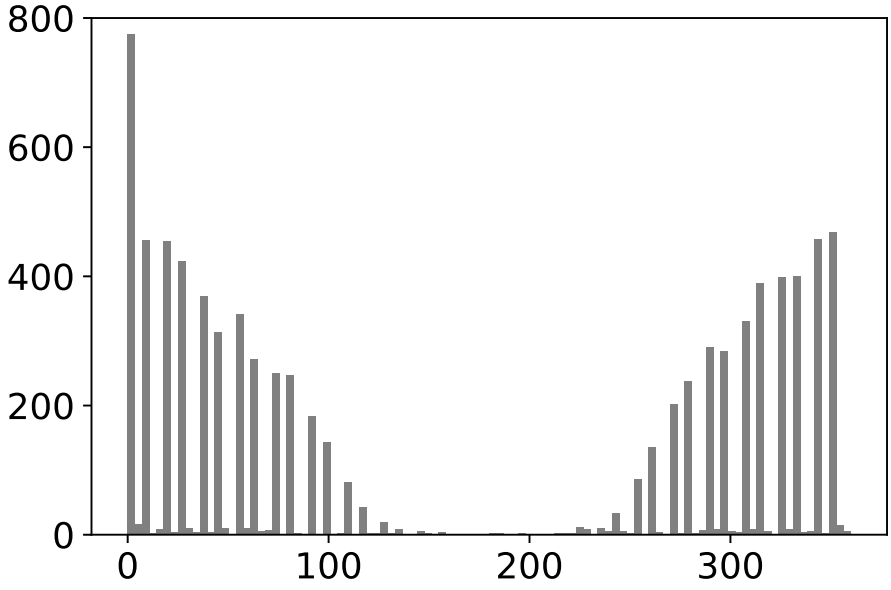


(b) P2 with mirroring transformation right and left

Figure 4-2: Fast Forward prediction over kick action with P1 and P2

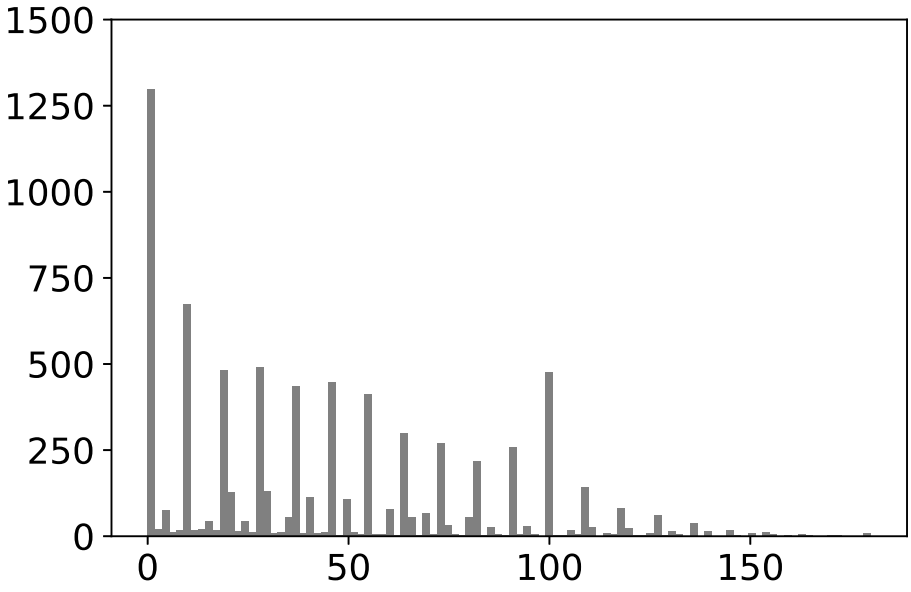


(a) P1 without applying mirror transformation (P2==False)

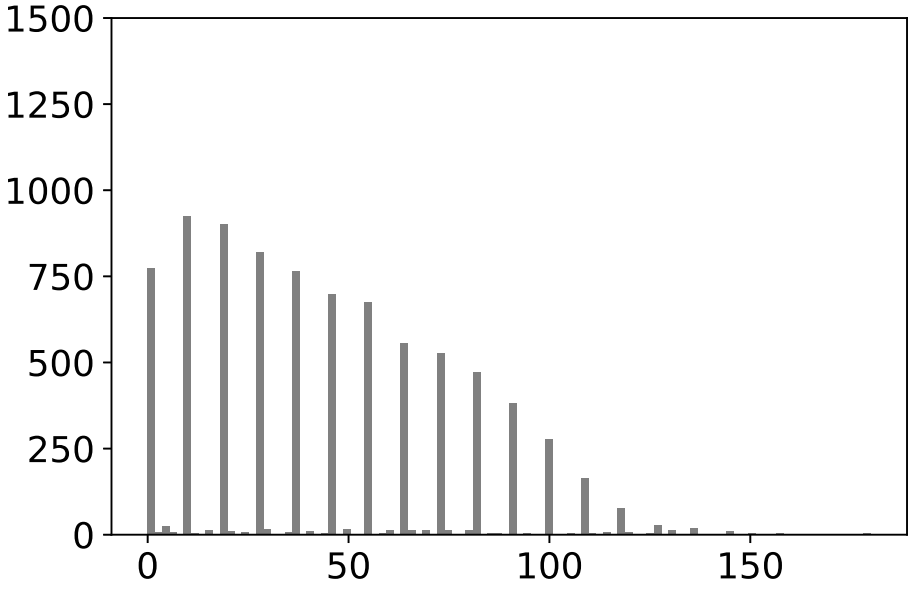


(b) P1 without applying mirror transformation (P2==True)

Figure 4-3: Tackle P1 histogram



(c) P1 after applying mirror transformation (P2==False)



(d) P1 after applying mirror transformation (P2==True)

Figure 4-3: Tackle P1 histogram

Chapter 5

Adding Agent Goals as Features

While the previous chapter considers domain-specific beliefs that are given by the human domain expert and may be held by the agent, this chapter focuses on the agent's possible goals in the domain. When kicking the ball, a soccer-playing agent may intend to pass the ball to a teammate, to clear the ball, to score, to dribble, etc. The intent of the agent, as expressed in its decision-making, is only evident—if at all—when considering the results of the action later on, and not at the time of the action itself. Yet, that same intent will significantly impact the choice of actions and/or their parameters.

If we could identify the intent of an action (i.e., its overarching goal), we could potentially help the learning process better predict the actions and their parameters. This again requires some interaction with a domain expert to understand what possible intents an agent may have G_t^i, and which actions parameters are used in the context of different goals.

This further transforms the learning task. We now aim to learn $(S_t^i, B_t^i, G_t^i) \rightarrow \overrightarrow{A_t}$. This general flow of identifying these feature will be explained in Section 5.1. We will build G_t^i in two complementary steps. The first step is termed *fast forward* (Section 5.2), a technique by which the logs are read forward in time, to identify what was the domain-specific goal for an action. The goals are given by the expert. This leads to very significant improvements in the results of the learning, but of course cannot be used without knowledge of the agent's goals. Thus a separate mechanism

can learn to identify the purpose of an action without such skipping ahead.

A second independent step, *parameter clustering* (Section 5.3) is used to identify clusters of parameters that are visually and automatically clustered together, with the hypothesis that these belong to some internal execution state. Once the clusters are identified, a separate learning mechanism can be used to identify them, and thus improve the overall prediction of agent actions from the logs.

5.1 Goal Feature Learning Flow

We add the goal state G_t^i as a feature of the agent's egocentric state S_t^i in addition to beliefs state B_t^i. We create the feature using domain expert to label the example, in Section 5.2 based on fast forward goals in Section 5.3 based on visualized cluster. This is *Step 1* in flow presented in Figure 5-1 on page 41. The user tags the goals in either of two ways (Section 5.2 and 5.3)

Step 2 describes the addition of these additional goal features (shown as Perfect since the manual classification is the ground truth), which significantly boost the learning results given the same setup described earlier (see next chapter for the results). However, as our mission is to predict the actions taken based on a given state. We cannot rely on having access to the goal information (that was taged by our domain expert) at the time the prediction is needed. We can only count on using S_t^i, B_t^i.

We therefore add a separate learning step (*Step 3* in Figure 5-1 on page 41) that learns the G_t^i tag using an off-the-shelf classification algorithm as *HistGradientBoosting* [15,39,61], *GradientBoosting* [22,23,29], *Bagging* [8], *AdaBoost* [20], *ExtraTrees* [24], or *RandomForest* [9]. The algorithm learns $(S_t^i, B_t^i) \rightarrow G_t^i$. In essence, we are using an off-the-shelf classification algorithm to predict the goal of the action. Later in *Step 4*, we use the classifier (from *Step 3*) to predict the goal feature (shown as AUTO). This feature will be used in *Step 5* to generate the agent's beliefs state in the same learning task.

In *Step 2* to *Step 5*, we divided the logs into training (75%) and testing (25%) sets

with 3-fold cross validation. In *Step 2* and *Step 5*, we experimented with *regression* algorithms because they achieved better results than *classification* algorithms in our initial tests discussed in Section 3.3.2. We picked three of the regression algorithms based on the best results and shortest run-time for the tests in Section 3.3.2. These algorithms were: *HistGradientBoosting* [15, 39, 61], *ExtraTrees* [24], or *RandomForest* [9].

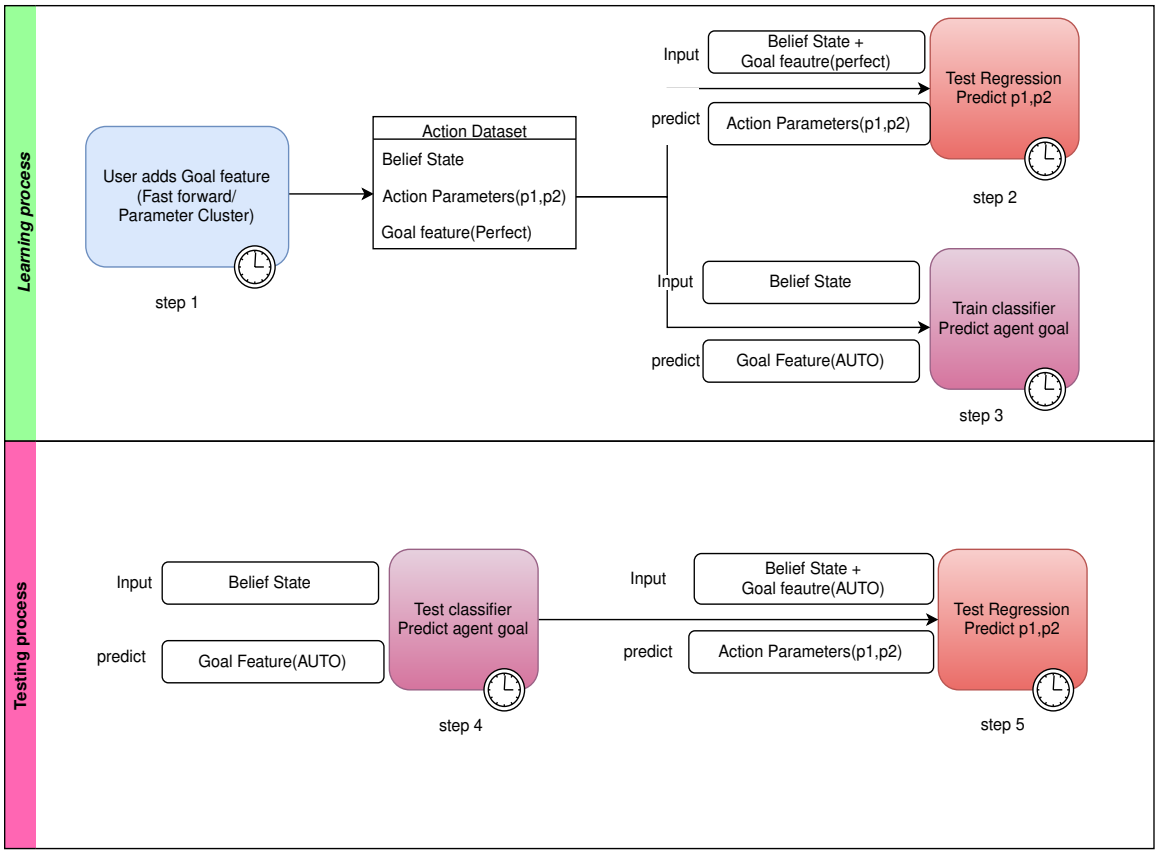


Figure 5-1: The full discovery flow of adding agent goal (fast forward/parameter clustering) as a feature

5.2 Fast Forward

An action taken by a well-designed agent is done with some goal in mind. In simple domains, the goal may be immediately achieved by the effects of the agent. However, in the complex, realistic domains this thesis targets, the intent of the action is not immediately achieved. Rather, the action taken is only one in a complex sequence of actions taken by the agent, and often others. For example, passing a ball to a teammate involves one or more kicks to the ball. As the ball leaves the immediate surroundings of the kicking agent, the pass is not yet complete. Instead, it takes a while for the ball to move through the field, sometimes in the vicinity of other players, until it is received by the teammate, if the pass is successful.

A glimpse into the future world state can provide important information about the agent's intent at the time an action was taken. Knowledge of what goals or future states to look for comes from the domain expert. Assuming the domain expert provides the learning system with a list of potential goals and rules for identifying them, it is possible to identify and tag each action based on the ultimate purpose it leads to.

This is done by searching forward in time in the log, identifying which goal was achieved, and then tagging the action as contributing to the goal. For example, in *RoboCup2D*, knowing that the ball reached a team member after an agent's kick tags the specific kick action with the feature *pass*. If instead a goal was scored, the kick would be tagged *kick to goal*. We tested our feature *fast forward* prediction technique on the tackle and the kick action. We did not test it on the other actions due to the lack of meaningful future states caused by those actions.

In Figure 6-1 on page 56 the *FastForwardPerfect* value is associated with the manual clustering of the *fast forward* feature based on the environment expert's prior knowledge; once that is done we ran the supervised learning using regression over the actions' parameters(*Step 2* from Section 5.1). The *FastForwardAUTO* value is associated with automatically clustering of the *fast forward* feature, using the chosen classification algorithm to infer the cluster. We subsequently used this auto cluster as a feature in the supervised learning of the actions' parameters (*Step 5* from

Section 5.1).

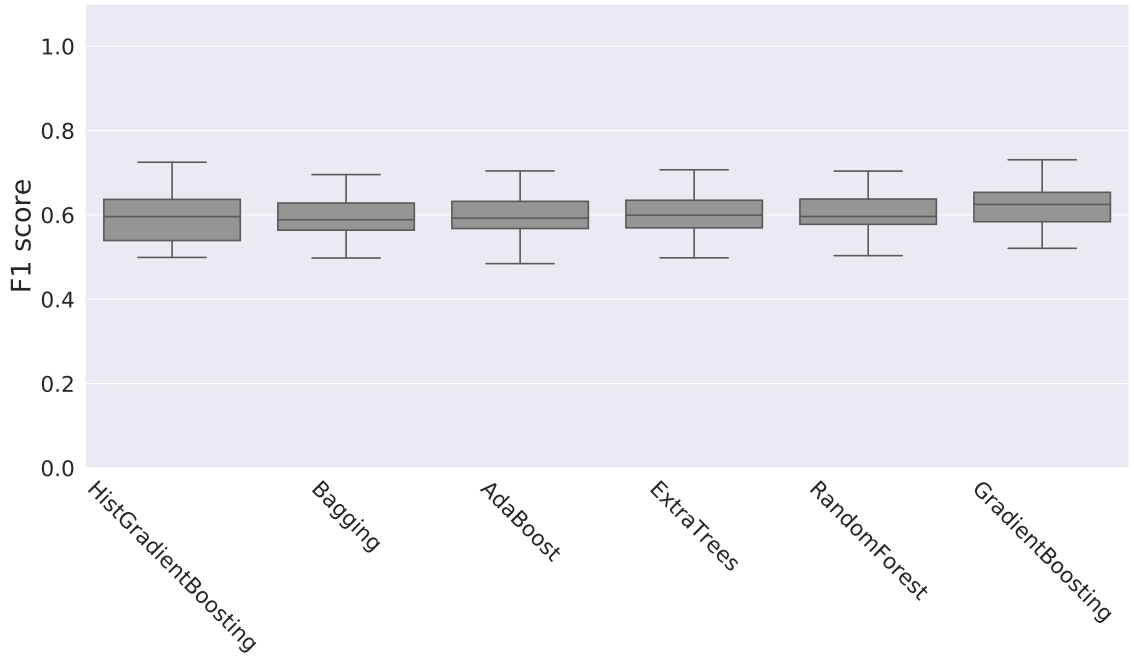
Kick Action We tested this technique over the same data presented in Section 3.3, in an attempt to improve the regression of the kick action's parameters. Initially, we received guidance from the system expert regarding significant results that were the result of the kick action; these helped us understand the agent's goal state (G_t^i) for example, a goal could be dribble, kick to the goal or pass the ball. We manually clustered each kick action based on prior knowledge over G_t^i.

To automate the clustering process of the G_t^i, we looked for an off-the-shelf algorithm to automate classification of the manual cluster as described in Section 5.1. The classification results are shown in Figure 5-2 on page 44. As can be observed, the *RandomForestClassifier* earned the second-best results concerning all metrics and target clusters. We therefore chose to use it instead of the *GradientBoostClassifier* for the same algorithm we used in the tackle action. We also found the *RandomForestClassifier* to be faster than the *GradientBoostClassifier* in classifying new samples, we achieved better results than what was presented in Section 3.3.

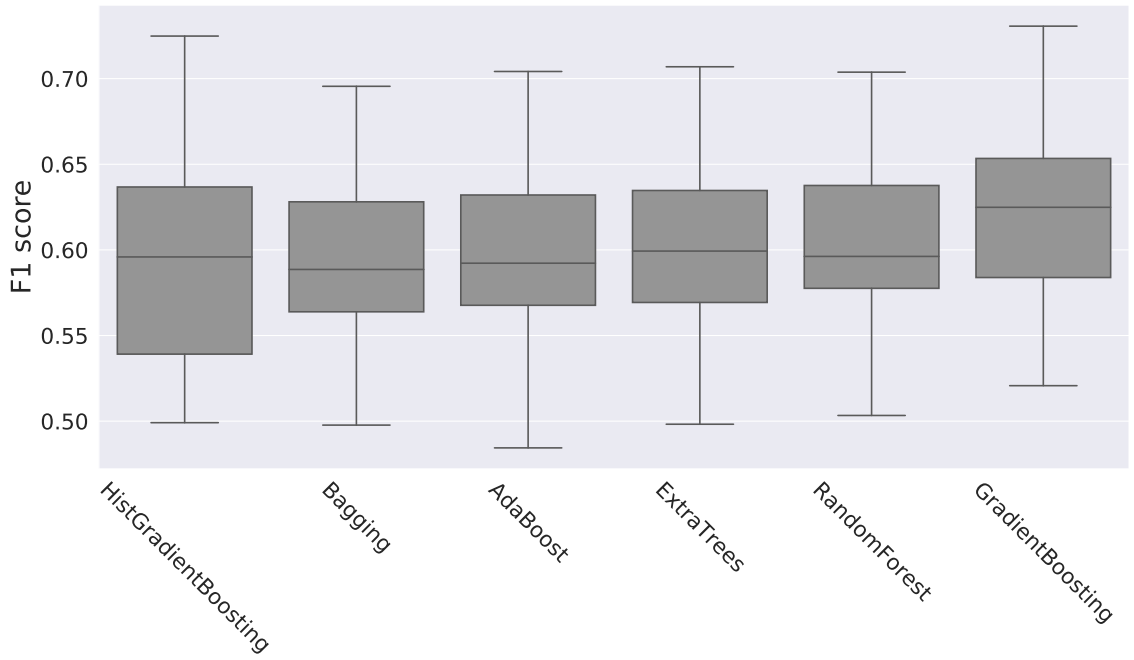
We also examined the option of dividing the samples according to the *FastForwardAUTO* values and then performed regression separately for each group of examples with the same *FastForwardAUTO* label. As can be seen in Figure 5-3 on page 47, there was no significant improvement obtained from splitting up the samples and then performing the regression as opposed to using *FastForwardAUTO* as a feature to perform the regression over all the examples together.

The final regression results (6-1), show that the *fast forward* technique improves the results around all measurements.

The results from this procedure to the tackle action are shown in Appendix C.

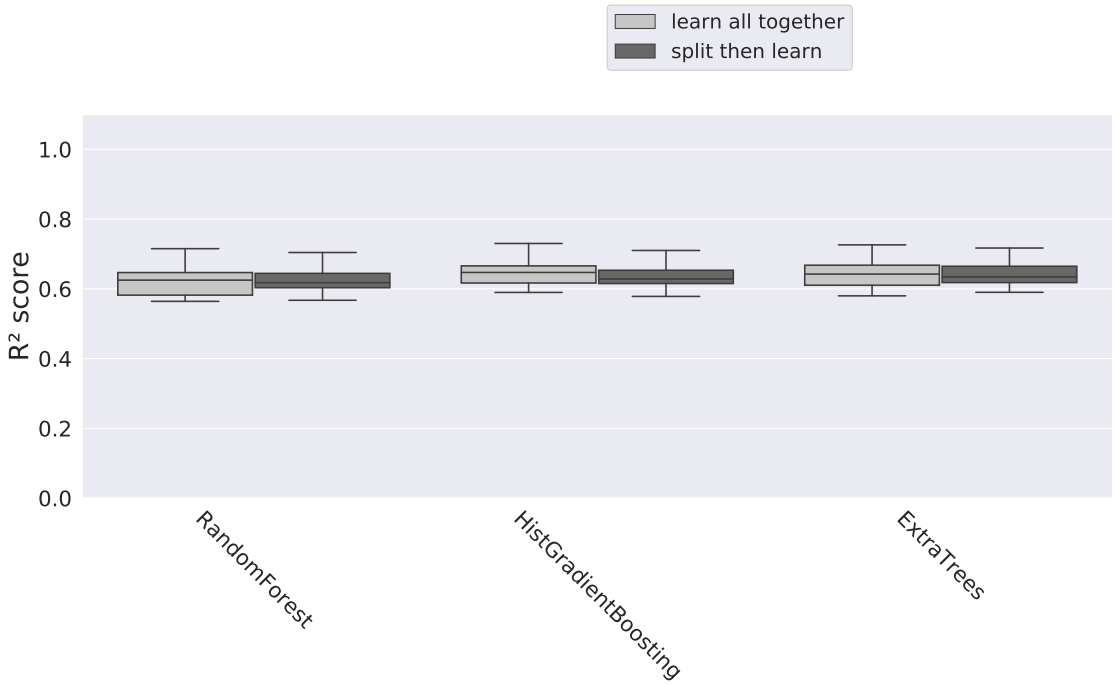


(a) F1 score range is from 0 to 1

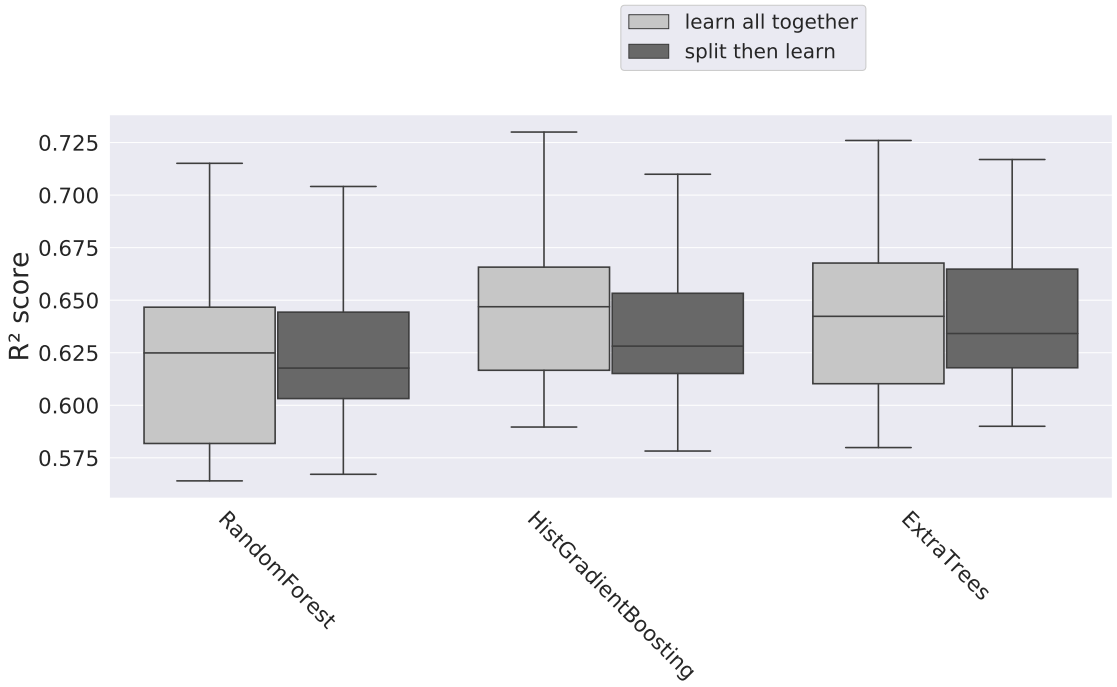


(b) F1 score range is from 0.5 to 0.8 zoom into Figure 5-2a

Figure 5-2: Kick FFP automatic classification step, higher results are better

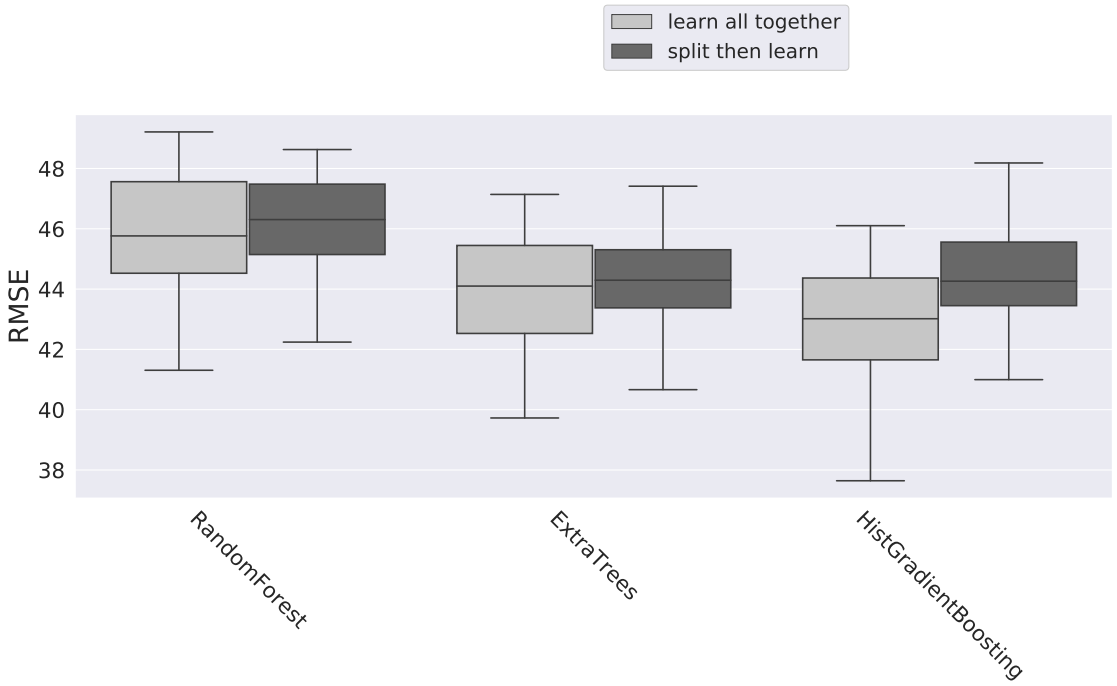


(a) R^2 score, higher results are better. Range 0 to 1.1

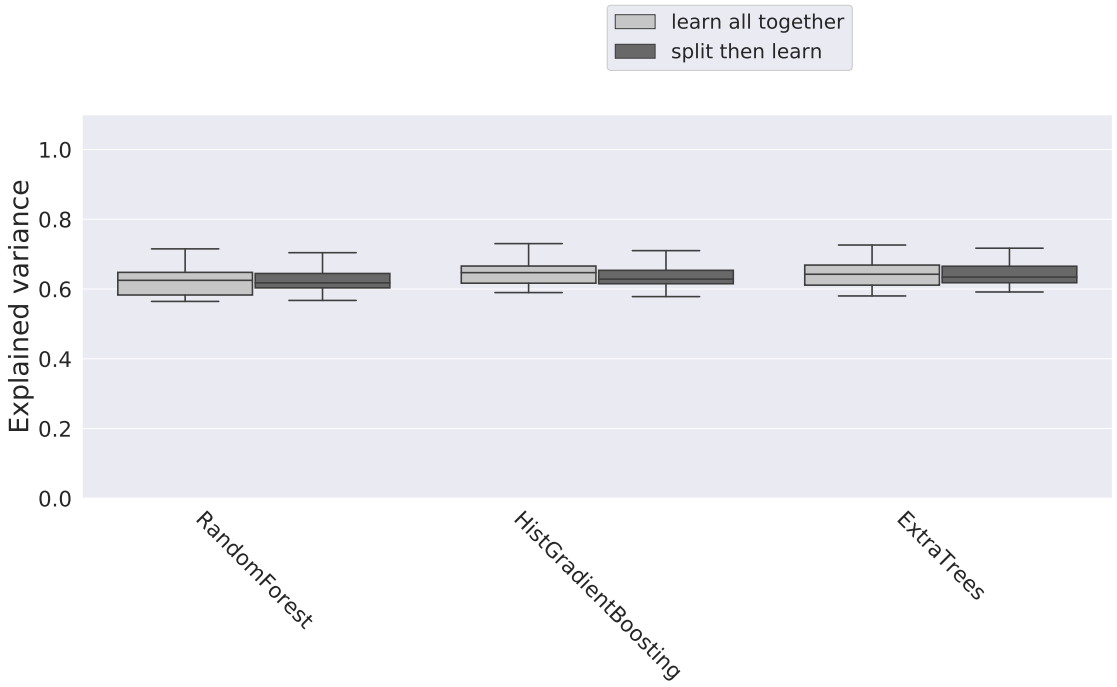


(b) R^2 score, higher results are better. Range 0.5 to 0.8 zoom into Figure 5-3a

Figure 5-3: Kick regression over P1 and P2 using FastForwardAUTO

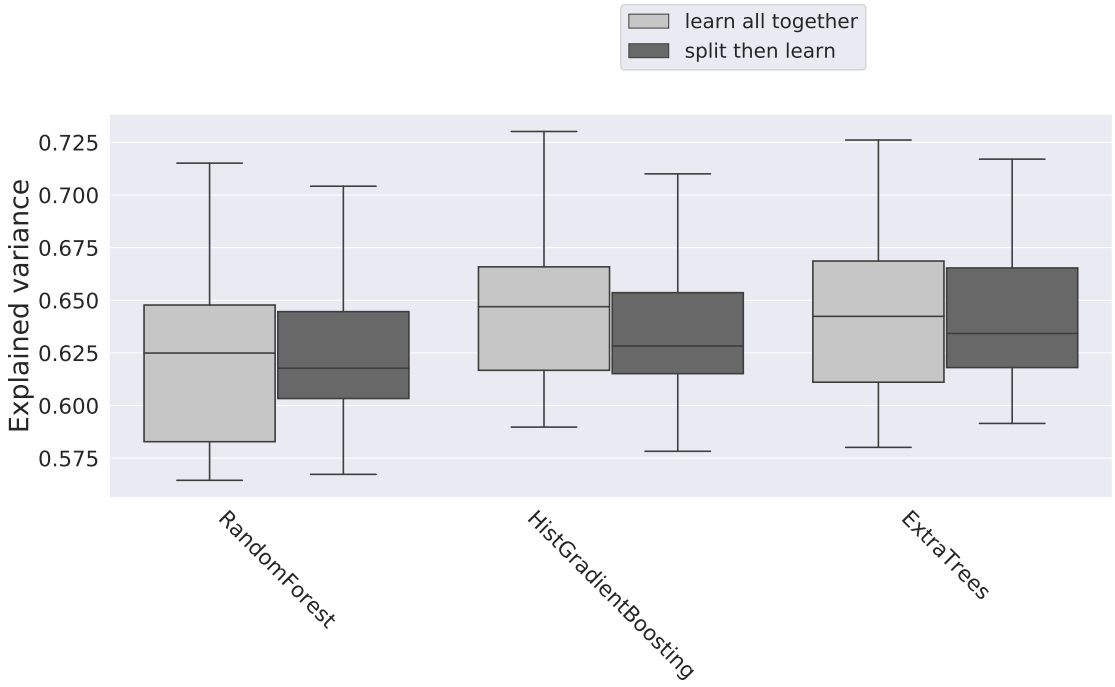


(c) RMSE, lower results are better.



(d) Explained variance, higher results are better. Range 0 to 1.1

Figure 5-3: Kick regression over P1 and P2 using FastForwardAUTO



(e) Explained variance, higher results are better. Range 0.5 to 0.8 zoom into Figure 5-3d

Figure 5-3: Kick regression over P1 and P2 using FastForwardAUTO

5.3 Parameter Clustering

Another complementary technique for identifying an agent's goals uses human expertise in a different way. Whereas the fast-forward method relied on domain expert knowledge to identify goals, *parameter clustering* relies on human visual processing. The idea is to visualize each action's parameter value distribution, using a histogram for single-parameter actions, a scatter plot for two-parameter actions, or some other visualization method. The human expert then identifies clusters within the visualization, based on the assumption that such clusters indicate some internal goal states for the agent.

In *Step 1* from Section 5.1, we plotted the parameters for each action as described above. We allowed the human expert to identify approximate clusters in the plot. Next, we applied clustering algorithms to the data, until we reached an algorithm and associated parameters that matched the human visualization. This allowed us to tag each example with the identification of the cluster to which it belong which add goal state G_t^i as a feature of the agent's egocentric state S_t^i in addition to beliefs state B_t^i. We tested our feature for the *parameters cluster* technique over all five actions mentioned in Table 3.1 on page 19.

In Figure 6-1 on page 56 the *ClusterPerfect* value is associated with the manual clustering of the *parameter clustering* feature. We used an off-the-shelf clustering algorithm, with manual fine-tuning of the hyper-parameters to get a cluster-based on the parameter distribution that provides knowledge for the agent's goal state (G_t^i) (*Step 2* from Section 5.1). Then we ran supervised learning using regression over the actions' parameters. The *ClusterAUTO* value is associated with automatically clustering the *parameter clustering* feature. We used the chosen classification algorithm to infer the cluster and subsequently used this auto cluster as a feature in the supervised learning of the actions' parameters (*Step 5* from Section 5.1).

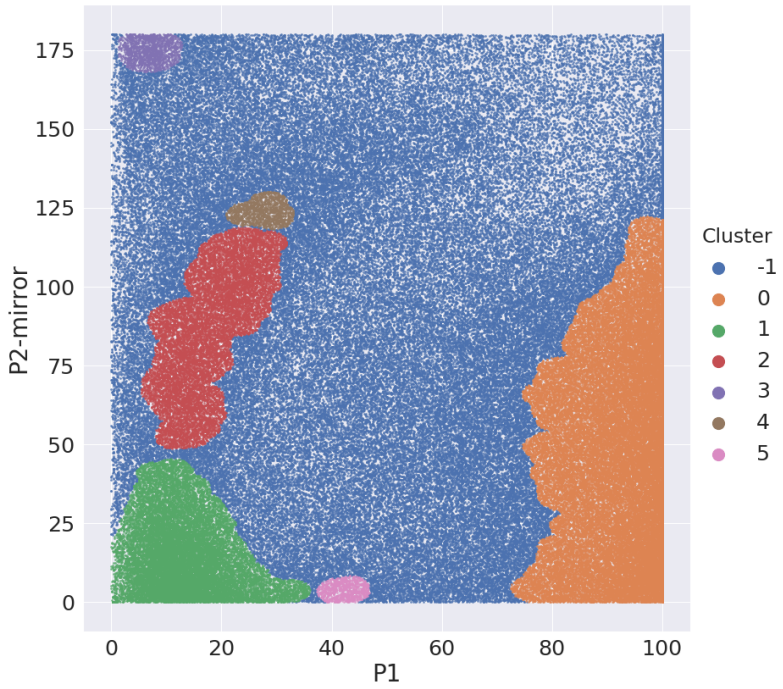
Kick Action We tested the parameter cluster technique over the same data presented in Section 5.2 in an attempt to improve the regression of the kick action's parameters. We clustered each kick action using off-the-shelf clustering algorithms such as:

MiniBatchKMeans, AffinityPropagation, MeanShift, Ward, AgglomerativeClustering, DBSCAN, Hdbscan, Optics, Birch, or GaussianMixture for more details over the chosen parameters see Appendix B. The results were plotted and provided to the human expert for manual visual confirmation. Some of the better plots are shown in Figure 5-4 on page 50. Note that the clusetering algorithms were applied it to the data *after* the mirroring transformation (here, over P2).

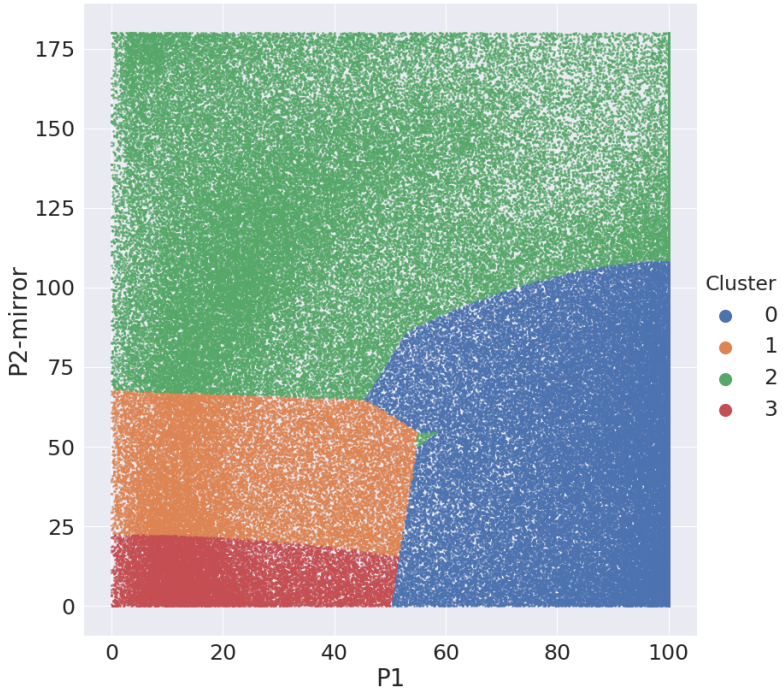
While the human expert chose *Hdbscan* [49] (5-4d), we ran the regression over P1 and P2, each time adding one of the clusters as a feature, to find the best *parameter clustering* cluster. The results are shown in Figure 5-5 on page 52. As can be seen, the *GaussianMixture* shown in Figure 5-4b presents the best results for almost all measurements, while the *Hdbscan* (5-4d) results did not. The human expert visualization is important to narrow down the list of candidate algorithms, but automated learning is still needed to determine the best technique.

Having selected the clustering algorithm, we can now use the cluster labels to learn a classifier—one that will learn to predict the cluster label as a feature of the data, and thus assist in the action parameter regression. The results of the learned classifier, when applied to the original problem, are shown in Figure 5-6 on page 54. The figure shows that the *HistGradientBoostClassifier* provides the best results concerning all metrics and target clusters. The results from this procedure for other actions are shown in Appendix D.

The next chapter shows the results for aggregating together all the techniques introduced in Part I of the thesis. It will show that *parameter clustering* improves the results for all measurements.

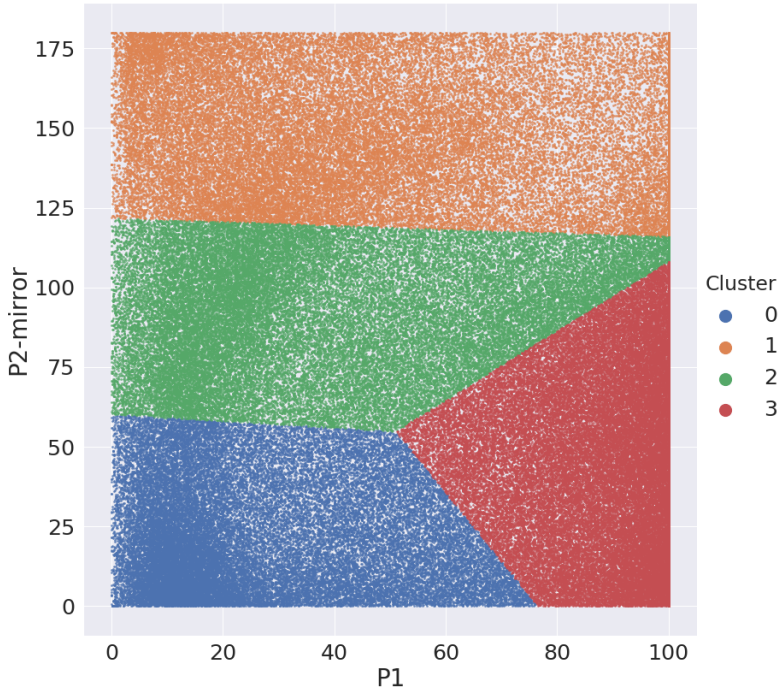


(a) DBSCAN

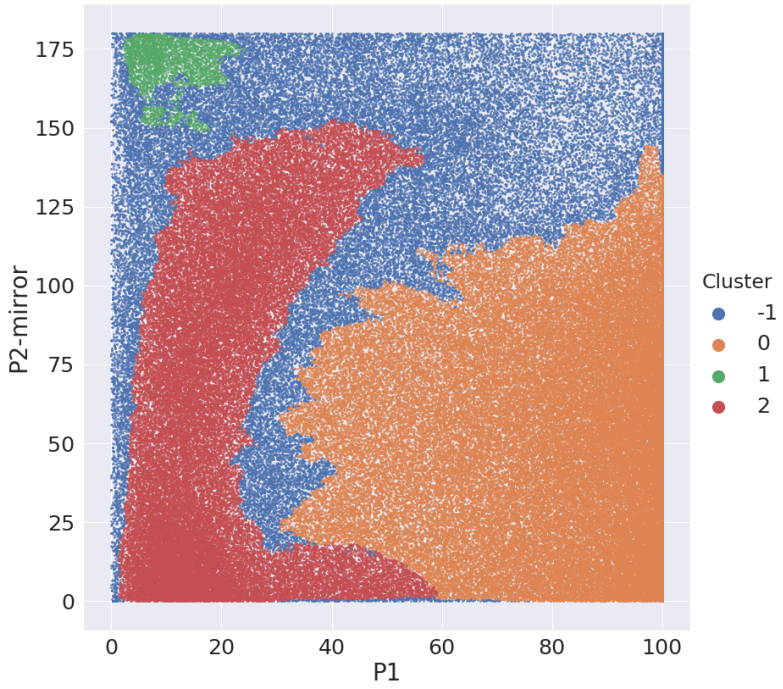


(b) GaussianMixture

Figure 5-4: Kick cluster algorithm plotting

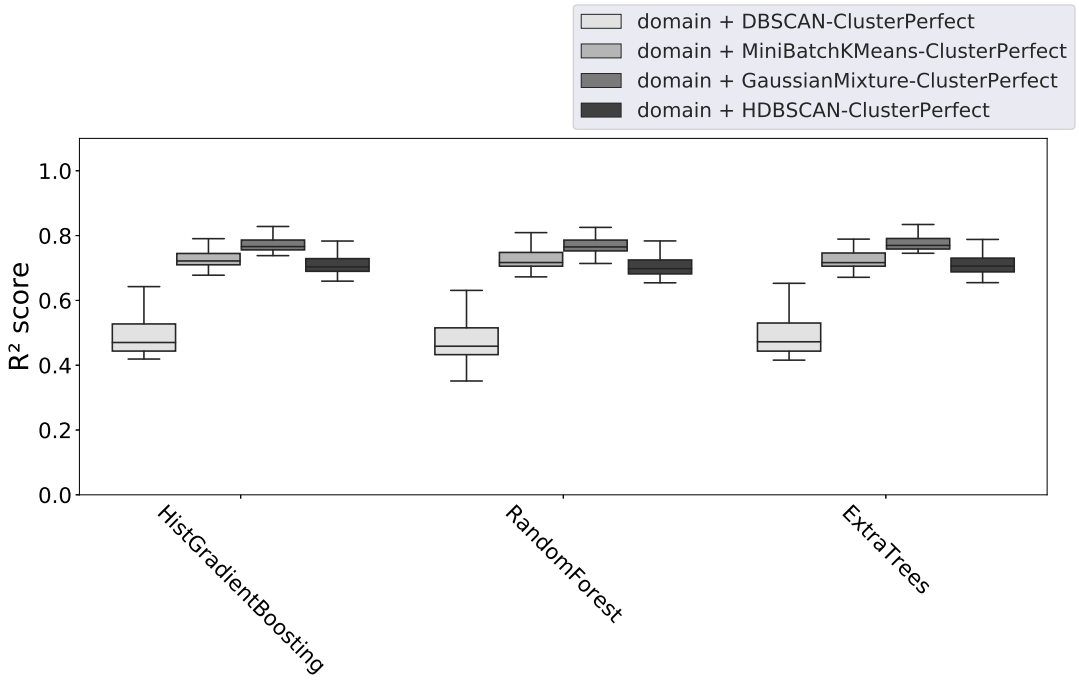


(c) MiniBatchKMean

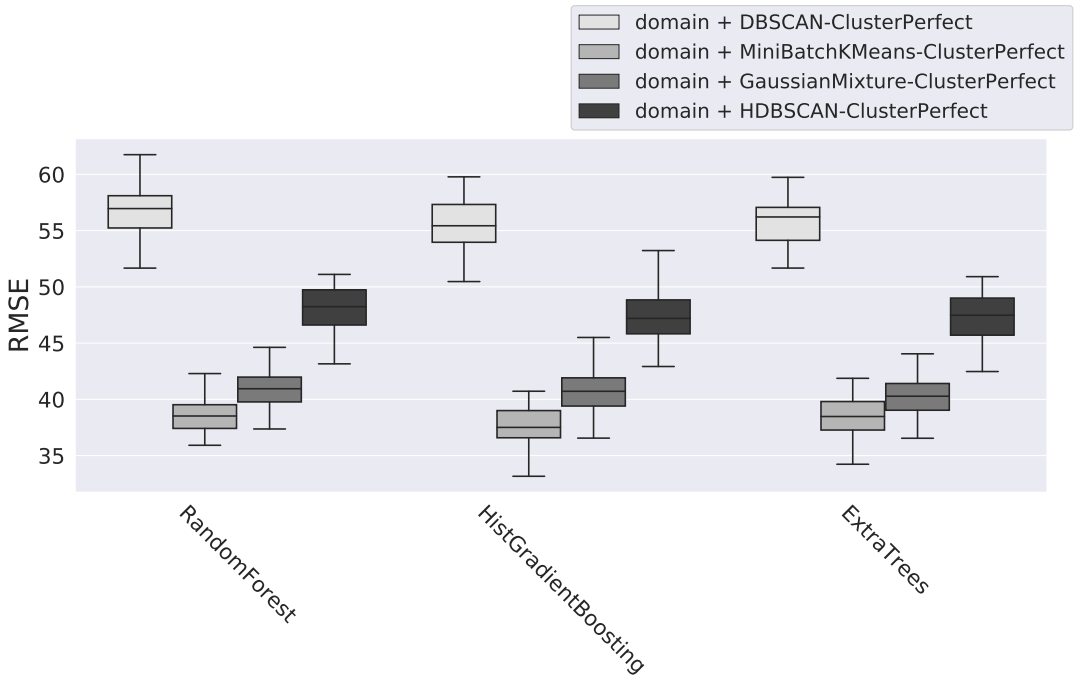


(d) Hdbscan

Figure 5-4: Kick cluster algorithm plotting

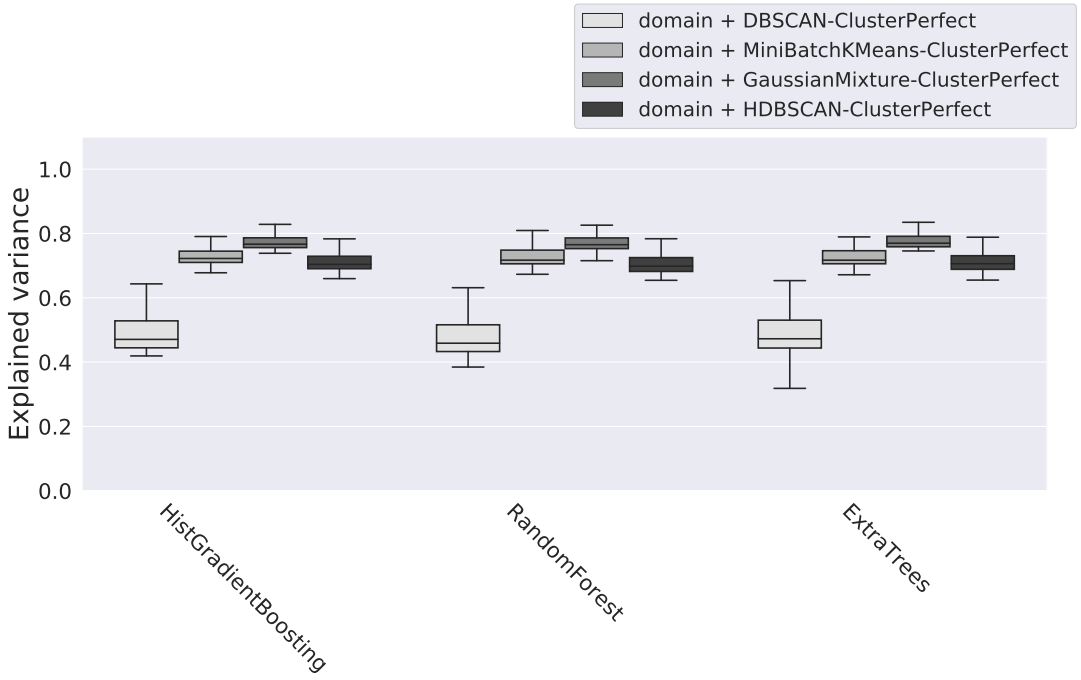


(a) R^2 score, higher results are better.



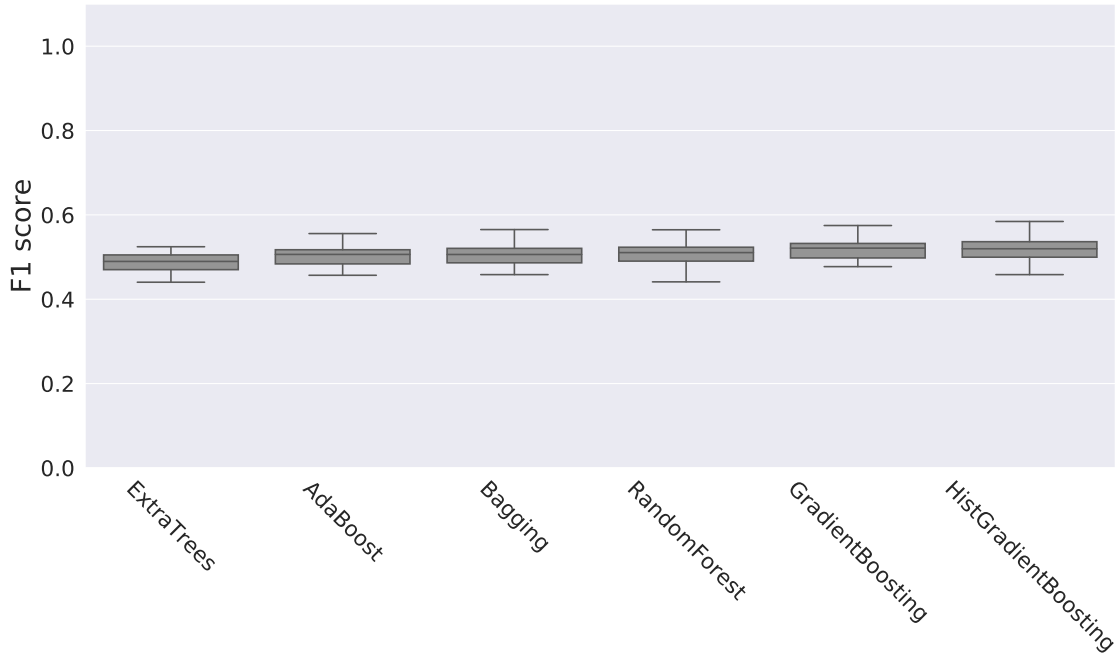
(b) RMSE, lower results are better.

Figure 5-5: Kick regression over P1 and P2, comparison of Cluster Perfect

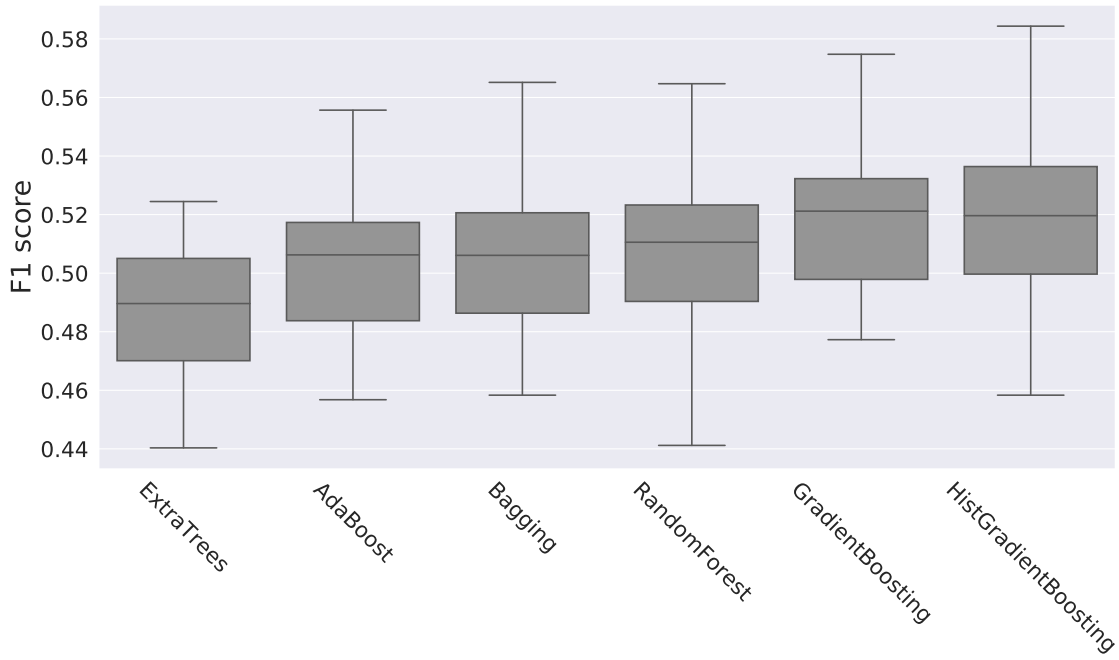


(c) Explained variance, higher results are better.

Figure 5-5: Kick regression over P1 and P2, comparison of Cluster Perfect



(a) F1 score range is from 0 to 1



(b) F1 score range is from 0.4 to 0.6 zoom into Figure 5-6a

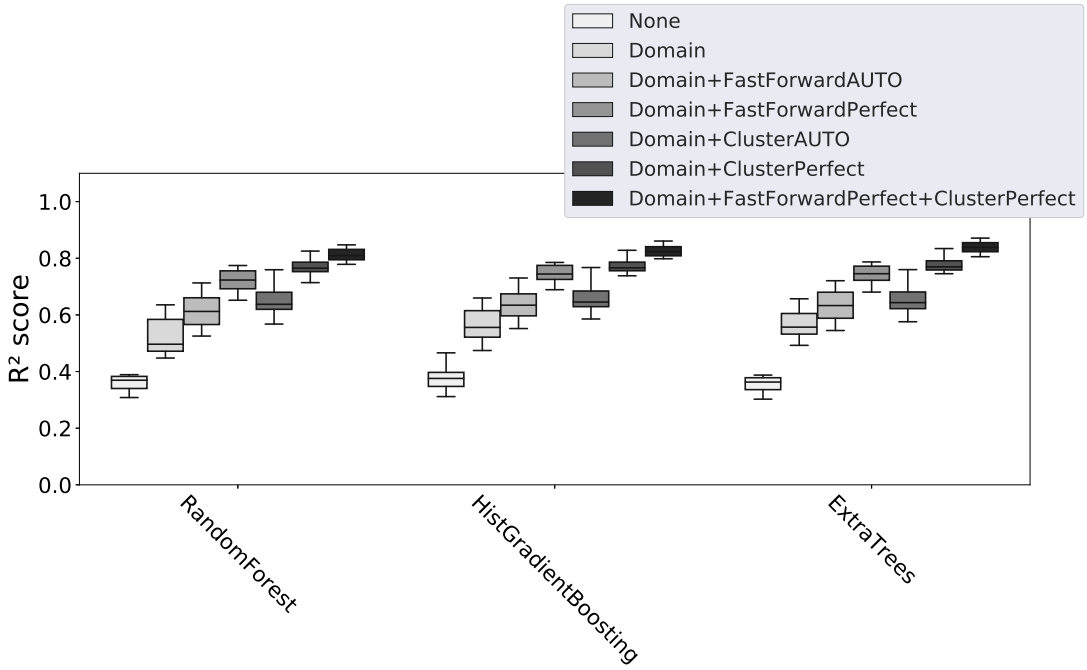
Figure 5-6: Kick distribution automatic classification step, higher results are better.

Chapter 6

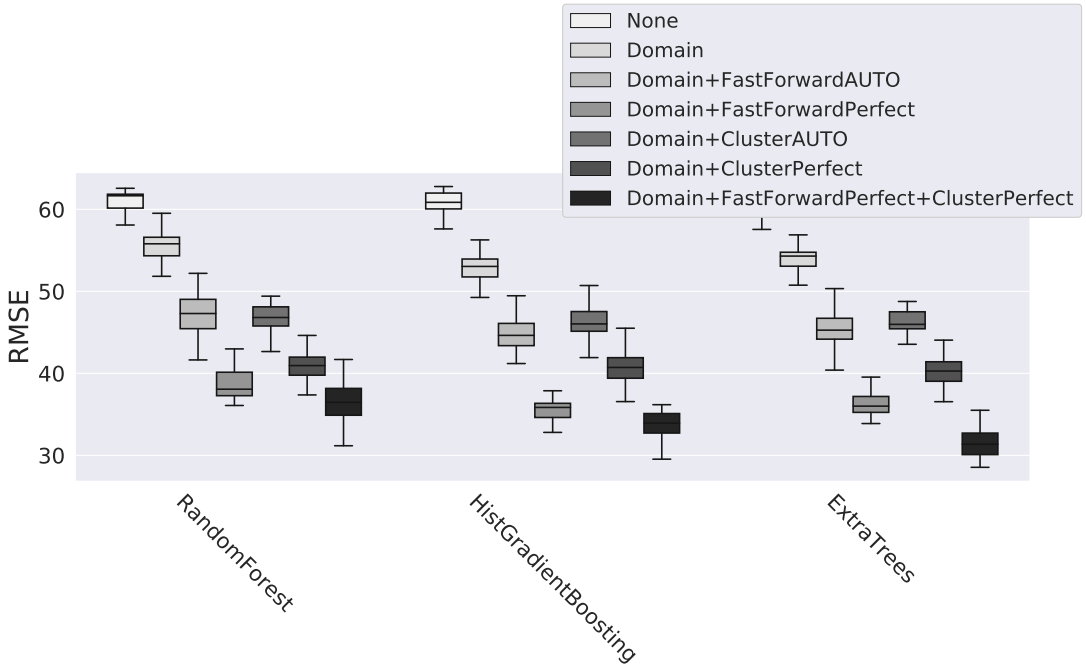
Experiments

This chapter reports on experiments aggregating all the techniques introduced in Part 1, compared to the initial results in Section 3.3. Our goal was to learn the parameters of each action as mentioned in Table 3.1. The results are shown for all actions.

Kick Action We performed regression on the kick parameters P1 and P2 together; the results are shown in Figure 6-1 on page 56. As can be observed, by adding the agent's domain-specific beliefs state (B_t^i) as features, we saw improved results across all metrics.

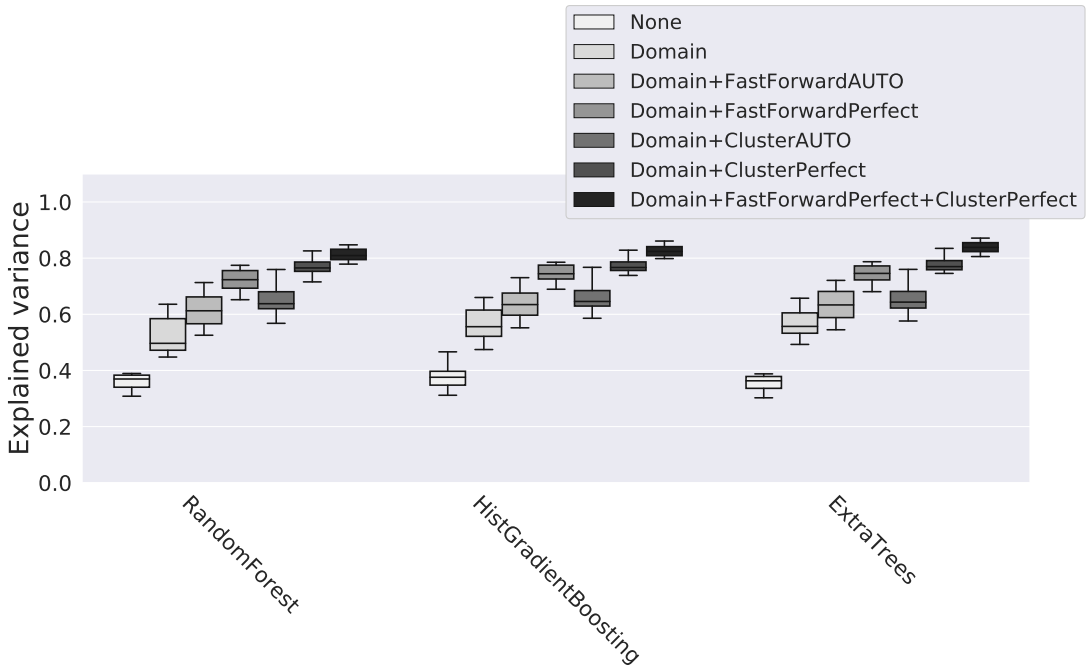


(a) R^2 score, higher results are better.



(b) RMSE, lower results are better.

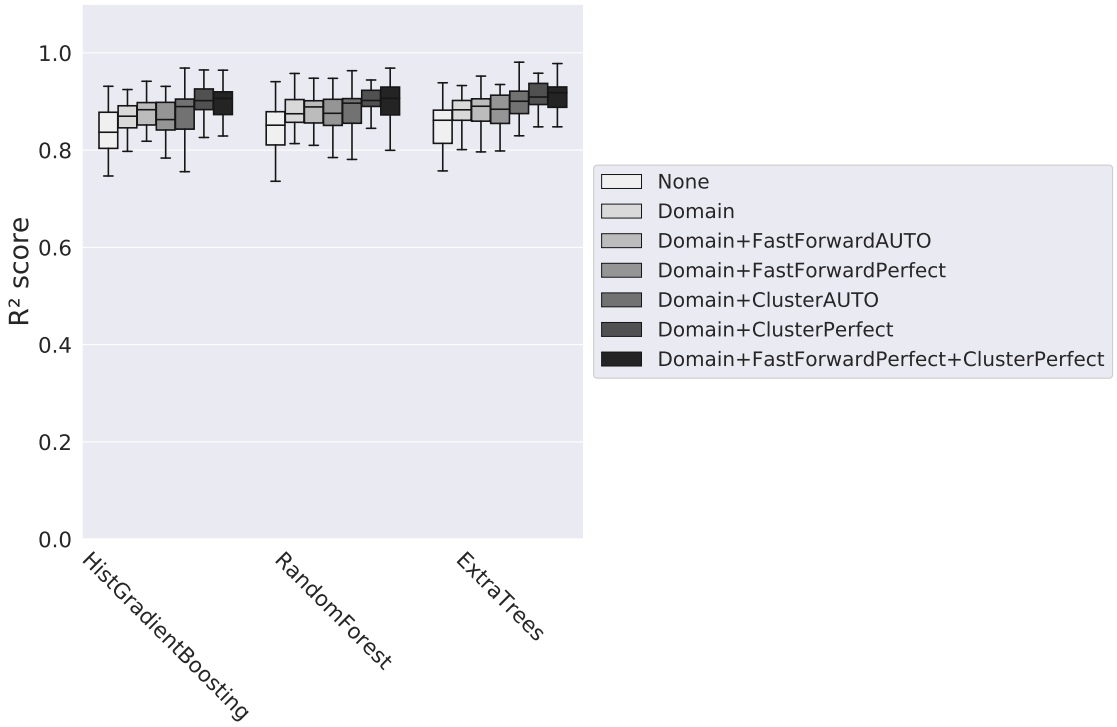
Figure 6-1: Kick final regression over P1 and P2



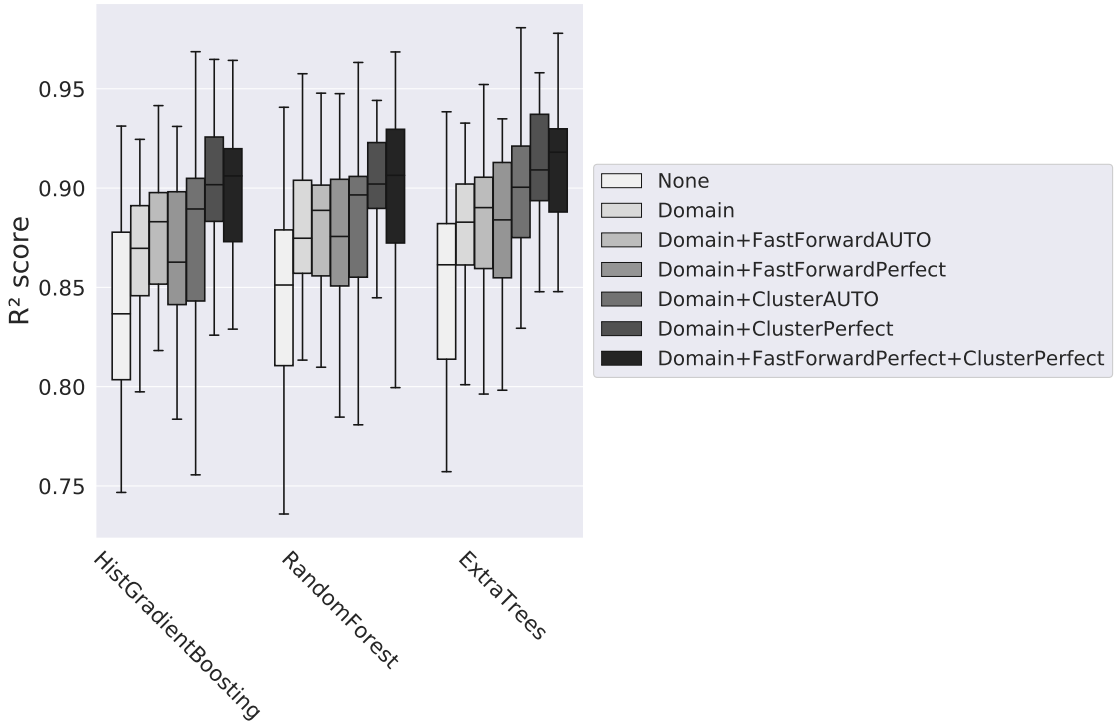
(c) Explained variance, higher results are better.

Figure 6-1: Kick final regression over P1 and P2

Tackle Action Since P1 is *float* (degree) and P2 is *Boolean*, even after plotting the histogram of P1 according to the value of P2, we chose to learn each parameter separately. We performed the regression over the P1 parameter and present the results in Figure 6-2 on page 61. By adding the agent's beliefs state (B_t^i) as features, we were able to observe a slight improvement in the results relative to those obtained for the kick action. We also classified the P2 parameter and present the results in Figure 6-3 on page 64. By adding the agent's beliefs state (B_t^i) as features, we noticed that the results improved across all metrics including recall and precision. It is also important to mention that the improvement of adding the agent's beliefs state (B_t^i) compared to other techniques, shows the most improvement.

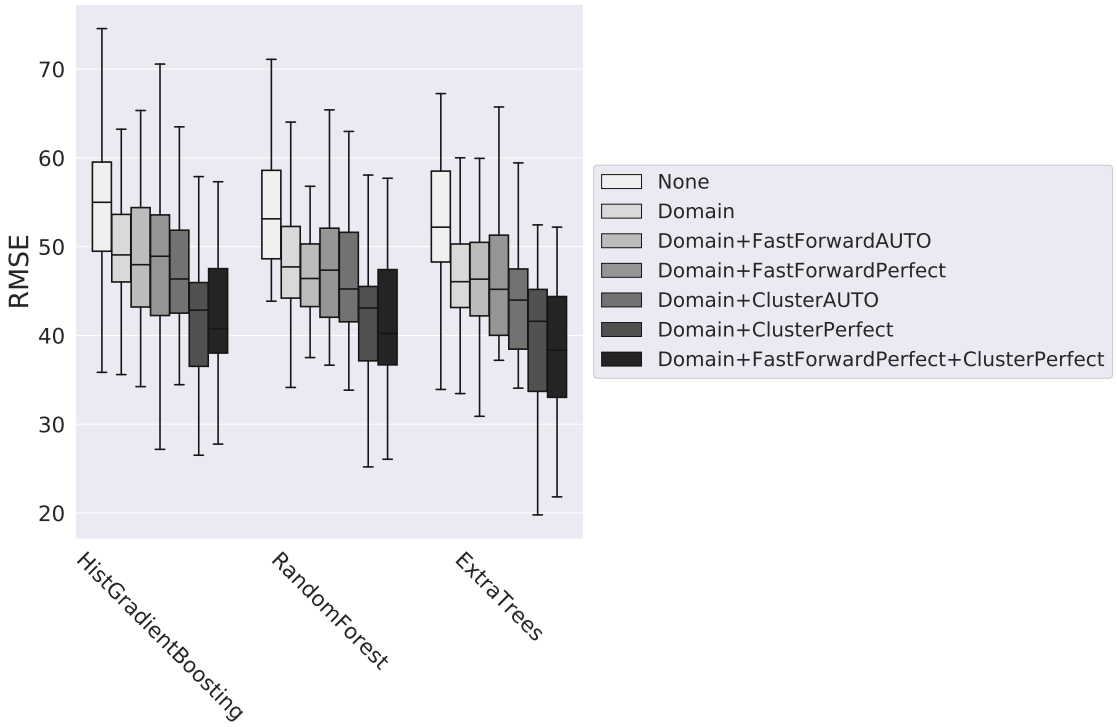


(a) R^2 score, higher results are better.

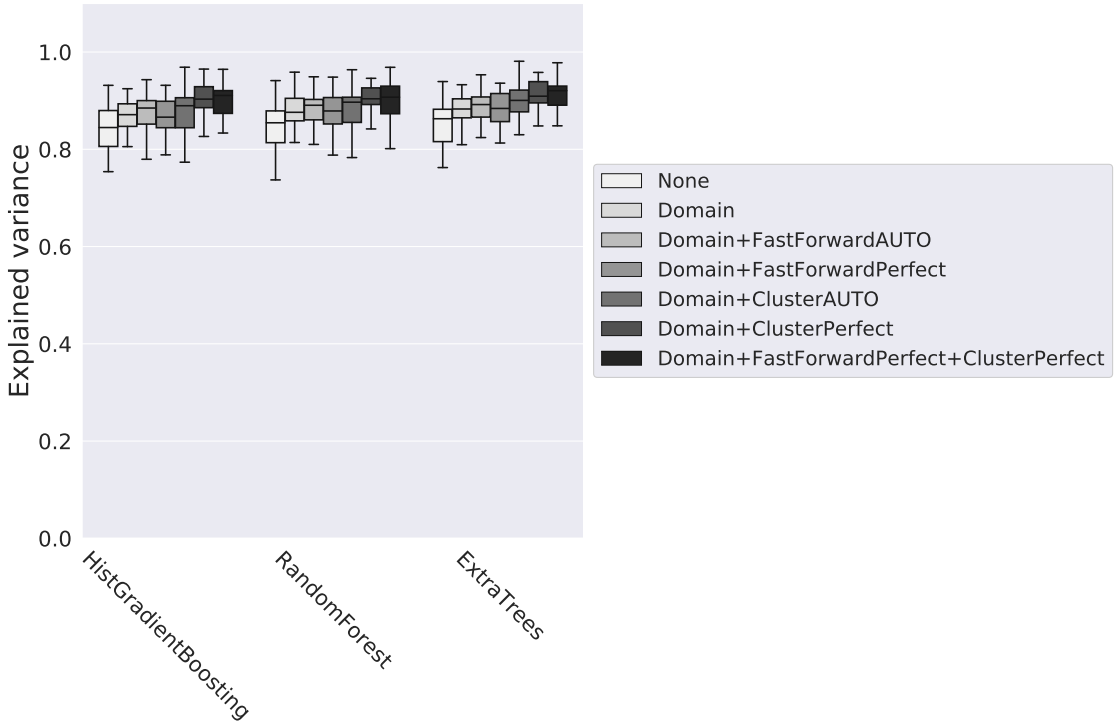


(b) R^2 score, higher results are better. Range is from 0.7 to 1, zoom over 6-2a

Figure 6-2: Tackle final regression over P1

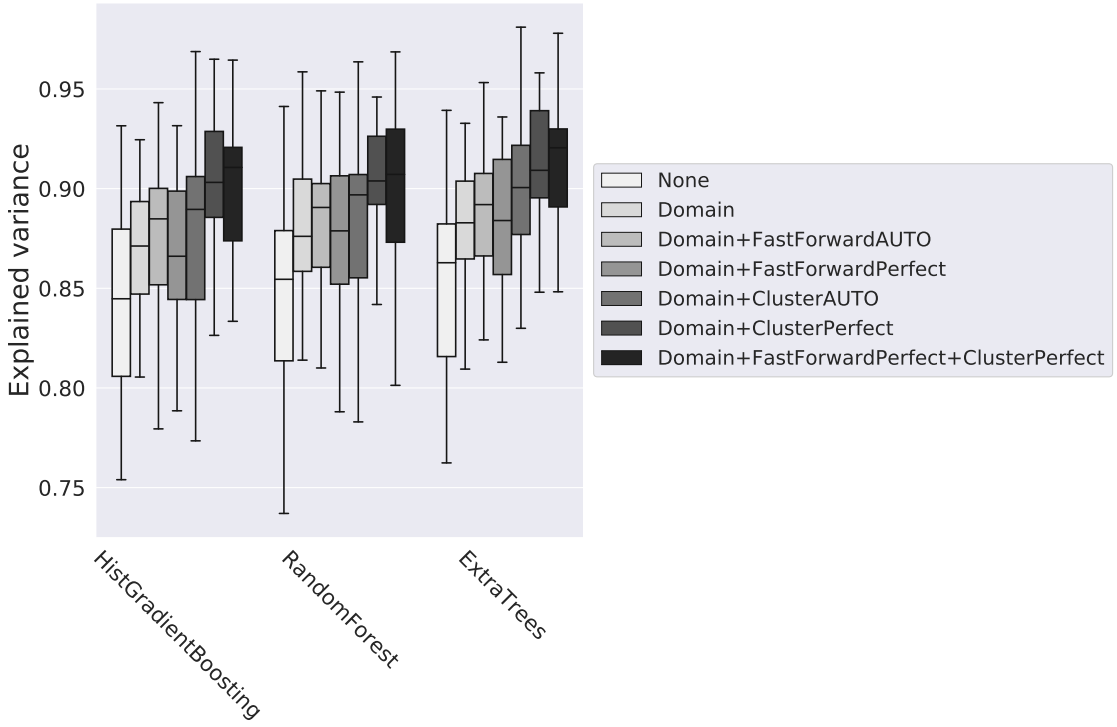


(c) RMSE, lower results are better.



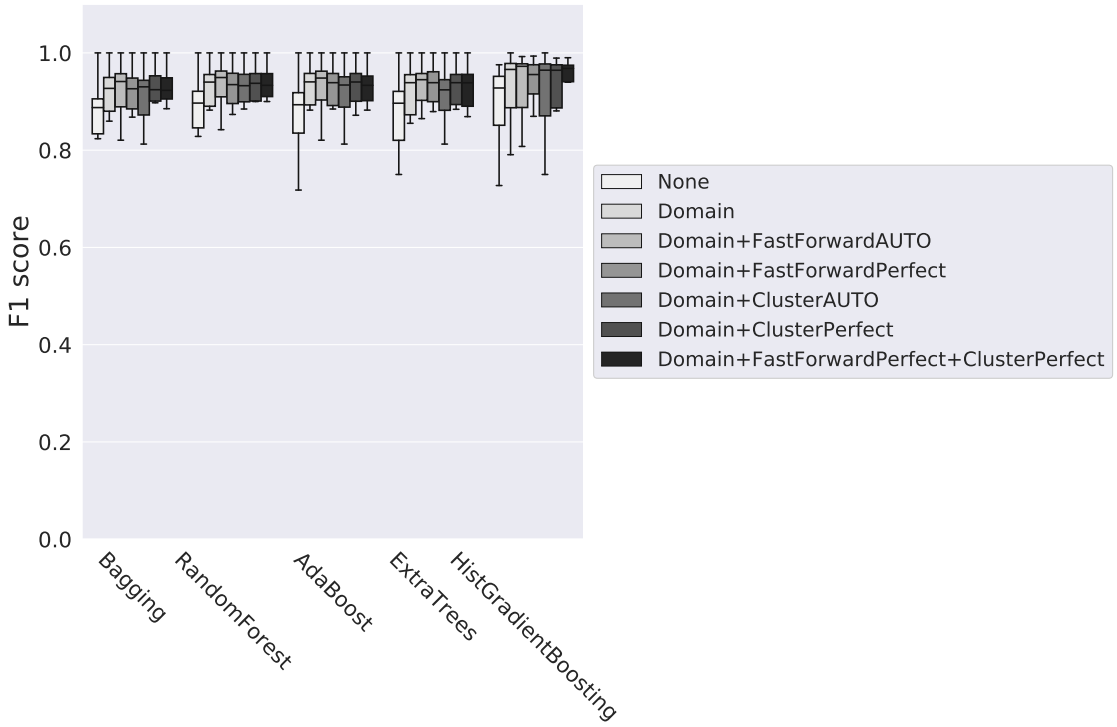
(d) Explained variance, higher results are better.

Figure 6-2: Tackle final regression over P1

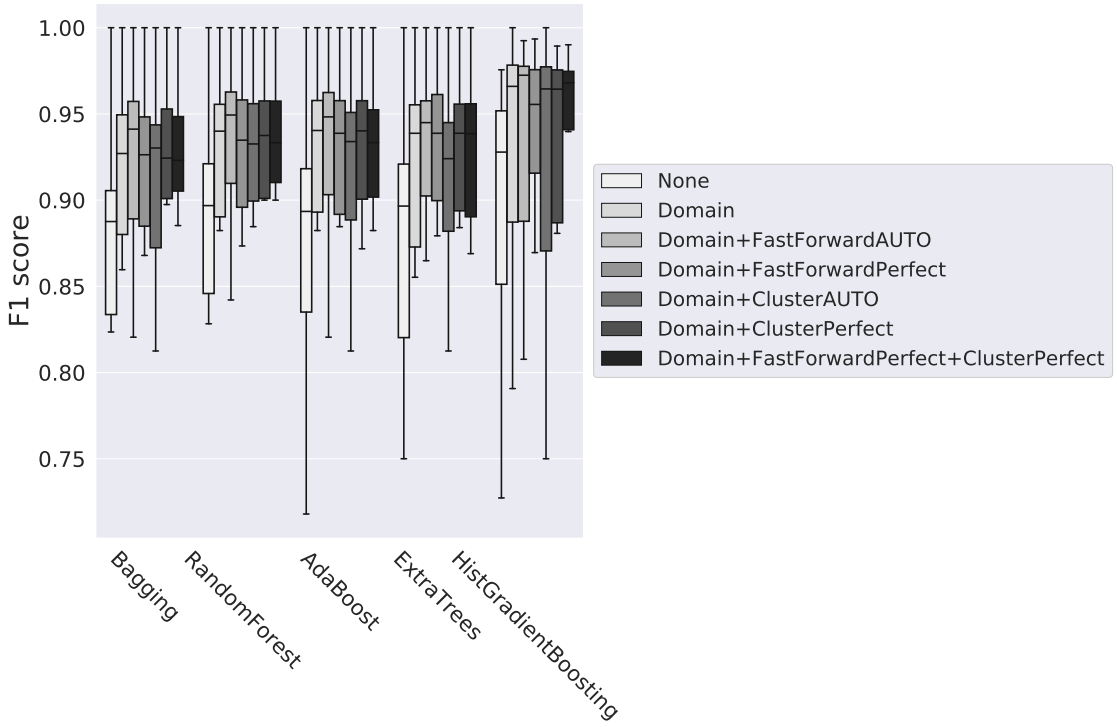


(e) Explained variance, higher results are better. Range is from 0.7 to 1, zoom over 6-2d

Figure 6-2: Tackle final regression over P1

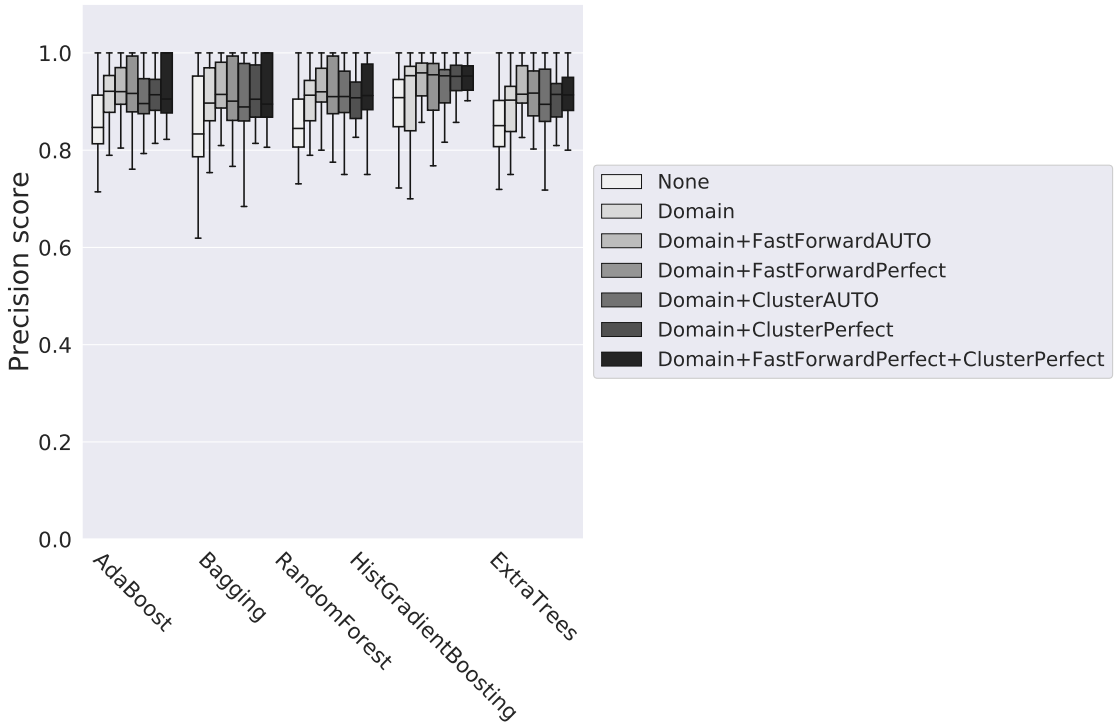


(a) F1 score, higher results are better.

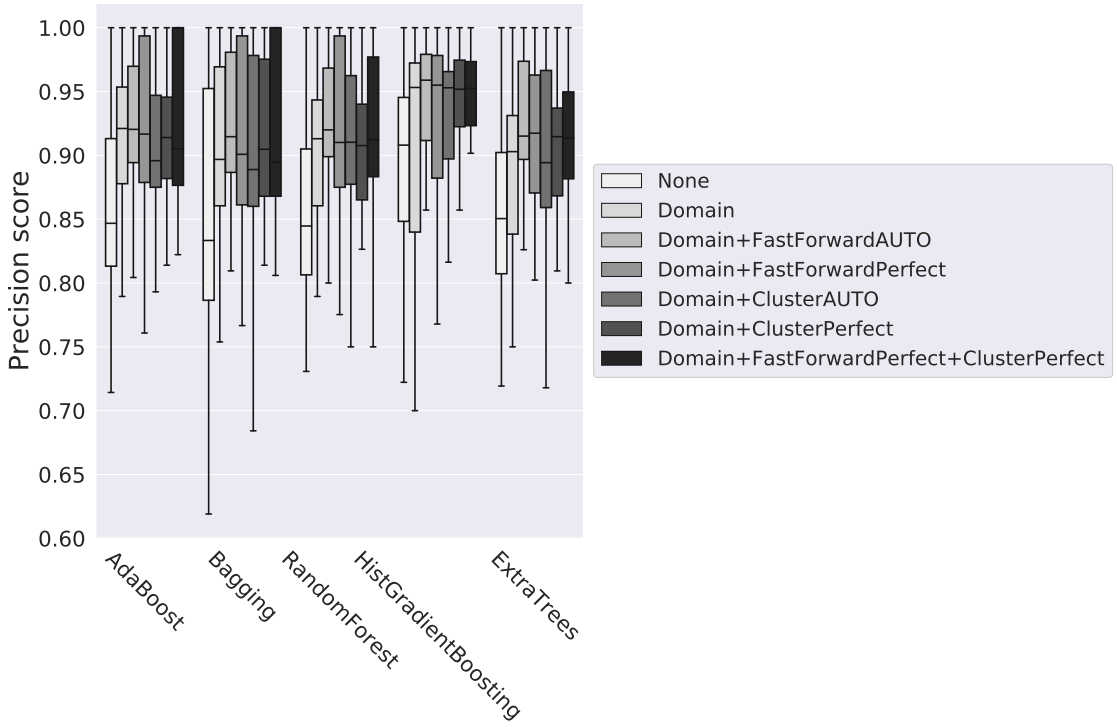


(b) F1 score, higher results are better. Range is from 0.6 to 1, zoom over Figure 6-3a

Figure 6-3: Tackle final regression over P2

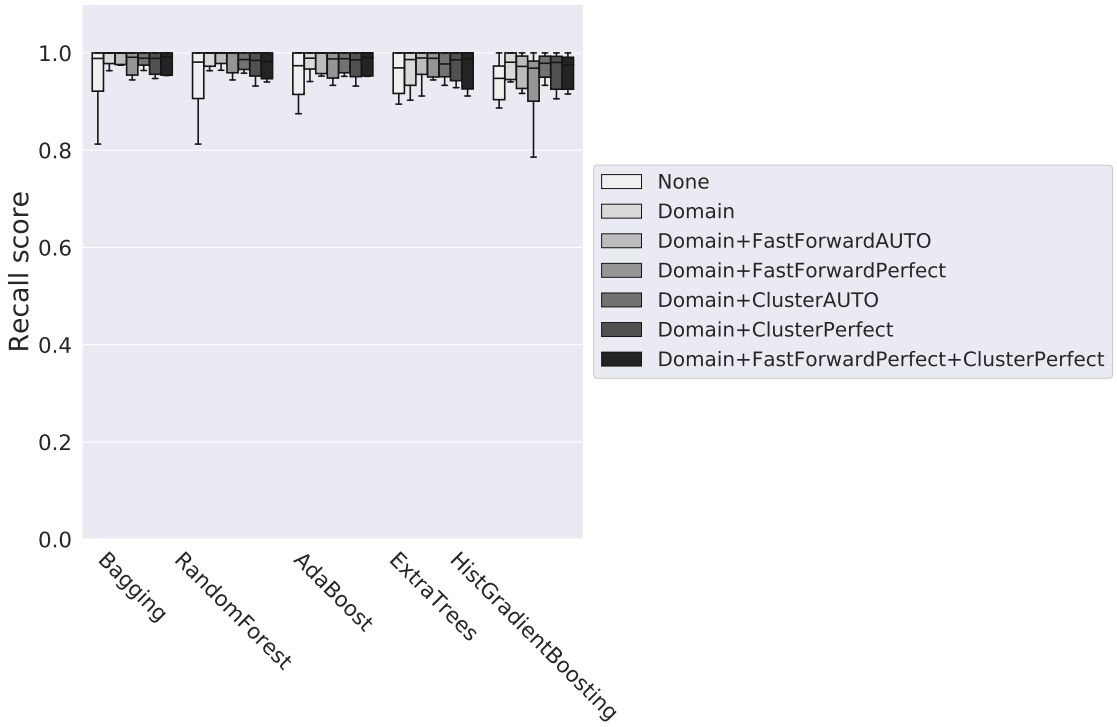


(c) Precision, higher results are better.

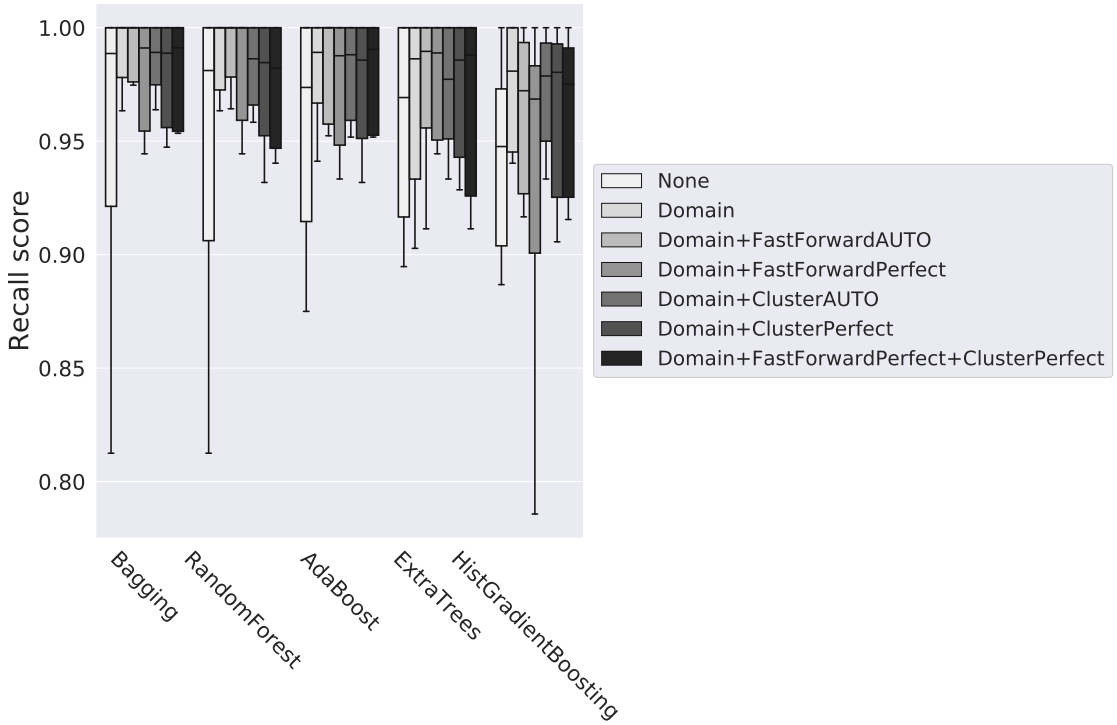


(d) Precision, higher results are better. Range is from 0.6 to 1, zoom over Figure 6-3c

Figure 6-3: Tackle final regression over P2



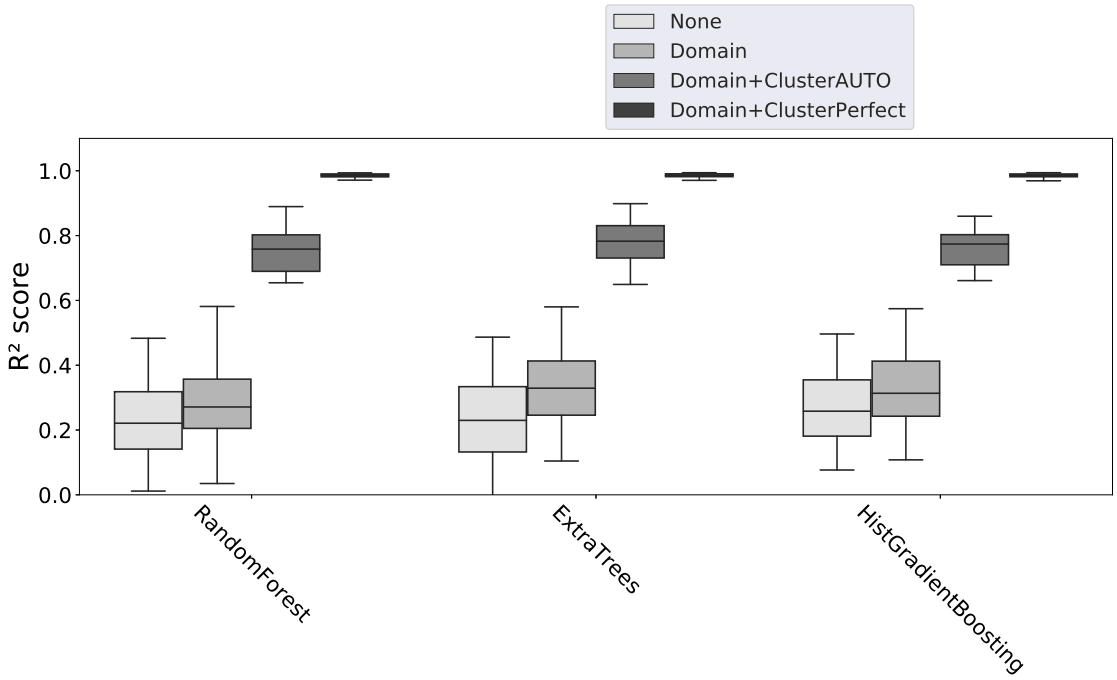
(e) Recall, higher results are better.



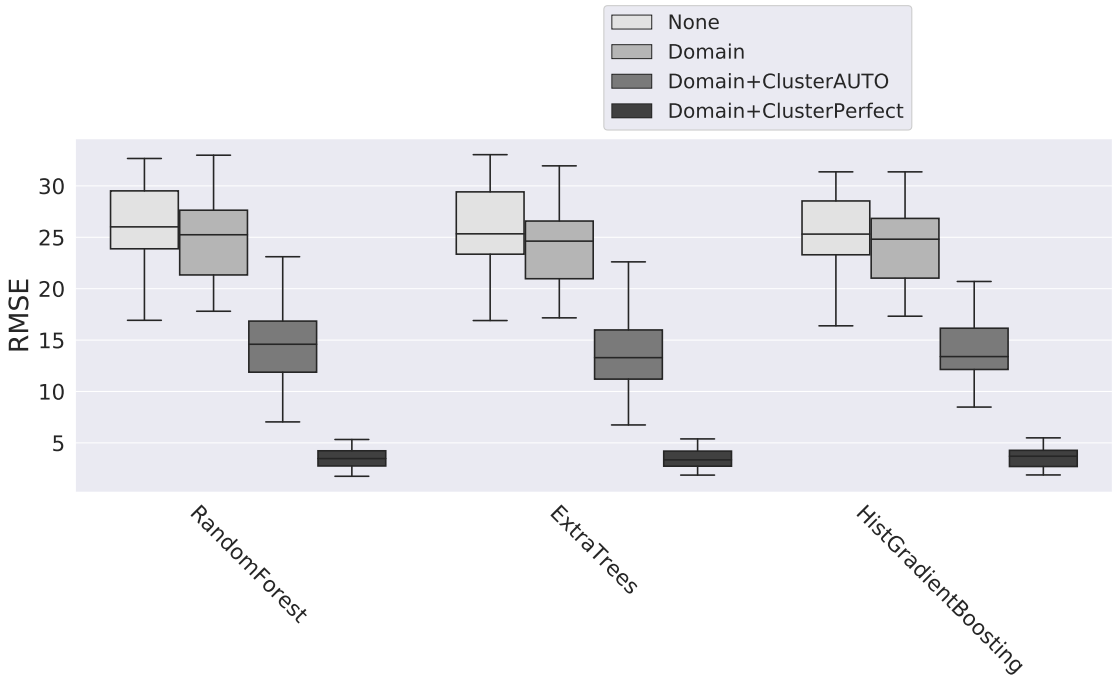
(f) Recall, higher results are better. Range is from 0.6 to 1, zoom over Figure 6-3e

Figure 6-3: Tackle final regression over P2

Dash Action We performed regression for the dash parameter P1 while ignoring P2 because P2 appears in only 2% of the dash actions; in the other actions, it appears as None. Since there are many dash actions, we randomly choose 6,000 actions for each team. The results are shown in Figure 6-4 on page 66. We noticed that by adding the agent's beliefs state (B_t^i) as features, we were able to improve the results slightly, relative to those for the kick action.

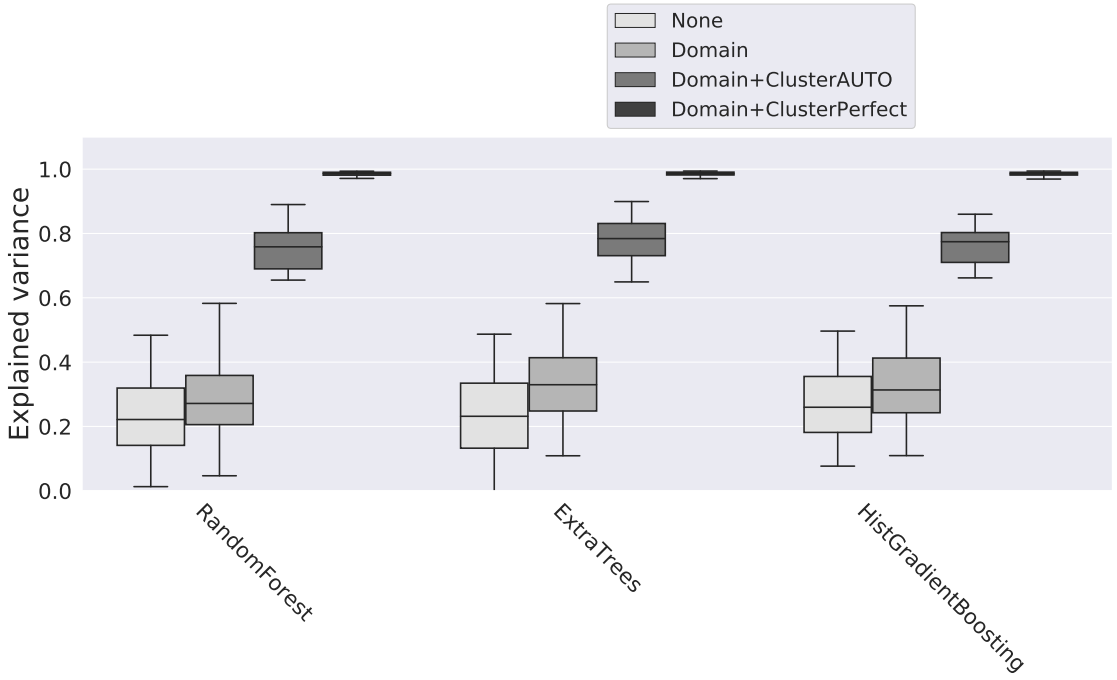


(a) R^2 score, higher results are better.



(b) RMSE, lower results are better.

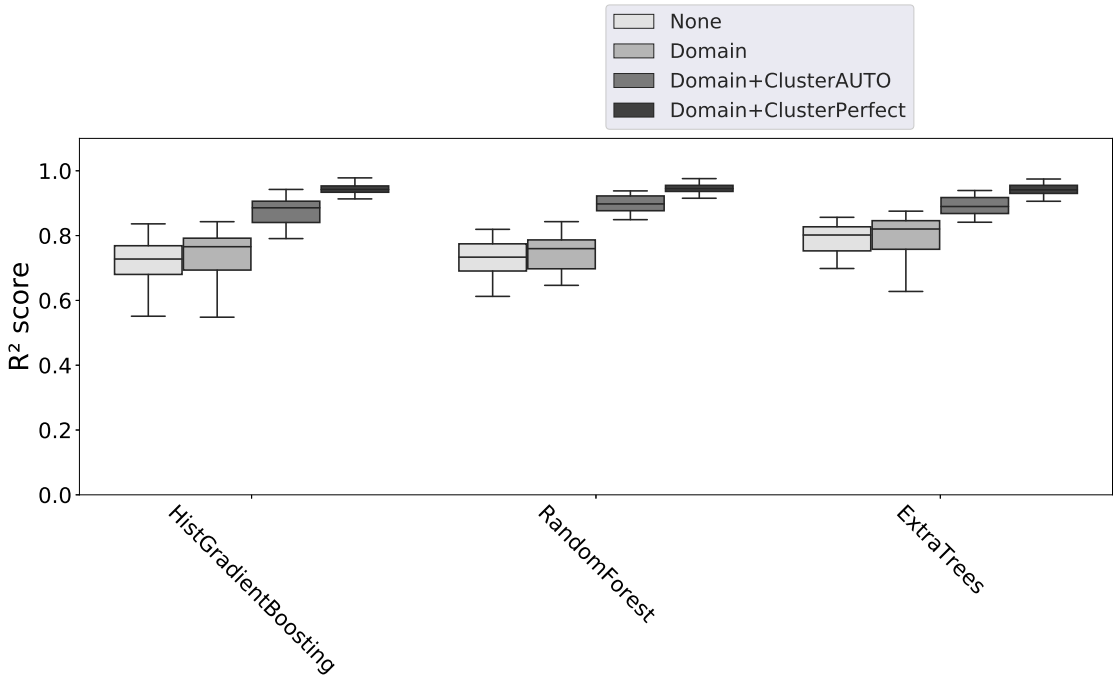
Figure 6-4: Dash final regression over P1



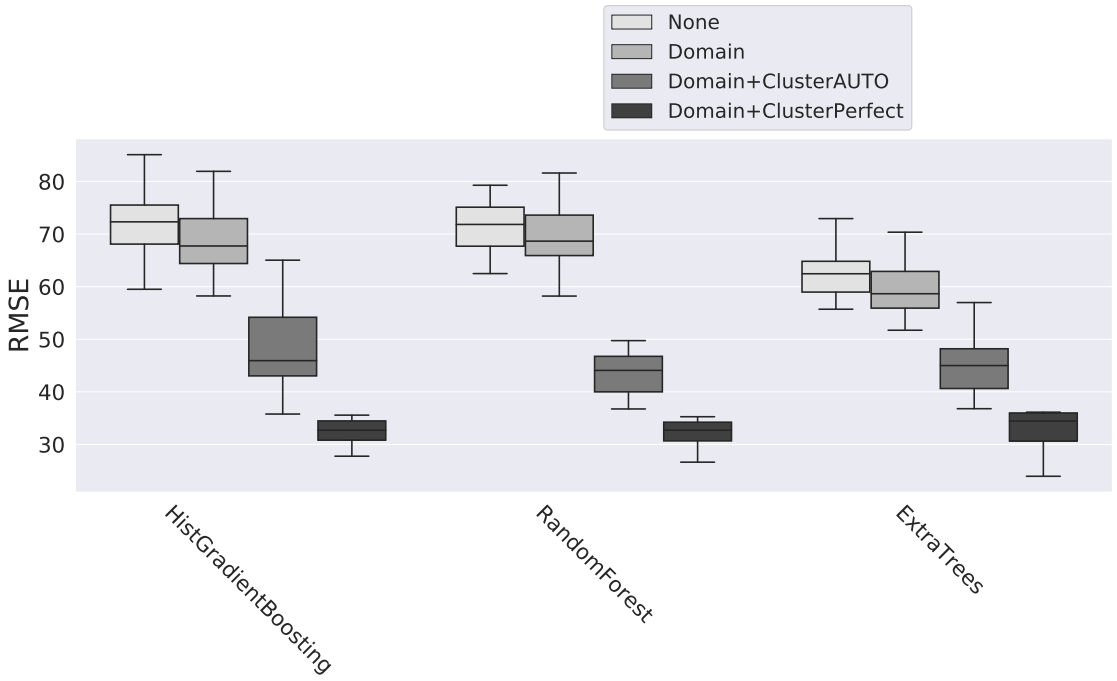
(c) Explained variance, higher results are better.

Figure 6-4: Dash final regression over P1

Turn Action We performed regression on the turn parameter P1. Since there are many turn actions, we randomly chose 6,000 actions per team. The results are shown in Figure 6-5 on page 69. We observed that by adding the agent's beliefs state (B_t^i) as features, we were able to improve the results slightly, relative to those of the kick action.

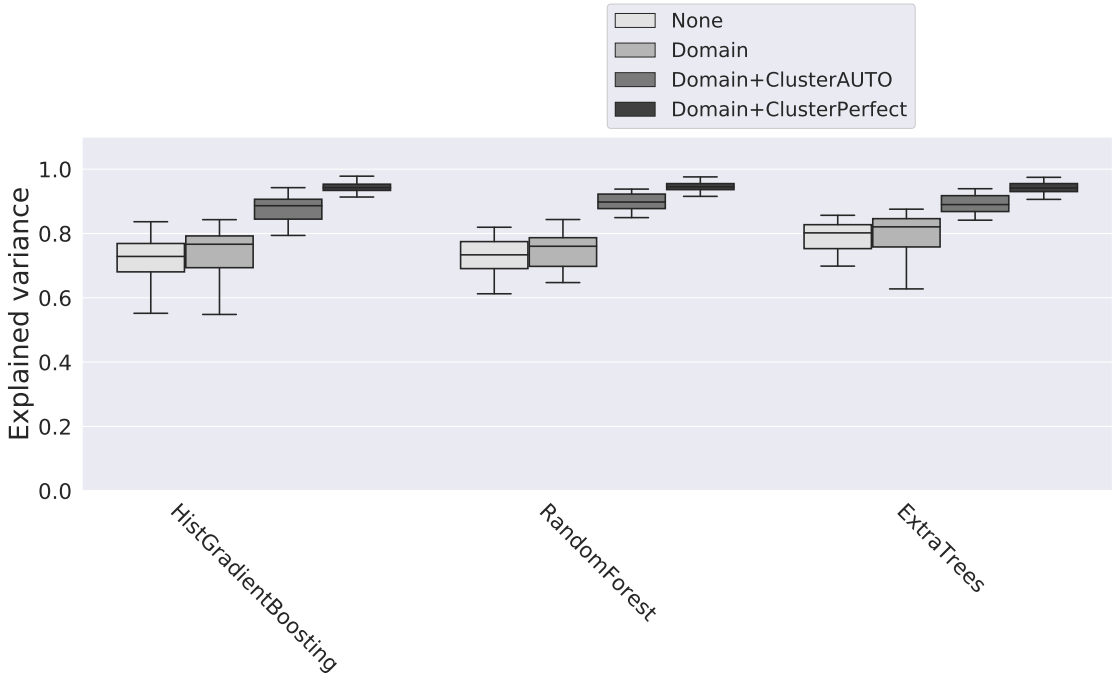


(a) R^2 score, higher results are better.



(b) RMSE, lower results are better.

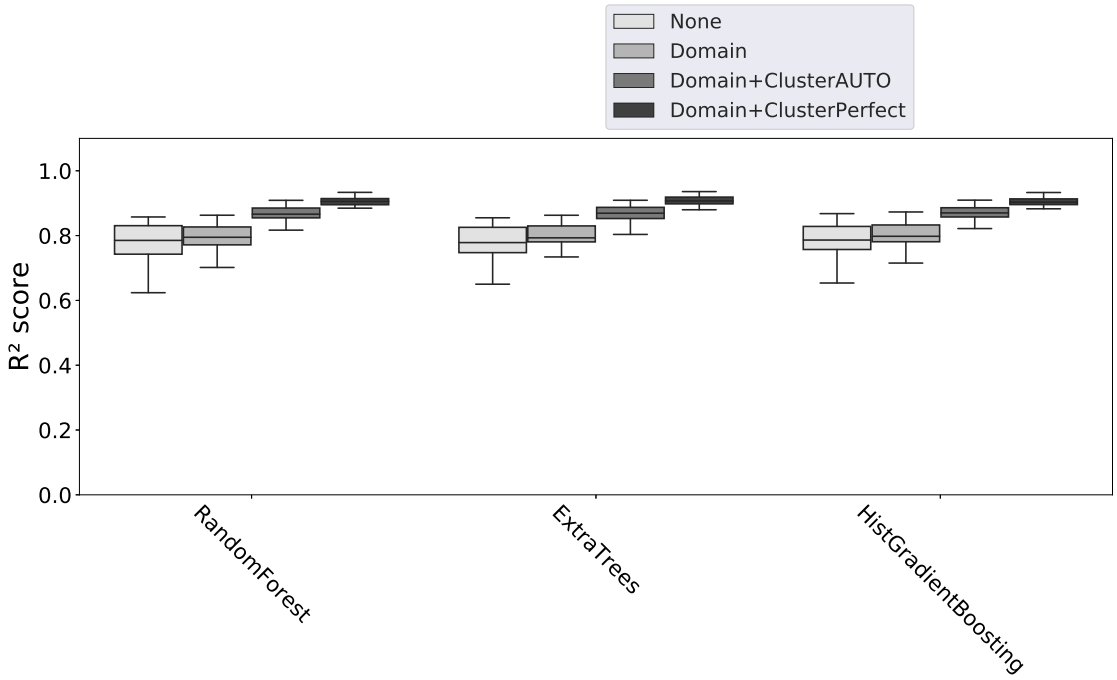
Figure 6-5: Turn final regression over P1



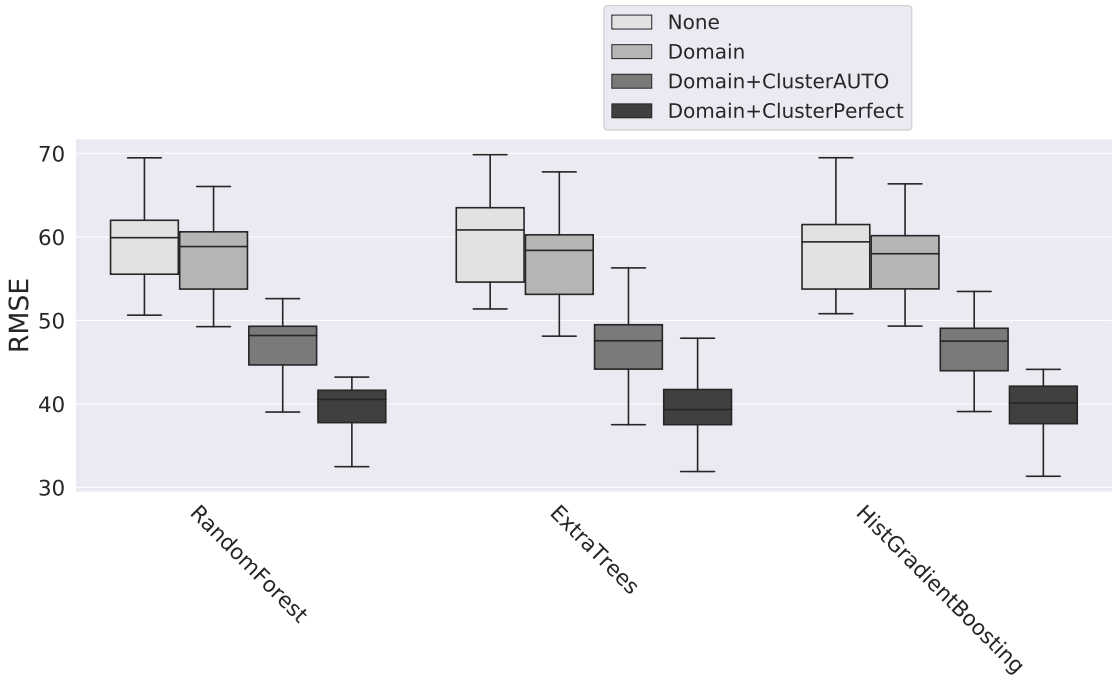
(c) Explained variance, higher results are better.

Figure 6-5: Turn final regression over P1

Turn Neck Action We performed regression on the turn-neck parameter P1. Since there is a lot of turn-neck action, we chose to explore those actions that occur at the same timestamp as the kick action. The results are shown in Figure 6-6 on page 72. We observed that by adding the agent's beliefs state (B_t^i) as features, we were able to improve the results slightly relative to those for the kick action.

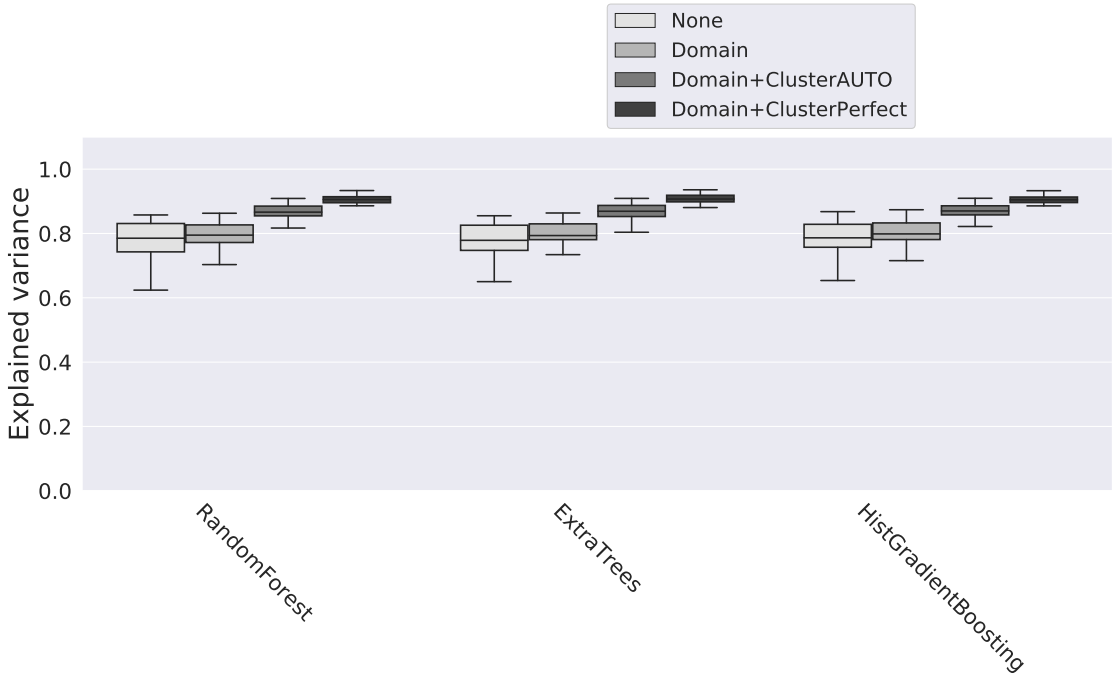


(a) R^2 score, higher results are better.



(b) RMSE, lower results are better.

Figure 6-6: TurnNeck final regression over P1



(c) Explained variance, higher results are better.

Figure 6-6: TurnNeck final regression over P1

Part II

Mining Action Sequences from Logs

Chapter 7

Hierarchical Sequence Mining for Discovering Action Sequences

The previous part of the thesis focused on predicting a single action, given the beliefs (and goals) of the agent at the time the decision on the action is taken. This implies a Markovian assumption in the sense that each such decision is assumed to be taken anew, without considering the agent's possible commitment to a plan, i.e., an ordered sequence of actions.

This part complements the approach taken above, by focusing on predicting sequences of actions, taken together, without considering the agent's beliefs state. We focus on mining the logs to discover frequent sequences of actions that may therefore be considered representative (fragments of) plans taken by the agent. This puts the beliefs and goals of the agent to the side, and only looks at the actions taken.

Section 7.1 will present the challenge in utilizing *hierarchical sequence mining* techniques for mining actions in logs. Section 7.2 will present the challenge raised in general in hierarchical sequence mining (and in particular in its use for learning from logs). The next chapters then present the solution approach.

7.1 Hierarchical Sequence Mining from Logs

In general, sequence mining aims to mine interesting (e.g., frequent) sequential patterns from the potentially large collection of input sequences. In principle, it may therefore be used to identify repeating interesting sequences of actions that are taken by agent, i.e., its *plans*. This allows prediction not only of a single action as discussed in the previous part, but prediction of multiple actions, in sequence, given a first action which starts the sequence. A different way of looking at this is that the prediction of an action will also consider the plans of the agent.

For example, consider the *dribble* action in the domain of *RoboCup2D*. The dribble action consists of sequences of three actions: *kick* with low power (P1) and direction(P2), followed by *turn* and *move* actions with direction which depending on where the ball rolls. The dribble ends after one or more repetitions of the sequence, when the last kick is with high power(P1). Predicting or recognizing the first action in the sequence and identifying it as such, and given knowledge of the expected sequence of turn and move actions, allows predicting the next few actions in the sequence after the first action.

The discovery of such representative sequences of actions may be tackled, in principle, by applying *sequential pattern mining* algorithms, such as those discussed earlier in Section 2.2. Such algorithms accept a database of observed sequences (in our case, a database of action sequences as observed in logs), and extract *interesting* patterns (sub-sequences) in the data; where *interest* is measured via some function, typically the frequency of appearance in the data. Commonly, such patterns are considered interesting when they repeat frequently, though there are various measures of interest depending on intended use.

Unfortunately, actions with continuous parameters pose a significant challenge to sequence mining algorithms. Sequence mining algorithms assume a finite (discrete) alphabet of symbols (letters) from which sequences are built. When we treat each action and its parameters as a separate symbol, the result is an infinite alphabet.

Restricting the parameter values to quantized (discrete) values puts a finite limit on the number of symbols, but inherently leads to information loss, as we lose infor-

mation about the distance between parameter values. A kick at an angle of 1 is closer to a kick at an angle of 2 than to a kick at an angle of 90. This information may be lost in the discrete version, as different kicks can become different symbols.

To limit the information loss, a common practice is to discretize the continuous values, by *binning* (collecting) together close values. For example,a binning at 90 degrees would result in four (4) kick symbols: kick directions at 0 degrees to 90 degrees can be binned together as a single symbol (*kick forward-right*), kick directions at 90–180 degrees would be considered *kick backward-right*, etc. Binning at 1-degree would result in 360 kick types (just on the direction parameter).

Indeed, one can construct a hierarchical graph of discretization levels—a hierarchical *taxonomy*. Figure 7-1 shows an example of such a taxonomy hierarchy, for the *Kick* action and its parameters: *power* $P1 \in [0, 100]$ and *direction* $P2 \in [0, 360]$. The figure shows a portion of the resulting taxonomy from applying only two levels of hierarchy, for the two parameters. The $P1$ is generalized to four values: *low* ($0 \leq P1 < 33$), *medium* ($33 \leq P1 < 66$), *high* ($66 \leq P1 \leq 99$), and *MAX* ($P1 = 100$). $P2$ is similarly generalized to four values: *Forward&Right* ($0 \leq P2 < 90$), *Backward&Right* ($90 \leq P2 < 180$), *Backward&Left* ($180 \leq P2 < 270$), and *Forward&Left* ($270 \leq P2 < 360$). The notation uses the # symbol to seperate between the action name and its parameters. For example, a *kick* action with $P1 = 100$ and $P2 = 0$ is denoted *Kick#100#0* and a *turn* with $P1 = 45$ is denoted *Turn#45*. The symbol ? is used to denote a wildcard, i.e., a value encompassing all below (the most abstract level of the taxonomy).

For example, we generalize both *kick#75#120* and *kick#75#105* to *kick#High#Backward&Right*. In the second level of hierarchy over the kick action, we will generalize P2 to Forward/Backward only, so to generalize the two items *kick#75#120* and *kick#85#200*. *kick#75#120* in first level of hierarchy will generalize to *kick#High#Backward&Right* and in second level will generalize to *kick#High#Backward*. *kick#85#200* in first level of hierarchy will generalize to *kick#High#Backward&Left* and in second level will generalize to *kick#High#Backward*. The third level of hierarchy accounts for any P1 value (marked

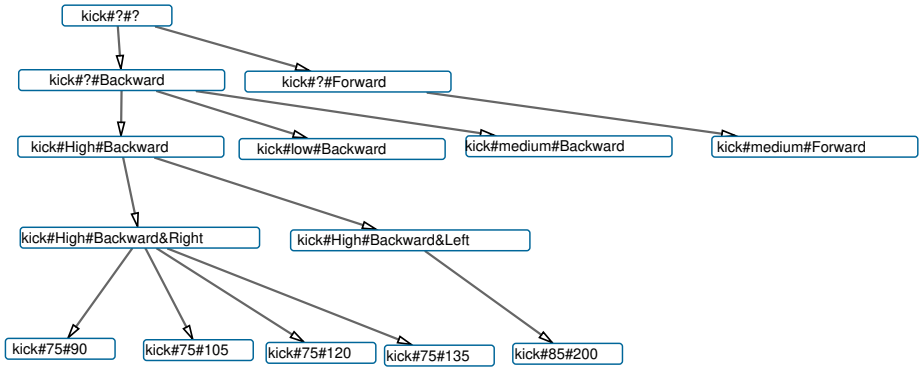


Figure 7-1: A part of the RoboCup *kick* taxonomy.

?). Thus *kick#High#Backward* and *kick#Low#Backward* to *kick#?#Backward*.

The fourth level of hierarchy over the kick action will be over any P2 value, denoted *?* as any value of some parameter. We will generalize *kick#?#Backward* and *kick#?#Forward* to *kick#?#?* which generalizes *kick* with any P1 and P2 values, which generalizes over 36,000 combinations of kick with changing values of P1 and P2.

Given a taxonomy hierarchy over the continuous parameters of actions, we now seek to discover (mine) generalized sequences of actions that appear in the logs. The idea is to adapt the appropriate level of discretization to the data, so as to discover the most useful sequences.

There are several relevant methods for hierarchical sequential pattern mining, which are able to find the frequency of sequential patterns combining symbols of different levels of generalization. Plantevit *et al.* [57] presented the *HYPE* algorithm to incorporate hierarchies in mining multidimensional (multi-parameter) sequences over several levels of hierarchy. In their approach, they prune hierarchies by only considering maf-sequences, which are pairs of items (each belonging to a dimension) that are maximal (i.e., each specialization is infrequent). A known limitation of their approach is that they do not mine all frequent sequences. While our interest is over

the coverage and the accuracy of the *sequence dataset*. *MFH-SPAM* (Multi-Feature Hierarchical Sequential PAttern Mining, [85]) is an algorithm that extracts a small set of patterns that best differentiate sequences of actions taken by students with different learning behavior profiles, similarly using a taxonomy hierarchy. In *MFH-SPAM* again, the known limitation of their approach is that they do not mine all frequent sequences. Large-Scale Sequence Mining with Hierarchies (LASH) [6] is a distributed algorithm for mining sequence patterns supporting parallelization, and allowing for gaps and maximum length constraints.

All of these algorithms assume that the taxonomy hierarchy is given by the human expert, and we follow in the same path. We ask the human expert to provide the taxonomy hierarchy for the continuous actions observed in the logs. We then use LASH to generate a list of sequences with frequency counts. This is the first step in our approach.

7.2 The Challenge: Generalization vs Accuracy

As we reveal below, there is an open challenge in using the results of hierarchical sequence mining algorithms as the basis for modeling an agent's behavior, as recorded in logs. The challenge is inherent to the use of a taxonomy hierarchy, and is general to all hierarchical sequence mining algorithms.

To simplify the presentation, we use English vocabulary to illustrate the challenge. Imagine mining English texts. Symbols are letters, and sequences thus form words. Let us define a generalization hierarchy for the English letters; a simple hierarchy with one level, where each letter's upper and lower cases are generalized together: The letter a and the letter A will be generalized to general letter a', the letter b and the letter B will be generalized to the general symbol b', and so on (Figure 7-2).

Suppose now we apply sequence mining to English texts. Intuitively, we expect the word *the* to appear very frequently. A non-hierarchical sequence miner is expected to yield a sorted list of all sequences found, with their frequency counts, as shown in Table 7.1 for all instances of the word *t'h'e'* (i.e., all mixed lower- and upper- case

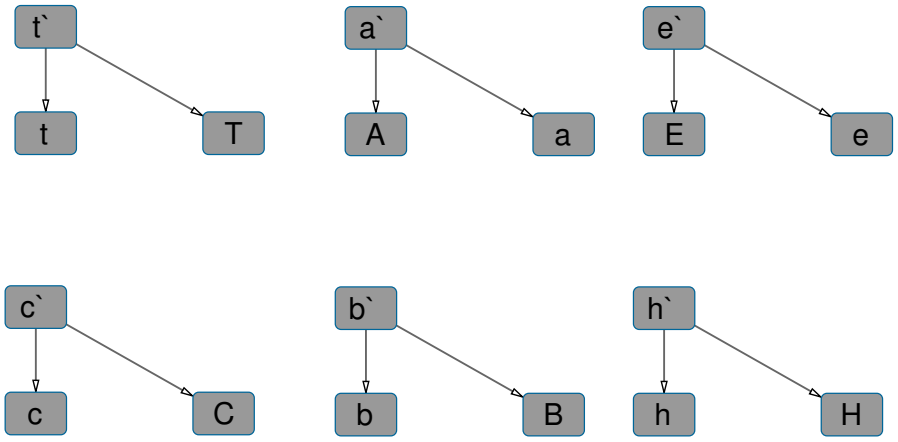


Figure 7-2: English part of *vocabulary* with hierarchy between items example.

forms of *the*). For each sequence s_i, we see its count c_i, i.e., the number of times it appears in the data. The lower-case word *the* appears 3000 times. The word *The* appears 2000 times. We see also there are some marginal counts of other mixed-case instances: *tHe, thE, tHE*.

s_1:	THe	c_1:	0
s_2:	ThE	c_2:	0
s_3:	THE	c_3:	0
	...		
s_8:	tHe	c_8:	2
s_9:	thE	c_9:	5
s_{10}:	tHE	c_{10}:	7
	...		
s_{16}:	The	c_{16}:	2000
	...		
s_{20}:	the	c_{20}:	3000

Table 7.1: Non-hierarchical counts of the word *the* in sequence dataset. We explicitly show entries with 0 counts, though in practice these would not be represented explicitly.

A hierarchical sequence mining algorithm, however, does not stop there, as it also discovers all generalized forms of the word, as shown in Table 7.2. It becomes apparent that more general sequences appear more frequently, as they combine the frequencies of the more specific instances which they generalize. For example, the

generalized sequence s_{21} (*th'e*) appears 3002 times. Of those, 3000 appearances are of the instance *the* (s_{20}), and the other two are of the instance *tHe* (s_8). s_{27} appears 5014 times, the sum of all instanciations of the word *t'h'e'* in all possible letter cases in the dataset.

s_1:	THe	c_1:	0
s_2:	ThE	c_2:	0
s_3:	THE	c_3:	0
s_4:	t'He	c_4:	0
s_5:	t'hE	c_5:	0
s_6:	Th'E	c_6:	0
s_7:	THe'	c_7:	0
s_8:	tHe	c_8:	2
s_9:	thE	c_9:	5
s_{10}:	tHE	c_{10}:	7
s_{11}:	t'HE	c_{11}:	7
s_{12}:	tHe'	c_{12}:	9
s_{13}:	t'He'	c_{13}:	9
s_{14}:	th'E	c_{14}:	12
s_{15}:	t'h'E	c_{15}:	12
s_{16}:	The	c_{16}:	2000
s_{17}:	The'	c_{17}:	2000
s_{18}:	Th'e	c_{18}:	2000
s_{19}:	Th'e'	c_{19}:	2000
s_{20}:	the	c_{20}:	3000
s_{21}:	th'e	c_{21}:	3002
s_{22}:	the'	c_{22}:	3005
s_{23}:	th'e'	c_{23}:	3014
s_{24}:	t'he	c_{24}:	5000
s_{25}:	t'h'e	c_{25}:	5002
s_{26}:	t'he'	c_{26}:	5005
s_{27}:	t‘h‘e‘	c_{27}:	5014

Table 7.2: Counts for the generalized word *the* appearing in sequence dataset, a product of a hierarchical sequence mining. We explicitly show entries with 0 counts, though in practice these would not be represented explicitly.

Given the output of the hierarchical sequence mining algorithm, it is not clear how to best proceed. For instance, even if we limit ourselves to selecting a single sequence out of the list to stand for the texts, the choice is not clear: In the example above, the most general sequence s_{27} *t'h'e'* has the highest frequency, and accounts for all instances appearing in the mined texts. Yet we understand intuitively, that it does

not at all represent the distribution of the instances in the text, as it implies that all instances are equally likely to appear. Instead the sequence s_{24} *t'he* has higher fidelity to the distribution of the instances in the texts, despite not accounting for 14 instances. Moreover, if we allow two sequences to be selected, should we select the two top sequences s_{24}, s_{20}? The sequences represented by s_{20} are already a part of the top sequence, so the selection of s_{20} does not seem to add information. Instead, selecting s_{24} and either s_{15} or s_{14} seems more informative: it accounts for almost all of the sequences found, while separating the most frequent sequences from those whose appearance seems to be an outlier.

These examples reveal an inherent trade-off in choosing an appropriate generalization level, between accuracy and representative power.

- Higher taxonomy levels generalizes words, and this results in fewer mined sequences accounting for observed words. A single sequence *t'h'e'* covers (represents) all words found in the database. However, the single sequence has limited *accuracy*. Seeing an upper case *T*, the sequence only allows predicting that the next letter is either an upper case or lower case *h'*, though clearly, based on the data, it is a lower case *h*. Put differently, the general sequence also represents sequences *not* found in the logs, and is therefore not accurate.

- On the other hand, more specific mined sequences can be much more accurate, but lose the generalization property. For example, selecting exactly the five sequences appearing in the data (*the, The, tHe, tHE, thE*) surely allows accurate prediction (within likelihoods) of letter appearance within the sequence. However, we now require five sequences to represent the words. If we follow this methodology, the only accurate description of a words database is a list of all words appearing in it, with counts.

Note that had we asked for the *minimal number of sequences balancing accuracy and coverage*, a reasonable answer would include two sequences (*the* (s_{16}) and *The* (s_{20}), which account for 5000 of the sequences in the data out of 5014 (thus provide excellent coverage of the words observed in the data), while providing very high

accuracy as to the distribution of words. The few instances not covered by these sequences English speakers would naturally classify as typos.

This is a general challenge for hierarchical sequence mining. Given K, a limit on the size of the set of *mined sequences* of a given length, the sequence mining process has to balance representative power (coverage; the number of actual sequences in the logs that are represented by sequences in K), with accuracy (reducing the errors implied by generalization). Or, given a target accuracy level or coverage level, the algorithm has to minimize $|K|$.

Chapter 8

Accuracy, Coverage, and Compactness

We present preliminaries for discussing hierarchical sequence mining (HSM) (Section 8.1). Then we describe our different measures for the quality of a sequence mining model (the result of running a mining algorithm) in Section 8.2. We then show how the balance between these different measures can be formally described (Section 8.3).

8.1 Formal Preliminaries

We begin with a formal definition of the hierarchical sequence mining (HSM) problem and related concepts. The notation and terminology closely follow standards in the sequence mining literature and differ only when necessary.

8.1.1 Datasets, symbol sequences, and taxonomies

The *sequence dataset* which is mined is a multiset $D = \{s_1, s_2, \ldots, s_{|D|}\}$. Each sequence $s \in D$ is a series of *symbols* $s \equiv (w_1, w_2, \ldots, w_{|s|})$, where each symbol w_i is taken from a set of symbols Σ, called an *alphabet* and assumed to be a finite set. Action logs, taken together as discussed above, can serve as a *sequence dataset* for hierarchical sequence mining. The mining algorithm will look for patterns of symbols which appear in the sequences making up the logs. A partial example of a dataset appears in Table 8.1. It shows subsequences of length 2 and 3 that appear within the

full dataset.

s_1:	the	s_2:	the	s_3:	the
s_4:	The	s_5:	The	s_6:	The
s_7:	The	s_8:	thE	s_9:	ThE
s_{10}:	tHE	s_{11}:	ThE	s_{12}:	ch
s_{13}:	the	s_{14}:	the	s_{15}:	CH
s_{16}:	the	s_{17}:	ch	s_{18}:	ch
s_{19}:	ch	s_{20}:	CH	s_{21}:	Ch
s_{22}:	bg	s_{23}:	bg	$s_{\|D\|}$:	BG

Table 8.1: English partial sequence dataset.

A taxonomy hierarchy $H(\Sigma)$[1] over the alphabet Σ is a *directed* graph $\langle V, E \rangle$ where V is a set of vertices, and E is a set of edges. The sets V and E are defined as follows.

1. All members of Σ are members of V, i.e., $\sigma \in \Sigma \implies \sigma \in V$. We call these the *grounded symbols*.

2. For any member $v \in V$ which is generalized to a symbol u, then by definition $u \in V$, and $(u, v) \in E$. We then denote $u \to v$, and say u is a generalization of v.

3. For all $\sigma \in \Sigma$, there exist no member $v \in V$ such that $\sigma \to v$, i.e., the grounded symbols are never generalizations of another symbol, grounded or otherwise. This makes the grounded symbols leafs in the graph. $\sigma \in V \wedge \sigma \in \Sigma \implies \nexists u \in V, (u, \sigma) \in E$

4. Given three vertices $v, u, r \in V$ such that $v \to u$, and $u \to r$, there must not exist $v \to r$. The hierarchical path $v \to u \to r$ can be annotated $v \rightsquigarrow r$.

5. A member in V is either a grounded symbol, or it is a generalization of a different member in V. It may not generalize itself, and if it is not a ground symbol, it must generalize some symbol.

6. A symbol $v \in V$ which has a positive out-degree but in-degree of 0 (i.e., generalizes other symbols, but is not generalized by any) is called a *root symbol*.

[1] We use H for short, when Σ is clear from context.

This definition of the taxonomy hierarchy results in clear hierarchical structures, with grounded symbols as leaves, and generalizations of symbols proceeding one level at a time along hierarchical edges. Note that a leaf (or any vertex) may have multiple parents that have a single joint parent, and thus there may be multiple hierarchical paths leading from a leaf to an intermediate vertex. However, no cycles are possible along the directed edges. We have seen example taxonomies in the previous chapter, for both English letters, as well as RoboCup kick actions (generalizing over the power and direction parameters).

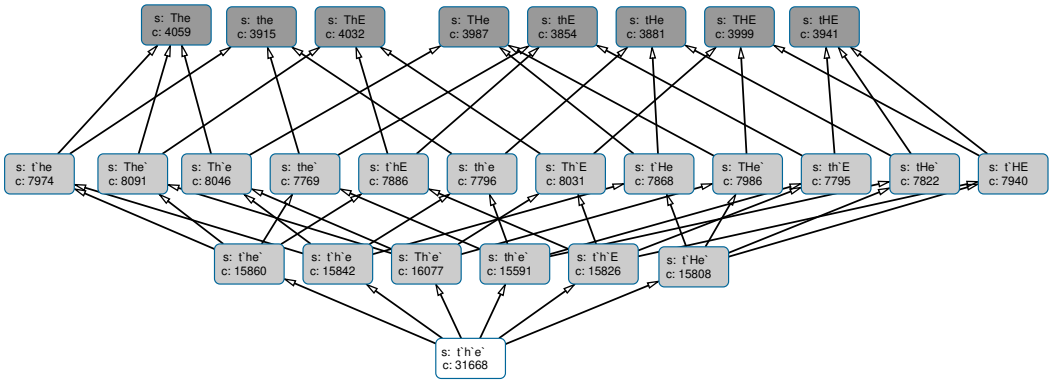
8.1.2 Grounded and Generalized Sequences in sequence dataset

Given a dataset D containing sequences made of symbols of an alphabet Σ, and a taxonomy $H(\Sigma) = \langle V, E \rangle$, we can also now extend the generalization to sequences, rather than single symbols. Specifically, we say that a sequence $s = (s_1, s_2, \ldots, s_{|s|})$ generalizes a sequence $t = (t_1, t_2, \ldots, t_{|t|})$ if each symbol of s, in order, is either equal to the corresponding symbol in t, or generalizes to it. This by itself allows identical sequences to be considered generalizations of each other, and so we also add the condition that at least one symbol in s is different from the corresponding symbol in t (combined with the previous condition, this means that necessarily at least one symbol in s generalizes the corresponding symbol in t). Formally, s generalizes t (denoted $s \to t$) if the following hold:

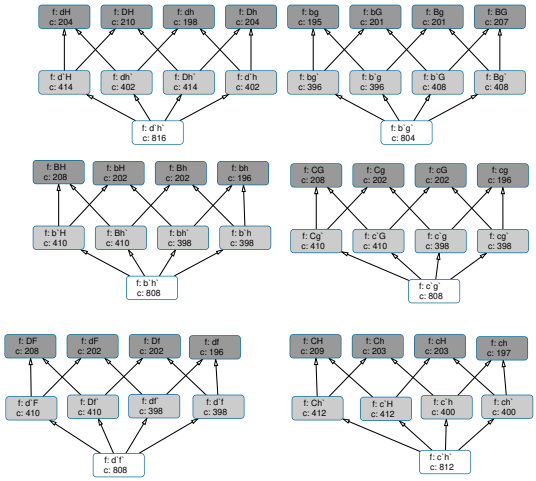
1. $|s| = |t|$
2. $\forall s_i \in V, \forall t_j \in V, 0 \leq i, j \leq |s|$
3. $\forall i, 0 \leq i \leq |s|$, either $s_i = t_i$, or $s_i \to t_i$
4. $\exists j, 0 \leq j \leq |s|$ where $s_j \neq t_j$

Part of hierarchy sequence tree over the English domain is shown in Figure 8-1a. In this example, the sequences $s_1 = The$, $s_2 = t\text{`}h\text{`}e$, $s_3 = t\text{`}he\text{`}$, $s_4 = t\text{`}h\text{`}e\text{`}$ satisfy $s_2 \longrightarrow s_1$, $s_3 \longrightarrow s_1$, $s_4 \longrightarrow s_3$, $s_4 \longrightarrow s_2$, and $s_4 \rightsquigarrow s_1$ (s_4 is the most general form of s_1, we re-use the notation $\rightsquigarrow$ used for symbols). Here, s_1 has more than one

direct parent (s_2 as well as s_3), a good example of a common property of generalized sequences.



(a) Part of English sequence hierarchy tree between items over length 3 example.



(b) Part of English sequence hierarchy tree between items over length 2 example.

Figure 8-1: Part of English sequence hierarchy tree between items example.

We distinguish *leaf (ground) sequences* (denoted W^+_{Leaf}) that are composed solely of ground symbols, from *root sequences* (denoted W^+_{Root}), made solely from root symbols. All other sequences, which are not made from all ground or all root symbols, are called *intermediate sequences*, and denoted I^+. $I^+ = W^+ \setminus (W^+_{Root} \cup W^+_{Leaf})$. Note

that we omit the data set for brevity, as it is typically clear from context. In our example:

- $W^{+}_{Leaf} = \{the, The, THe, bg, BG, \ldots, ch\}$
- $W^{+}_{Root} = \{t`h`e`, b`g`, \ldots, c`h`\}$
- $I^{+} = \{t`h`e, Th`e, THe`, b`g, Bg`, \ldots, c`h\}$

8.1.3 Sequential Pattern Summary of a Sequence Data Set

A *sequence dataset* D will contain only leaf sequences, made of grounded symbols. Commonly, sequence mining algorithms seek to determine a set of sequences that best represent D, in a compact form, generally a list of sequences and their relative count in D (this is called *support* in data mining literature). Non-hierarchical algorithms output a set of ground sequences and their counts. As it is impractical (and not useful) to simply output the list of all ground sequences, there are common constraints on the sequences, e.g., that their frequencey in D is above some support threshold, and they satisfy some statistical property.[2]

We use the term *Summarized Data Set* to denote the frequency count of all sequences in the data set, without any filtering (e.g., ignoring support threshold and any statistical property testing). The *Summarized Data Set* is the most basic summary of a *sequence dataset* D. It is a set of tuples (s, c), where s is a sequence and c is its frequency in the data set.

Non-hierarchical algorithms can only output the ground-truth *Summarized Data Set*: A list of all sequences appearing in the data, each with its frequency count. Hierarchical algorithms can shorten this list of sequences, by using a single generalized sequence g to capture the appearances of multiple ground sequences s_i in D, if $\forall s_i \in D, g \rightsquigarrow s_i$. For example, instead of providing both *The* and *the* in the resulting sequence list, a hierarchical miner may return *t`he*, which not only is a more compact representation of both sequences, but also admits a more accurate statistical property checks (as it captures the underlying language knowledge).

[2]See [19] for surveys of such properties and their significance.

The task is to determine a set of sequences—leaf, intermediate, or root—that best describes D, in the sense of frequency, i.e., the number of times a given sequence appears in D. The challenge for hierarchical algorithms is to shorten this list while not sacrificing other properties (e.g., fidelity of the description).

We begin with notations and definitions. Supposed we are given a sequence s, and a *sequence dataset* D. We use the notation $D(s)$ to denote the multi-set that contains all appearances of s in D, or all appearances of any sequences in D which s generalizes. $D(s)$ is induced by the *appearances* of s in D, as follows:

$$D(s) := \{t | t \in D \wedge (s = t \vee s \rightsquigarrow t)\}$$

The frequency of a sequence s_{dest} in a given *sequence dataset* D is then simply the size of the induced multiset $D(s_{dest})$, which we denote $Freq(s_{dest}, D)$. For example, in *sequence dataset* is D^{part} represent in Table 8.1 on page 85, we have:

$$\begin{aligned}
Freq(the, D^{part}) &= |D^{part}(the)| \\
&= |\{s_1, s_2, s_3, s_{13}, s_{14}, s_{16}\}| \\
&= |\{the, the, the, the, the, the\}| = 6 \\
Freq(t`he, D^{part}) &= |D^{part}(t`he)| \\
&= |\{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_{13}, s_{14}, s_{16}\}| \\
&= |\{the, the, the, The, The, The, The, the, the, the\}| = 10 \\
Freq(b`g`, D^{part}) &= |D^{part}(b`g`)| \\
&= |\{bg, bg, BG\}| = |\{s_{22}, s_{23}, s_{|D|}\}| = 3
\end{aligned}$$

We use the term *Summarized Data Set* to denote the sequential pattern mining result, i.e, the output of a mining algorithm. The most basic form of *Summarized Data Set* is a set of tuples (s, c), where s is a sequence and c is its frequency in the data set. Thus for instance, a possible *Summarized Data Set* for the data in

sequence dataset (D^{full}) can be found in Table 8.2 on page 90. It consists of tuples $(s_i \in D, Freq(s_i, D))$. For example, the sequence $s_2 = the‘$ appears in D^{full}, 7769 times, in two grounded forms: *the* (3915 times, see Figure 8-1 above), and *thE* (3854 times, Figure 8-1). Indeed, we add the $c_i = Freq(s_i, D^{full})$ to each sequence $s_i \in W^+$ appearing in Figure 8-1 on page 87.

(s_1, c_1) :	(The, 4059)
(s_2, c_2) :	(the‘, 7769)
(s_3, c_3) :	(tHe‘, 7822)
(s_4, c_4) :	(t‘h‘e‘, 31688)
(s_5, c_5) :	(c‘h‘, 812)
(s_6, c_6) :	(b‘g, 816)
(s_7, c_7) :	(b‘G, 4059)

Table 8.2: English domain SDS_{Dest}

The *Ground Truth Summarized Data Set* of a dataset D is the set of tuples made of all *ground* sequences in the D and their frequency count, i.e.,

$$SDS_{GT} := \{(s, c) | s \in D \cap W^+_{Leaf}, c = Freq(s, D)\}$$

For the example *English domain* D^{full}, the SDS_{GT} is shown in Table 8.3 on page 90.

(s_1, c_1) :	(The, 4059)
(s_2, c_2) :	(the, 3915)
(s_3, c_3) :	(ThE, 4032)
(s_4, c_4) :	(THe, 3987)
(s_5, c_5) :	(thE, 3854)
(s_6, c_6) :	(tHe, 3881)
(s_7, c_7) :	(THE, 3999)
(s_8, c_8) :	(tHE, 3941)
(s_9, c_9) :	(cH ,203)
(s_{10}, c_{10}) :	(ch, 197)
(s_{11}, c_{11}) :	(CH, 209)
(s_{12}, c_{12}) :	(Ch, 203)
(s_{13}, c_{13}) :	(Bg, 201)
(s_{14}, c_{14}) :	(bg, 195)
(s_{15}, c_{15}) :	(bG, 207)
(s_{16}, c_{16}) :	(BG, 201)

Table 8.3: English domain SDS_{GT}

The ground truth SDS_{GT}, is the basis for computing the quality of any other *Summarized Data Set*. The size of SDS_{GT} is, by definition, the largest and so it gives a measure of the worst-case length of the *Summarized Data Set*. Intuitively, however, the SDS_{GT} also represents a best case, as it is the most accurate description of the data, an exact histogram of the data set. All ground sequences are present and precisely counted.

In contrast, the *General Summarized Data Set* of a dataset D is the set of tuples made of all *general* sequences in the D and their frequency count, i.e.,

$$SDS_g := \{(s, c) | s \in D \cap W^+_{Root}, c = Freq(s, D)\}$$

For the same D^{full}, the SDS_g is shown in Table 8.4 on page 91.

(s_1, c_1) :	(t‘h‘e‘, 31668)
(s_2, c_2) :	(b‘g‘, 804)
(s_3, c_3) :	(c‘h‘, 812)

Table 8.4: English domain SDS_g

SDS_g is the basis for computing the *compactness* of any other *Summarized Data Set*. It accounts for every possible sequence in the database, but its size is the smallest possible . It there measures the best-case length of any *Summarized Data Set* with maximum *coverage*. Intuitively, however, the SDS_g may be a less accurate description of the data, as ground truth sequence counts can only be inferred from the their combined totals in their general forms.

8.2 Evaluating the Quality of a *Summarized Data Set*

The building of SDS_{Dest} needs to evaluate the trade-off between several measures:

- Compactness, which is the size of the *Summarized Data Set* (number of entries in $|SDS_{Dest}|$), discussed in Section 8.2.1.
- Coverage, which measures how many of the sequences in the database D are accounted for by the SDS_{Dest} (Section 8.2.2).

- Accuracy, which measures the number of correct predictions represented by SDS_{Dest}, with respect to the ground truth sequence frequency counts (Section 8.2.3).

8.2.1 Compactness

A *Summarized Data Set* is a set of tuples, which represent a dataset. Different SDSs can be more or less compact (i.e., can vary in size). The compactness of an SDS is easily measured by its size compared to the size of the ground truth representation SDS_{GT} (Eq. 8.1). It takes values from $\frac{1}{|SDS_{GT}|}$ (best, a single sequence) to 1 (worst, same size as the ground truth SDS).

$$Compactness(SDS, SDS_{GT}) := \frac{|SDS|}{|SDS_{GT}|} \tag{8.1}$$

The size of the SDS is easily manipulated simply by inserting or removing sequences from it. What we require is a measure of the *fidelity* of the SDS to the data. We do this using two separate factors: *coverage* and *accuracy*.

8.2.2 Coverage

We define the *Coverage* of a candidate *Summarized Data Set* over D (denoted SDS) as the total number of *ground* sequences which are either directly contained in it, or are a specialization of a generalized sequence appearing in it.

$$Coverage(SDS, D) := |\{e \in D | (s, c) \in SDS \wedge (s \rightsquigarrow e \vee s = e)\}| \tag{8.2}$$

The definition is somewhat close to the definition of frequency, but it prohibits counting a grounded sequence more than once. Two sequences in a SDS may induce the same grounded sequence, and it should be counted only once for the purposes of coverage. For example, suppose we are evaluating the coverage of a SDS for the data set D^{full} (Figure 8-1), where SDS is based on two sequences *The* and *Th'e*. The frequency of the sequence *The* is 4059 ($Freq(The, D^{full})$). The

frequency of the sequence *Th'e* in the same data set is 8046 (it generalizes both *The* and *THe*). Thus $SDS := \{$(The,4059), (Th'e,8046)$\}$. Naively, one would assume $coverage(SDS, D^{full}) = 4059 + 8046$. However, because $Th'e \rightsquigarrow The$, then $D^{full}(The) \subset D^{full}(Th'e)$. The count 8046 *includes* the count 4059, and therefore $coverage(SDS, D^{full}) = 8046$. Similarly, given $SDS := \{(Th'e, 8046), (THe, 3987)\}$, it is still the case that $coverage(SDS, D^{full}) = 8046$, as we avoid double-counting over the sequence intersection.

Relative Coverage Given a specific *SDS*, it is possible to compute its coverage in relative terms, i.e., the percentage of sequences it covers compared to the entire data set. This is easily computed by dividing $Coverage(SDS, D)$ by the coverage of the ground truth SDS (Eq. 8.3).

$$RC(SDS_{Dest}, D) := \frac{Coverage(SDS_{Dest}, D)}{Coverage(SDS_{GT}, D)} \tag{8.3}$$

This normalized measure of the coverage of a SDS_{Dest} ranges from 0 to 1. A RC of 1 represents maximum coverage over D, while a value of 0 represents no coverage.

By nature, two specific *SDS* always have perfect relative coverage of 1: SDS_{GT}, and SDS_g. The first, because all sequences are present in the ground truth SDS_{GT}, and the latter, because it includes all top-generalized sequences, which generalize all grounded sequences, and thus covers all of them. Intuitively, we understand however that these two descriptions of the data have differences which are very meaningful. Their compactness is very different, and likewise their accuracy with respect to the frequency counts. We discuss this next.

8.2.3 Accuracy

Intuitively, SDS_{GT} is maximally accurate, as it reports on the precise counts of each ground sequence. Likewise intuitively, SDS_g which includes only the most general sequences is not likely to be accurate, as it reports only on the sum of the ground sequence frequencies, which means that generally, the exact counts of ground sequences are unknown.

Here we formalize these intuitions in a measure of the accuracy of a candidate SDS. The idea is that the accuracy of a candidate SDS should be increased with the number of correct counts.

We are given an summary description SDS, and a dataset D. Tuples $t = (s, c) \in SDS$ are supposed to represent (summarize) the occurrence of s in D, i.e., ideally $c = Freq(s, D)$. We will define an accuracy measure that aggregates the predictions of all individual tuples in SDS. Our definition of accuracy is based on standard definitions in machine learning, which considers accuracy to be the ratio of the number of correct predictions, to the total number of predictions.

We begin by considering the simplest case. A singleton SDS, containing a single summary tuple $(s, c) \in SDS$ where s is a ground sequence. In this case, the *accuracy* of the SDS with respect to D is given by

$$\psi_1(s, c, D) := \begin{cases} \frac{c}{|D|} & c < Freq(s, D) \\ \frac{\max(0, 2Freq(s,D)-c)}{|D|} & c \geq Freq(s, D) \end{cases} \tag{8.4}$$

The accuracy is measured by considering the number of correct predictions of s implied by (s, c). Suppose $s \in D$, then $Freq(s, D) > 0$. There are three cases when this is true:

1. If $c = Freq(s, D)$, then the number of correct predictions of s is indeed $Freq(s, D) = c$, which is $(2Freq(s, D) - c)$.

2. If $c < Freq(s, D)$ it means c underestimates $Freq(s, D)$. Then c is the number of correct predictions.

3. Otherwise, c over-estimates $Freq(s, D)$. In this case, we want to penalize the accuracy score, by "folding" the excess $(c - Freq(s, D))$ and subtracting it from the correct value $Freq(s, D)$: $Freq(s, D) - (c - Freq(s, D))$. If c is more than twice $Freq(s, D)$, this will be a negative number. Hence the use of $\max(x, 2Freq(s, D) - c)$ to bound the nominator from below at 0.

If $s \notin D$, the $Freq(s, D) = 0$, and indeed the number of correct predictions is 0

$(\max(0, 2 \cdot 0 - c))$.

Let us now consider a more general case. Suppose we are considering a singleton SDS, containing a single tuple (s, c), where s is a non-ground sequence, that generalizes multiple ground sequences.

We denote the set of all ground sequences generated from s by $Leaves(s)$ (Def. 8.5).

$$Leaves(s) = \{s_g \in W^+_{Leaf} | (s \rightsquigarrow s_g) \vee (s = s_g)\} \tag{8.5}$$

In the case that the sequence itself is a leaf (a ground sequence), the set will only include the leaf sequence itself, $Leaves(s_r) = \{s_r\}$ if $s_r \in W^+_{Leaf}$. Here are some examples:

1. $|Leaves(The)| = |\{The\}| = 1$
2. $|Leaves(t`he)| = |\{The, the\}| = 2$
3. $|Leaves(c`h`)| = |\{CH, Ch, cH, ch\}| = 4$
4. $|Leaves(Bg`)| = |\{Bg, BG\}| = 2$
5. $|Leaves(t`h`e`)| = |\{The, the, ThE, THe, thE, tHe, THE, tHE\}| = 8$

When s is a generalized sequence, its count c is the total for all ground sequences that are generated from s. Lacking other information, we assume that the ground sequences generated from s are uniformly distributed. The implication is that the summary tuple (s, c) generates a set of summary tuples $\{(s_i, \frac{c}{|Leaves(s)|}) | s_i \in Leaves(s)\}$.

For example, *Leaves(Th'e')* yields the set *{THE, THe, The, ThE}*. Suppose we are given the summary tuple $t = (Th'e', 4000)$. The implication of the uniform distribution assumption is that the tuple t stands for four summary tuples $t_1 = (THE, \frac{4000}{4})$, $t_2 = (THe, \frac{4000}{4})$, $t_3 = (The, \frac{4000}{4})$, $t4 = (ThE, \frac{4000}{4})$.

More generally, given a summary tuple (s, c), the *ground tuple set* of a summary tuple (s, c) is given by:

$$gts(s, c) := \{(s_i, \frac{c}{|Leaves(s)|}) | s_i \in Leaves(s)\}. \tag{8.6}$$

Note that for the case where s is a ground sequence, this still holds, as $Leaves(s) = s$ in this case, with $|Leaves(s)| = 1$.

Let us now remove the assumption the our SDS is a singleton, and instead consider a more general case of an $SDS(s_i, c_i), i \geq 1$ made of several tuples. We form a *ground summary* $Ground(SDS)$ by collecting together all ground tuple sets resulting from tuples in SDS:

$$Ground(SDS) := \bigcup_{(s,c) \in SDS} gts(s, c) \tag{8.7}$$

In principle, the accuracy of SDS can now be calculated as the sum of the accuracy values of each tuple in $Ground(SDS)$, i.e.,

$$\psi(SDS, D) := \sum_{(s_i,c_i) \in Ground(SDS)} \psi_1(s_i, c_i, D)$$

However, a closer examination reveals that this is not straightforward, because some sequences may be summarized more than once in $Ground(SDS)$, i.e., it is possible that there are two elements $(s, c), (s, n) \in SDS$ such that $c \neq n$. In that case, the accuracy for the same sequence will be measured twice, and added to the sum.

For example, consider the following $SDS = \{(The, 4050), (t'h'e', 31688)\}$ (a subset of SDS_{Dest} appearing in Table 8.2). The ground summary is computed by Eq. 8.7, which in turn relies on Eq. 8.6:

$$
\begin{aligned}
Ground(SDS) &= \bigcup_{(s,c)\in SDS} gts(s,c) && \text{Eq. 8.7}\\
&= gts(The, 4050) \cup gts(t'h'e', 31688)\\
&= gts(The, 4050) \cup \{(the, 3961), (\mathbf{The, 3961}), \ldots, (THE, 3961)\}\\
&= \{(\mathbf{The, 4050})\} \cup \{(the, 3961), (\mathbf{The, 3961}), \ldots, (THE, 3961)\}
\end{aligned}
$$

The last two lines demonstrate the problem. The first of the lines shows the expansion of $gts(t'h'e')$ into the ground summary tuple set. The general sequence *t'h'e* in SDS is replaced in $Ground(SDS)$ by every ground sequence generated from it, i.e., every sequence in $Leaves(t'h'e')$. Each of these is given a count of $\frac{31688}{|Leaves(t'h'e')|} = \frac{31688}{8} = 3961$. In particular, a summary tuple is created for the sequence *The*: $(The, 3961)$. The last line shows that a second summary was generated, from the ground tuple $(The, 4050)$. Thus $Ground(SDS)$ contains two elements for the same sequence, with two different counts, making the value of $\psi(SDS)$ incorrect if we simply sum ψ_1 of the SDS elements.

Given multiple predictions for the same sequence, both implied by the same SDS (i.e., both are members of $Ground(SDS)$), we should prefer the element with the greater accuracy Thus before computing the sum in Eq. 8.9 directly, we collect the *maximally accurate* value for each sequence s.

We do this by further filtering the $Ground(SDS)$ set, to remove members that are *duplicate*, in the sense that they offer additional counts for the same sequence. We leave only the most accurate counts.

$$
Minimal(SDS, D) := \{(s,c) \in Ground(SDS) | c = \operatorname*{argmax}_x \psi_1(s, x, D)\} \qquad (8.8)
$$

We can now make a slight change to the definition of *psi*. Instead of summing the counts from the ground SDS it should sum the counts from the *minimal* SDS

(Eq. 8.8):

$$\psi(SDS, D) := \sum_{(s_i,c_i)\in \textbf{Minimal}(\textbf{SDS},\textbf{D})} \psi_1(s_i, c_i, D) \tag{8.9}$$

8.3 Optimizing Accuracy, Coverage and Compactness

Given a database D, the quality of a *Summarized Data Set* of D is measured by three factors discussed in Section 8.2:

- Its *compactness* (the size of the description, described in Section 8.2.1)
- Its *relative coverage* (the percent of sequences in D which are summarized, as described in Section 8.2.2)
- Its *accuracy* (its ability to correctly summarize the number of appearances in D, for each sequence in the summary, as described in Section 8.2.3)

The three factors are somewhat independent of each other. The ground-truth summary SDS_{GT} has perfect relative coverage, and perfect accuracy, but worst compactness (the size of D, worst case). The general summary SDS_g has perfect coverage, bad accuracy (in general), and perfect compactness. Any proper subset of the ground-truth summary can have good compactness, but will lose coverage, and likely have imperfect accuracy. Table 8.5 shows examples of the three measures for different possible summaries, all for the dataset shown in Figure 8-1a.

We are therefore interested in balancing the three factors, in a manner that allows explicit trading the achievement of one factor for improvements in the others. For example, one may with to ask for an SDS_{Dest} that has limited compactness (i.e., K is bounded), and achieves the maximum accuracy and coverage possible. Or one may ask for an SDS_{Dest} that maintains accuracy and coverage above some a threshold, and ask for the minimal K that is able to do so. This balancing of the three factors can be illustrated as a triangle (Figure 8-2).

Table 8.5: Examples of summaries and their various quality measures. K is *Compactness*. For all, $|D| = 31668$.

SDS_{Dest}	K	$RC(SDS_{Dest}, D)$	Ground(SDS)	$\psi(SDS, D)$
(The, 4059)	1	$\frac{4059}{31668} = 0.1281$	$\{(The, 4059)\}$	$\frac{\max(0, 8118 - 4059)}{31668} = 0.1281$
(The, 4059), (the,3915)	2	$\frac{7974}{31668} = 0.251$	$\{(The, 4059)\} \bigcup \{(the, 3915)\}$	$\frac{\max(0, 8118 - 4059)}{31668} + \frac{\max(0, 7830 - 3915)}{31668} =$ **0.251**
(The, 4059), (t'he,7974)	2	$\frac{7974}{31668} = 0.251$	$\{(The, 4059)\} \bigcup \{(The, 3987), (the, 3987)\} =$ $\{(The, 4059), (the, 3987)\}$	$\frac{\max(0, 8118 - 4059)}{31668} + \frac{\max(0, 7830 - 3987)}{31668} =$ $\frac{7902}{31668} =$ **0.249**
(the, 3915), (t'he,7974)	2	$\frac{7974}{31668} = 0.251$	$\{(the, 3915)\} \bigcup \{(The, 3987), (the, 3987)\} =$ $\{(The, 3987), (the, 3915)\}$	$\frac{3987}{31668} + \frac{\max(0, 7830 - 3915)}{31668} + =$ $\frac{7902}{31668} =$ **0.249**
(t'h'e',31668)	1	$\frac{31668}{31668} = 1$	$\{(The, 3961), (the, 3961),$ $(ThE, 3961), (THe, 3961),$ $(thE, 3961), (tHe, 3961),$ $(THE, 3961), (tHE, 3961)\}$	$\frac{3961}{31668} + \frac{\max(0, 7830 - 3961)}{31668} +$ $\frac{3961}{31668} + \frac{3961}{31668} +$ $\frac{\max(0, 7708 - 3961)}{31668} + \frac{\max(0, 7762 - 3961)}{31668} +$ $\frac{3961}{31668} + \frac{\max(0, 7882 - 3961)}{31668} =$ $\frac{31182}{31668} =$ **0.984**

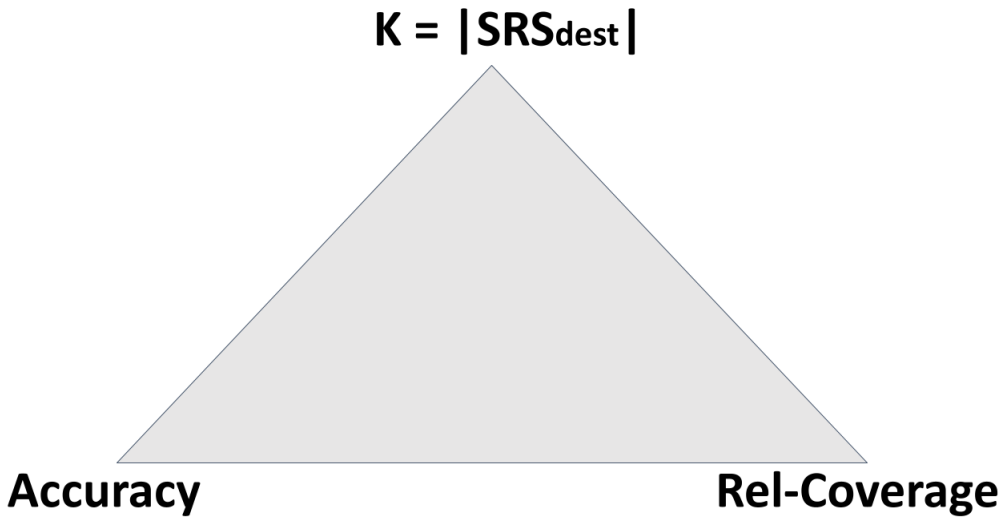


Figure 8-2: The three parameter we need to balance when searching for a good SDS.

Balancing Accuracy and K Given as an input the database D, one can can compute the ground truth SDS_{GT} which represents an ideal summary, from the point of view of accuracy and relative coverage (i.e., for both accuracy and coverage its score is 1). However, its size K, in the worst case, is $|SDS_{GT}|$. This raises several computational problems involving selecting an SDS that balances the accuracy and K:

- We may fix $K \ll |SDS_{GT}|$ and optimize accuracy, a problem we referred to as the *Top-K Summary Problem*.
- Or we may set a target accuracy threshold, and minimize the summary size K. We refer to this problem as the *Minimal Accuracy Summary Problem*.

Chapter 9

The Top-K Summary Problem

We tackle the problem of finding an optimal bounded-size summary of the dataset, i.e., the *Top-K Summary Problem*. We are given a bound on the compactness $k \ll |SDS_{GT}|$. The task is find a summary SDS_{Dest}^{OPT} that which maximize accuracy, with compactness no worse than k. In other words, determine SDS_{Dest} such that (i) accuracy is maximized, and (ii) $|SDS_{Dest}| \leq k$.

9.1 Greedy Sequence Selection is Not Enough

One may think that a simple greedy approach may suffice to solve the Top-K Summary problem, by sorting SDS_{GT} according to each sequence's individual accuracy, and then selecting the Top K sequences. But this is not the case.

The underlying assumption in this procedure is that the accuracy value of the combination of sequences is correlated with the sum of individual-sequence accuracy values. In other words, the assumption is that by selecting K sequences with the highest accuracy individual values (thus maximizing the sum of accuracy values), we will also be maximizing the accuracy of the combined K-sequence summary.

Unfortunately, this approach does not work in general. In the special case where the K sequences have no overlaps (i.e., no sequence among the K is a generalization of another), the accuracy of the combined summary is indeed a sum of the individual sequence accuracy values. But as we saw earlier, when one sequence is a generalization

of another, their combined accuracy is not a sum of their individual accuracies, due to the use of max in the maximal union ($\uplus$) operator used in the definition of the $Ground(SDS)$ set.

9.2 Solving the Top-K Summary Problem

The Top-K Problem is *almost* a 0-1 Knapsack Problem: We are given a finite number of items (sequences) to select from, each with weight 1. Each sequence may or may not be selected for the knapsack (0-1 constraint), with no duplicates (each sequence is different). The total weight of the sequences selected is bounded by the knapsack capacity K. We wish to select the K sequences which together maximize the profit—the accuracy of the selected set.

However, differently from the familiar definition of the 0-1 Knapsack problem, the objective function (the profit) is not a simple sum of the individual sequence accuracy, as we have discussed in the previous section. Instead, the accuracy of K sequencs is a monotonically non-decreasing function, which is submodular.

It is easy to see that the variant, is at least as hard as the 0-1 Knapsack Problem, which is known to be *NP-Complete*. [43] This is because the case where the sequences do not overlap, a special case of the *top-K problem*, is precisely a 0-1 Knapsack problem.

This type of variant knapsack problem has been studied directly by different authors, mostly in efforts to find useful approximation algorithms. A key result is presented by Sviridenko [76], who shows that a previously published algorithm by Khuller *et al.* [40] can find an approximate solution for this variant in time $O(n^5)$, with a performance guarantee of $(1 - e^{-1})$, which is optimal.

Tight integration of hierarchical sequence counting and top-K summary selection. The pipeline of the mining process can be found in Figure 9-1 on page 103. We first apply the LASH algorithm [6] to generate counting of the hierarchical sequences. The results are used to solve the Top-K problem, e.g., by using the approximation

algorithm discussed above.

Figure 9-1: Pipeline of the mining process, rectangles represent processes while rhombuses represent data.

A reasonable question that might be asked is to combine the counting and the mining task to one with one algorithm. Yu *et al.* [86] a *streaming* knapsack algorithm intended for knapsack problems with monotone submodular objective function. The incorporation of a streaming algorithm should allow, in principle, online selection of sequences even while the LASH main algorithm is generating their counts. Likewise, the use of interactive submodular optimization [27] is a different approach towards the same goal. We envision an active-learning component selecting sequences to be counted, to accelerate the creation of the set of K sequences.

Chapter 10

Discussion and Conclusions

In Part I of the thesis, we relies on the human domain expert to interact in the behavior modeling process, by infusing the machine learning problem definitions with domain-dependent insights, with respect to the ordering of features, the recognition of cyclic features, the distinction of goals and visual affirmation of clusters in the data, etc. In Part II, the human domain-expert is involved in the definition of a symbol generalization hierarchy—its taxonomy.

The two parts tackle different challenges within the overall task of behavior modeling from logs. The first part tackles the challenge of associating the modeled agent's beliefs and goals with its decision on the next action (in its continuous parameterized form) to be taken by the agent. It thus focuses on a single future action, in great detail. In contrast, second part sacrifices detail to attempt to discover abstract repeating patterns, which may generate insight into multi-step decision-making processes that the agent employs. Put another way, Part I examines the agent as if it were completely reactive, while Part II seeks to discover underlying multi-step plans that are the result of a deliberative decision-making process.

We hope the combination of the techniques from the two parts will lead towards the capacity for building fuller agent behavior models from logs of actions and environment settings. In particular, immediate steps to consider are the identification of beliefs and goals that would lead an agent to select a multi-step plan, discovery of beliefs that cause the agent to interrupt a plan, or to choose between different

variations on it, etc.

Consider the following example. Suppose that the hierarchical sequence miner discovers two sequences with an identical prefix: $(Kick\#100\#30; Turn\#90)$ appearing 300 times, and $(Kick\#100\#30; TurnNeck\#270)$ appearing 100 times. The techniques in Part I may be able to predict the first action $Kick\#100\#30$. The prior probabilities for the second action will then be 0.75 for $Turn\#90$, and 0.25 for $TurnNeck\#270$, even if no prediction is possible based on the agents inferred beliefs or goal.

Several research thrusts for future work are relevant. Chief among them would be identifying a solution for the *Minimal Accuracy Summary* problem, i.e., the problem of finding the smallest-size SDS whose accuracy satisfies a required threshold. We believe a very useful lead towards this goal is presented by Iyer and Bilmes [36]. They had shown a close relation between the two problems, which they term *Submodular Cost Submodular Knapsack* (which is analogous to the top-K problem), and *Submodular Cost Submodular Cover* (which is analogous to the minimal accuracy problem), and provide approximation algorithms for both.

We have also discussed briefly another important direction for future work; finding an efficient integration of the procedures for counting hierarchical sequences (currently achieved by relying on the LASH sequence miner [6]), and for selecting the target SDS either via top-K summary generation or by generating a satisficing summary. The separation of the two procedures in the current form is highly inefficient, as many sequences are counted only to be discarded later in the process.

Bibliography

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Proceedings of Advances in Neural Information Processing Systems 19*, 2007.

[2] P. H. Abreu, D. C. Silva, J. Portela, J. Mendes-Moreira, and L. P. Reis. Using Model-Based Collaborative Filtering Techniques to recommend the Expected Best Strategy to Defeat a Simulated Soccer Opponent. *Intelligent Data Analysis*, 18(5):973–991, 2014.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Comput. Soc. Press.

[4] R. Aler, O. Garcia, and J. M. Valls. Correcting and Improving Imitation Models of Humans for Robosoccer Agents. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2402–2409. IEEE, 2005.

[5] J. A. Bagnell. An Invitation to Imitation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 2015.

[6] K. Beedkar and R. Gemulla. LASH: Large-Scale Sequence Mining with Hierarchies. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 491–503, 2015.

[7] I. Bratko, T. Urbančič, and C. Sammut. Behavioural Cloning: Phenomena, Results and Problems. *IFAC Proceedings Volumes*, 28(21):143–149, 1995.

[8] L. Breiman. Bagging Predictors. *Machine learning*, 24(2):123–140, 1996.

[9] L. Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001.

[10] D. Brugali. Model-Driven Software Engineering in Robotics: Models are Designed to Use the Relevant Things, Thereby Reducing the Complexity and Cost in the Field of Robotics. *IEEE Robotics & Automation Magazine*, 22(3):155–166, 2015.

[11] D. Brugali and A. Shakhimardanov. Component-Based Robotic Engineering (part ii). *IEEE Robotics & Automation Magazine*, 17(1):100–112, 2010.

[12] S. Calinon, F. Guenter, and A. Billard. On Learning, Representing, and Generalizing A Task in A Humanoid Robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

[13] L. Cao, G. Weiss, and P. S. Yu. A Brief Introduction to Agent Mining. *Autonomous Agents and Multi-Agent Systems*, 25(3):419–424, Nov. 2012.

[14] M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, et al. Robocup Soccer Server, 2001.

[15] T. Chen and C. Guestrin. Xgboost: A Scalable Tree Boosting System. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[16] J. Comptdaer, E. Chiva, S. Delorme, H. Morlaye, J. Volpoët, and O. Balet. Multi-Scale Behavioral Models for Urban Crisis Training Simulation. In *Proceedings of 16th conference on behavior representation in modeling and simulation (BRIMS)*, 2007.

[17] M. Dastani. A survey of multi-agent programming languages and frameworks. In *Agent-Oriented Software Engineering*, pages 213–233. Springer, 2014.

[18] K. Fischer, C. Hahn, and C. Madrigal-Mora. Agent-Oriented Software Engineering: A Model-Driven Approach. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):334–369, 2007.

[19] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A Survey of Sequential Pattern Mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[20] Y. Freund and R. E. Schapire. A Desicion-Theoretic Generalization of On-Line Learningand An Application to Boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[21] N. Fridman and G. A. Kaminka. Towards A Cognitive Model of Crowd Behavior Based on Social Comparison Theory. In *AAAI*, pages 731–737, 2007.

[22] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of statistics*, pages 1189–1232, 2001.

[23] J. H. Friedman. Stochastic Gradient Boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[24] P. Geurts, D. Ernst, and L. Wehenkel. Extremely Randomized Trees. *Machine learning*, 63(1):3–42, 2006.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[26] D. H. Grollman and O. C. Jenkins. Incremental Learning of Subtasks from Unsegmented Demonstration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 261–266. IEEE, 2010.

[27] A. Guillory and J. Bilmes. Interactive Submodular Set Cover. In *ICML*, 2010.

[28] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining Top-K Frequent Closed Patterns Without Minimum Support. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 211–218. IEEE, 2002.

[29] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

[30] M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467, 2008.

[31] J. Ho and S. Ermon. Generative Adversarial Imitation Learning. *arXiv:1606.03476 [cs]*, June 2016.

[32] Y. Horman and G. A. Kaminka. Removing Biases in Unsupervised Learning of Sequential Patterns. *Intelligent Data Analysis*, 11(5):457–480, 2007.

[33] Z. Huang, Y. Yang, and X. Chen. An Approach to Plan Recognition and Retrieval for Multi-Agent Systems. In *Workshop on Adaptability in Multi-Agent Systems, First RoboCup Australian Open*, volume 47, 2003.

[34] J. A. Iglesias, A. Ledezma, A. Sanchis, and G. A. Kaminka. A Plan Classifier Based on Chi-Square Distribution Tests. *Intelligent Data Analysis*, 15(2):131–149, 2011.

[35] Y. A. Ivanov and A. F. Bobick. Recognition of Visual Activities and Interactions by Stochastic Parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):852–872, 2000.

[36] R. Iyer and J. Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints, Nov. 2013. arXiv:1311.2106 [cs].

[37] G. A. Kaminka, M. Fidanboylu, A. Chang, and M. M. Veloso. Learning the Sequential Coordinated Behavior of Teams from Observations. In *Robot Soccer World Cup*, pages 111–125. Springer, 2002.

[38] G. A. Kaminka and N. Fridman. Simulating Urban Pedestrian Crowds of Different Cultures. *ACM Transactions on Intelligent Systems and Technology*, 9(3):27:1–27:27, 2018.

[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in neural information processing systems*, pages 3146–3154, 2017.

[40] S. Khuller, A. Moss, and J. Naor. The Budgeted Maximum Coverage Problem. *Information Processing Letters*, 70:39–45, 1999.

[41] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous Helicopter Flight via Reinforcement Learning. In *Proceedings of Advances in Neural Information Processing Systems 16*, 2003.

[42] K. M. Kitani, Y. Sato, and A. Sugimoto. Recovering the Basic Structure of Human Activities from Noisy Video-Based Symbol Strings. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(08):1621–1646, 2008.

[43] M. G. Lagoudakis. The 0 – 1 Knapsack Problem An Introductory Survey. 1996.

[44] R. Le Hy, A. Arrigoni, P. Bessière, and O. Lebeltel. Teaching Bayesian Behaviours to Video Game Characters. *Robotics and Autonomous Systems*, 47(2-3):177–185, 2004.

[45] K. Lee and Y. Demiris. Towards Incremental Learning of Task-Dependent Action Sequences using Probabilistic Parsing. In *2011 IEEE International Conference on Development and Learning (ICDL)*, volume 2, pages 1–6. IEEE, 2011.

[46] M. A. Leece and A. Jhala. Sequential Pattern Mining in Starcraft: Brood War for Short and Long-Term Goals. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[47] S. Levine and V. Koltun. Continuous Inverse Optimal Control with Locally Optimal Examples. *arXiv preprint arXiv:1206.4617*, 2012.

[48] N. R. Mabroukeh and C. I. Ezeife. A Taxonomy of Sequential Pattern Mining Algorithms. *ACM Computing Surveys (CSUR)*, 43(1):1–41, 2010.

[49] L. McInnes, J. Healy, and S. Astels. HDNSCAN: Hierarchical Density Based Clustering. *The Journal of Open Source Software*, 2(11), mar 2017.

[50] H. Mellmann, B. Schlotter, and C. Blum. Simulation Based Selection of Actions for a Humanoid Soccer-Robot. In S. Behnke, R. Sheh, S. Sarıel, and D. D. Lee, editors, *RoboCup 2016: Robot World Cup XX*, volume 9776, pages 193–205. Springer International Publishing, Cham, 2017.

[51] P. Moylan and B. Anderson. Nonlinear Regulator Theory and an Inverse Optimal Control Problem. *IEEE Transactions on Automatic Control*, 18(5):460–465, 1973.

[52] C. G. Nevill-Manning and I. H. Witten. Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[53] A. S. Ogale, A. Karapurkar, and Y. Aloimonos. View-Invariant Modeling and Recognition of Human Actions using Grammars. In *Dynamical vision*, pages 115–126. Springer, 2006.

[54] I.-S. Oh, H. Cho, and K.-J. Kim. Playing Real-Time Strategy Games by Imitating Human Players' Micromanagement Skills Based on Spatial Analysis. *Expert Systems with Applications*, 71:192–205, Apr. 2017.

[55] L. Padgham, J. Thangarajah, and M. Winikoff. Prometheus Research Directions. In *Agent-Oriented Software Engineering*, pages 155–171. Springer, 2014.

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

[57] M. Plantevit, A. Laurent, and M. Teisseire. Hype: Mining Hierarchical Sequential Patterns. In *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP*, pages 19–26, 2006.

[58] D. A. Pomerleau. Alvinn: An Autonomous Land Vehicle in a Neural Network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[59] L. Rabiner and B. Juang. An Introduction to Hidden Markov Models. *ieee assp magazine*, 3(1):4–16, 1986.

[60] A. Richardson, G. A. Kaminka, and S. Kraus. REEF: Resolving Length Bias in Frequent Sequence Mining. *International Journal On Advances in Intelligent Systems*, 7(1–2):208—-222, 2014.

[61] G. Ridgeway. Generalized Boosted Models: A Guide to the gbm Package. *Update*, 1(1), 2007.

[62] 2D Robocup Soccer Simulation Screenshot. www.robocup.org/leagues/24.

[63] S. Ross and J. A. Bagnell. Efficient Reductions for Imitation Learning. In *AISTATS*, 2010.

[64] S. Russell. Learning Agents for Uncertain Environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[65] M. S. Ryoo and J. K. Aggarwal. Robust Human-Computer Interaction System Guiding a User by Providing Feedback. In *IJCAI*, pages 2850–2855, 2007.

[66] I. Sagredo-Olivenza, P. P. Gómez-Martín, M. A. Gómez-Martín, and P. A. González-Calero. Trained Behavior Trees: Programming by Demonstration to Support AI Game Designers. *IEEE Transactions on Games*, 11:5–14, 2019.

[67] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning To Fly. In *Machine Learning Proceedings 1992*, pages 385–393. Elsevier, 1992.

[68] Y. Schroecker and C. L. Isbell. State Aware Imitation Learning. In *Advances in Neural Information Processing Systems*, pages 2911–2920, 2017.

[69] Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, Mar. 1993.

[70] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-play. *Science*, 362(6419):1140–1144, 2018.

[71] Z. Solan, D. Horn, E. Ruppin, and S. Edelman. Unsupervised Learning of Natural Languages. *Proceedings of the National Academy of Sciences*, 102(33):11629–11634, 2005.

[72] A. Stolcke and S. Omohundro. Inducing Probabilistic Grammars by Bayesian Model Merging. In *International Colloquium on Grammatical Inference*, pages 106–118. Springer, 1994.

[73] A. Sturm and O. Shehory. Agent-Oriented Software Engineering: Revisiting the State of the Art. In *Agent-Oriented Software Engineering*, pages 13–26. Springer, 2014.

[74] A. Sturm and O. Shehory. The Landscape of Agent-Oriented Methodologies. In *Agent-Oriented Software Engineering*, pages 137–154. Springer, 2014.

[75] K. Sullivan and S. Luke. Real-Time Training of Team Soccer Behaviors. In *Robot Soccer World Cup*, pages 356–367. Springer, 2012.

[76] M. Sviridenko. A Note on Maximizing a Submodular Set Function Subject to a Knapsack constraint. *Operations Research Letters*, 32(1):41–43, Jan. 2004.

[77] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent Agents for Interactive Simulation Environments. *AI Magazine*, 16(1), 1995.

[78] F. Torabi, G. Warnell, and P. Stone. Behavioral Cloning from Observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4950–4957, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization.

[79] J. Tsai, N. Fridman, E. Bowring, M. Brown, S. Epstein, G. A. Kaminka, S. Marsella, A. Ogden, I. Rika, A. Sheel, et al. ESCAPES: Evacuation Simulation with Children, Authorities, Parents, Emotions, and Social Comparison. In *AAMAS*, volume 11, pages 457–464, 2011.

[80] A. Ude, C. G. Atkeson, and M. Riley. Programming Full-Body Movements for Humanoid Robots by Observation. *Robotics and autonomous systems*, 47(2-3):93–108, 2004.

[81] U. Visser and H.-G. Weland. Using Online Learning to Analyze the Opponent's Behavior. In *Robot Soccer World Cup*, pages 78–93. Springer, 2002.

[82] G. Wallner. Sequential Analysis of Player Behavior. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, pages 349–358, 2015.

[83] M. Winikoff. Future Directions for Agent-Based Software Engineering. *International Journal of Agent-Oriented Software Engineering*, 3(4):402–410, 2009.

[84] P. R. Wurman, R. DÁndrea, and M. Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, Spring, 2008.

[85] C. Ye, J. R. Segedy, J. S. Kinnebrew, and G. Biswas. Learning Behavior Characterization with Multi-Feature, Hierarchical Activity Sequences. *International Educational Data Mining Society*, 2015.

[86] Q. Yu, E. L. Xu, and S. Cui. Submodular Maximization With Multi-Knapsack Constraints and its Applications in Scientific Literature Recommendations. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1295–1299, Dec. 2016.

Appendix A

High-Level View of Our Approach as a System

As part of the thesis work, we defined a new log format that will be accessible to humans and machines, to enable the algorithms to be tested on the data from *RoboCup2D* as well as in other domains. The log content reflects movements, actions, and states of various agents' environment and the environment state. It is important to find a format that does not require a particular storage platform.

In order to implement our approach, several steps need to be done. The flow of our approach is shown in Figure A-1 on page 114.

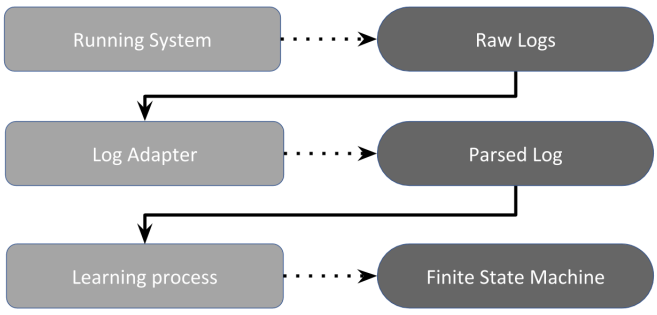


Figure A-1: Our approach implementation flow.

Running System: As a first step, we need to run the domain's original system as a demonstrator; the output of this step will be the raw log of the system, without

adherence to the file format. For example in our domain, *RoboCup2D*, the system outputs raw logs will be in formats *.rcg* and *.rcl*.

Log Adapter: The step input will be the raw logs from the running system, while it is output will be the parsed logs. The parsed logs will concentrate on two Goals: 1. The log format will be readable for humans and machines as well. 2. It will be robust for as many environments as possible.

Parsed Log: To handle the Goals mentioned above, we defined the base format, using *one line* record file. We designate the mandatory keys name and possible values to help our system handle a new domain easily. Each record contains the key *TimeStamp* (For example at *RoboCup2D* domain $TimeStamp \in [1 - 6,000,000]$ where the unit equivalent to millisecond), another necessary key is $RecordType \in \{AgentAction, AgentState, EnvState\}$ while the other possible keys in record will be derived by the *RecordType* value. *AgentAction* represents an action that the agent acts at each *TimeStamp*. *AgentState* represents the agent's knowledge of the world through his eyes including viewed objects/agents, position, velocity and any other feature that is related to the agent at some specific *TimeStamp*. *EnvState* represents the global environment state at some *TimeStamp*.

The Convert of the raw log to parsed log will need to handle some challenges; sometimes the raw logs are in a global view, so the *EnvState* is supplied while the *AgentState* will need to be inferred from the *EnvState*. This inference is based on the knowledge of the agent's sense. For example, the agent's range of view, figuring out which objects or agents were viewed by the learned agent. Also, there is a need to convert the viewed objects or agents' position and velocity from absolute measure to relative to the viewing agent's position.

Learning Process: This step input is the parsed log, while the output will be the FSM. We will try to learn this FSM with several AI techniques. First, we will learn the number of states using cluster techniques. Then we will learn the partial-policies of each execution state using supervised learning. Last we will learn the transition

between the execution states.

Appendix B

Clustering Algorithm Experiment Setup

We used the clustering technique with several algorithms. We used the following parameters in the different algorithms.

DBSCAN: eps: 4.5
min_sample: 512
algorithm: ball_tree
leaf_size: 60

HDBSCAN: min_sample: 512
algorithm: boruvka_balltree
leaf_size: 60
alpha: 4.5

MiniBatchKMeans: n_clusters: 4/5

Ward: n_clusters: 4

AgglomerativeClustering: n_clusters: 4

Birch: n_clusters: 3

GaussianMixture: n_components: 5/6/7

Optics: min_sample: 512

min_cluster_size: 0.1

Appendix C

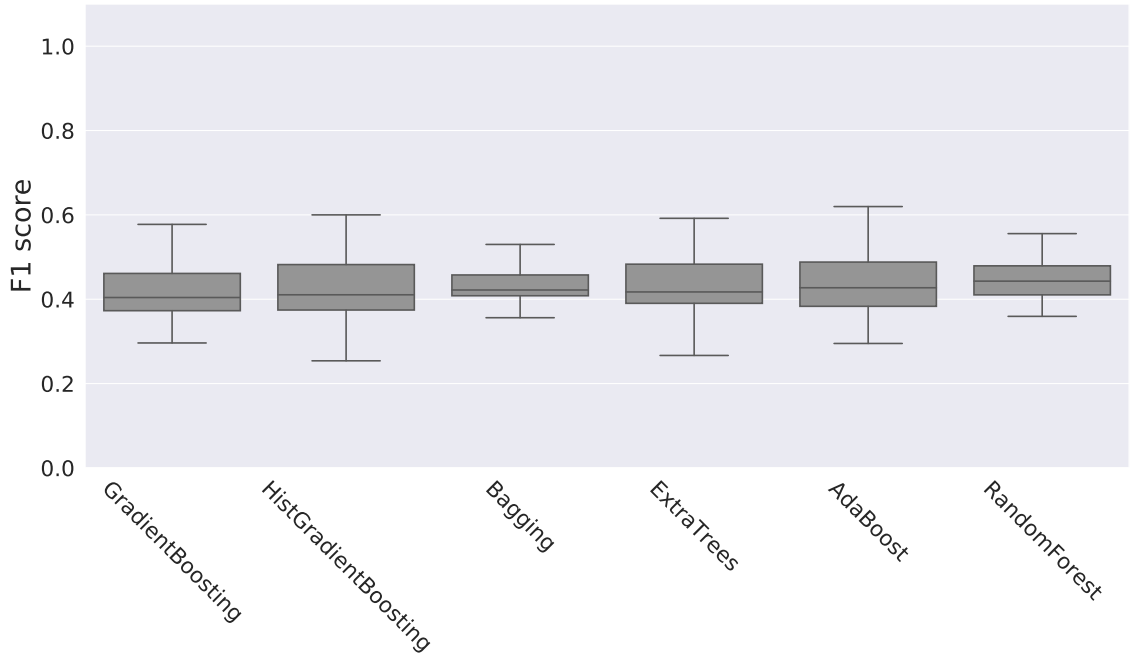
Additional Results for Feature Fast Forward

C.1 Tackle Action

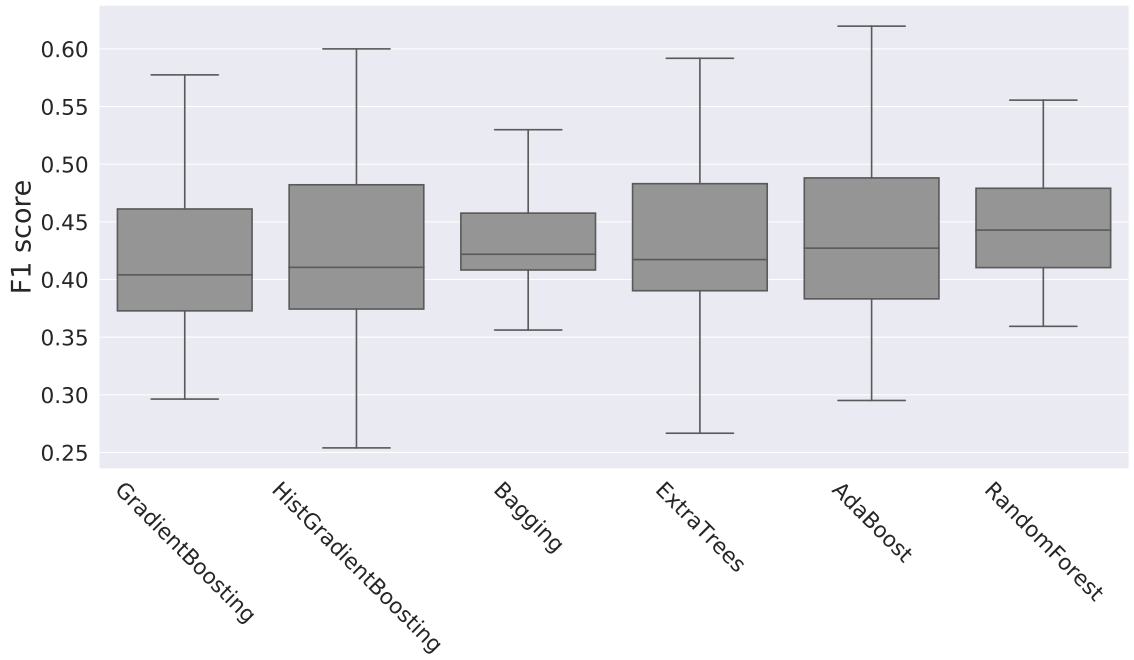
We tested the *fast forward* technique over the same data presented in Section 5.2, to improve the regression of tackle P1 and the classification of tackle P2. Initially, we received guidance from the system expert regarding significant results that were later caused by the tackle action; these helped us determine the agent's goal state (G_t^i).

To automate the clustering process, we looked for an off-the-shelf classification algorithm to automatically classify the manual cluster. As can be seen in the classification results shown in Figure C-1 on page 120 , the *RandomForestClassifier* earns the best results concerning all metrics and target clusters.

The final results of regression tackle P1 are shown in Figure 6-2 on page 61 and the classification tackle P2 appears in Figure 6-3 on page 64. As can be seen, the *fast forward* cluster did not improve significantly across all measurements. Also, as expected, the manual cluster (*FastForwardPerfect*) was more significant than the automatic cluster ($FastForwardAUTO$).



(a) F1 score range is from 0 to 1



(b) F1 score range is from 0.2 to 0.7 zoom into Figure C-1a

Figure C-1: Tackle fast forward automatic classification step, higher results are better.

Appendix D

Additional Results for Parameter Clustering

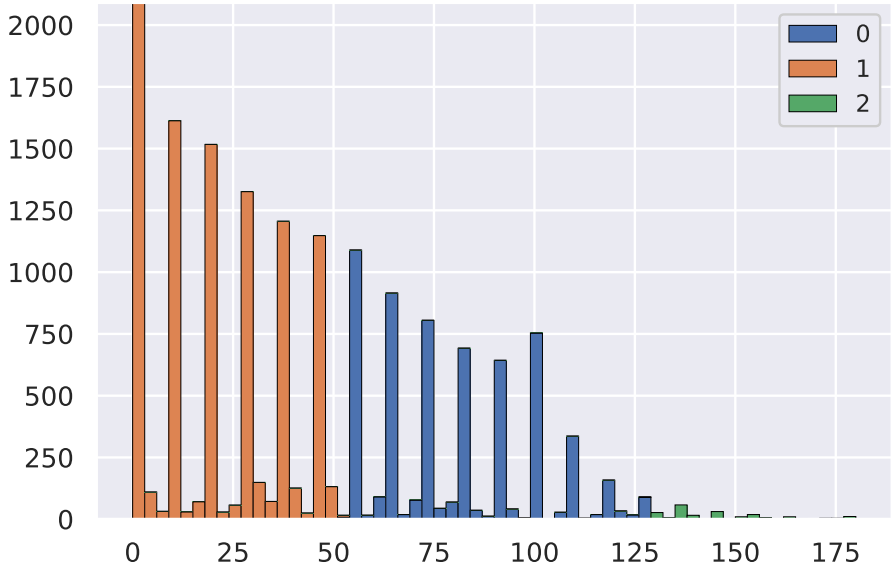
D.1 Tackle Action

We tested the parameter cluster technique over the same data noted in Section 5.3, to improve the regression of tackle P1 and the classification of tackle P2.

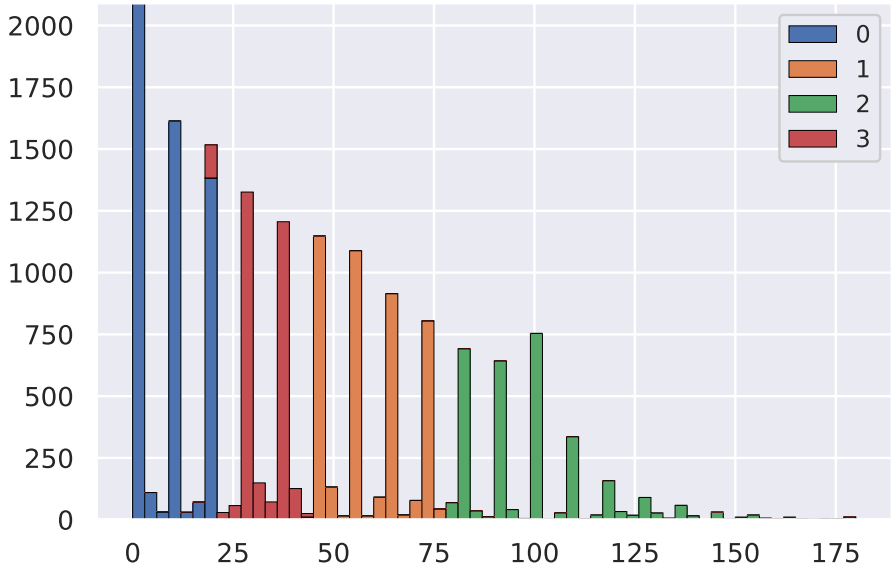
We chose to generalize the P1 parameter, ignoring the right/left direction based on the *mirror transform*. Moreover, because tackle P2 is *Boolean*, we used the *parameter clustering* technique based on P1 only.

We plotted P1 using manual clustering to find a good distribution cluster and produced the parameter's histograms using an off-the-shelf cluster algorithm such as: *MiniBatchKMeans, AffinityPropagation, Ward, AgglomerativeClustering, DBSCAN, Hdbscan, Optics, Birch, or GaussianMixture*. Based on the results of our attempts, as shown in Figure D-1 on page 122, we chose the *GaussianMixture* cluster algorithm (D-1d)

We validate that we chose the best cluster of the agent's goal state using the *parameter clustering* technique. We ran the regression over P1 and the classification over P2 by adding in each run, one of the clusters (whose plot is shown in Figure D-1) as a feature to determine the agent's goal state best *Parameter Clustering* algorithm. The results over P1 regression are shown in Figure D-2 on page 127 and

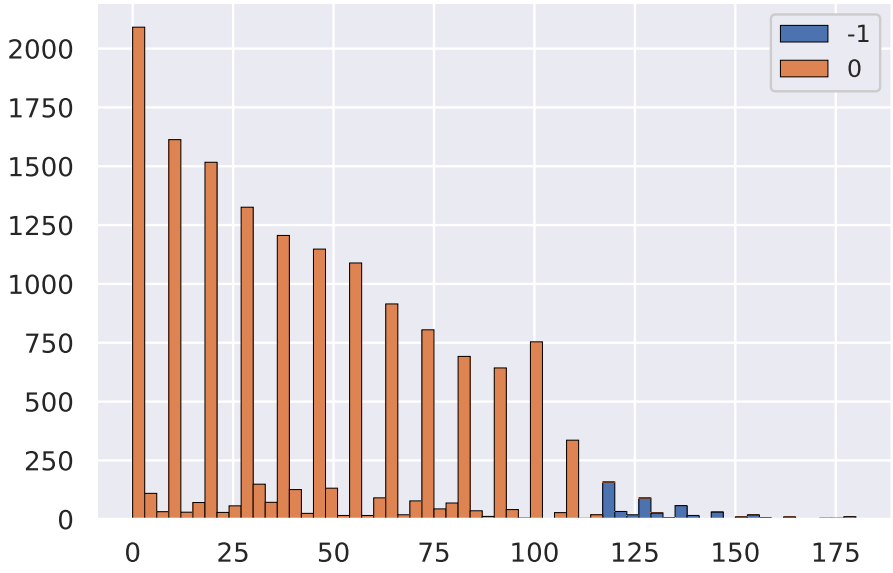


(a) Birch

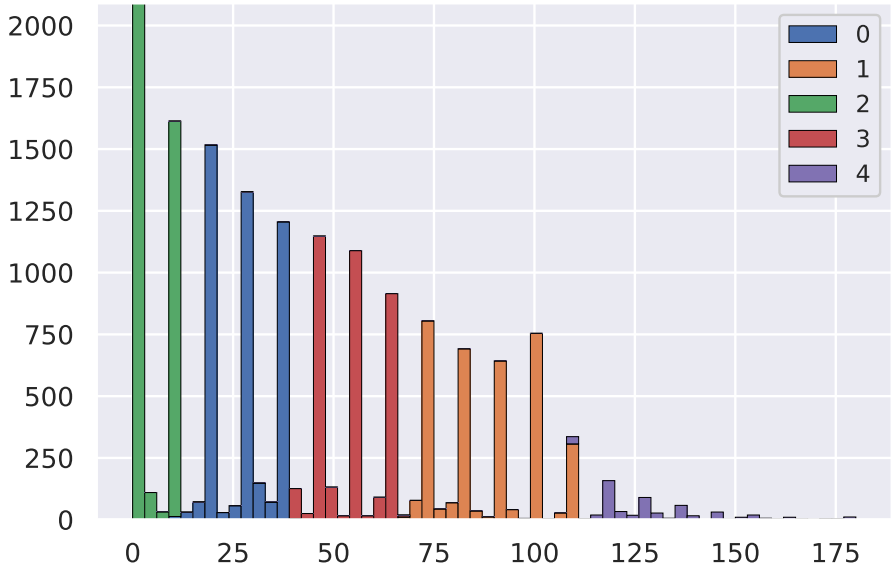


(b) MiniBatchKMeans

Figure D-1: Plots for tackle cluster algorithms



(c) DBSCAN



(d) GaussianMixture

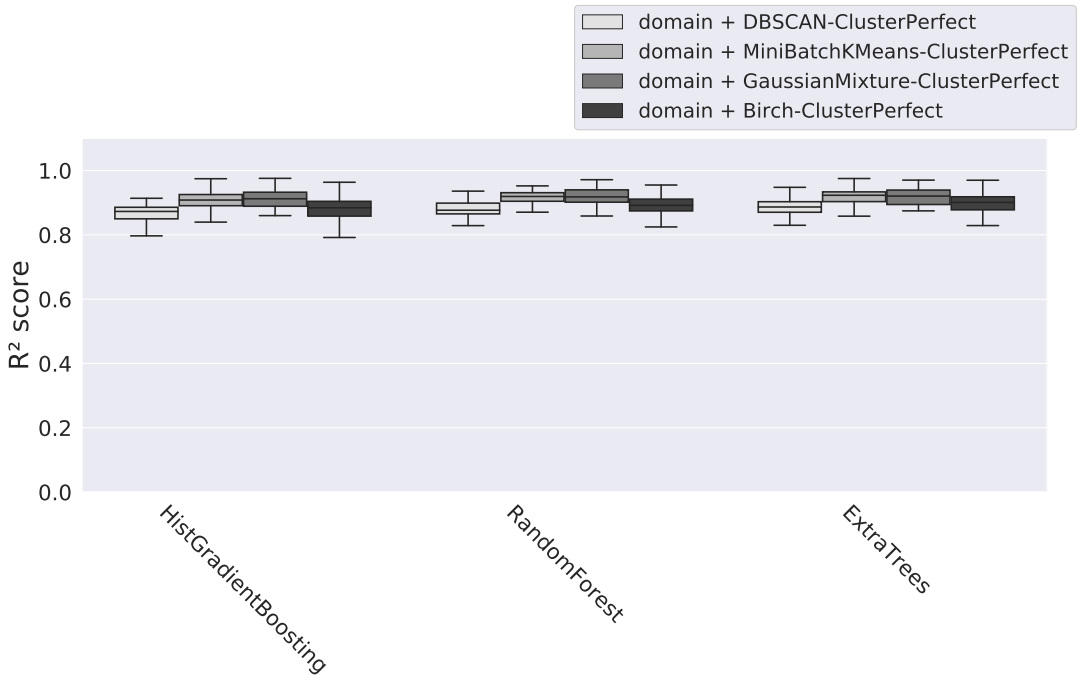
Figure D-1: Plots for tackle cluster algorithms

those for P2 classification are shown in Figure D-3 on page 130. As can be seen, the *GaussianMixture* showed the best results for almost all measurements.

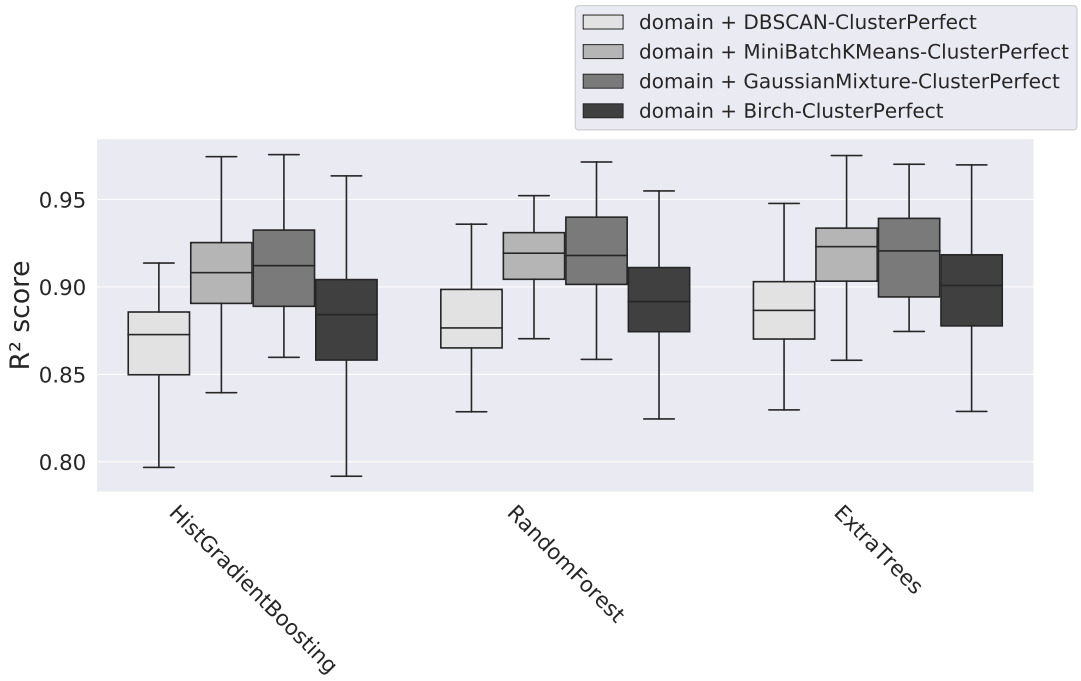
To automate the clustering process, we looked for an off-the-shelf algorithm to automate classification of the manual cluster. Based on the classification results shown in Figure D-4 on page 131, we can see that the *HistGradientBoostClassifier* earns the best results for all metrics and target clusters.

The final results over P1 regression are shown in Figure 6-2 on page 61 while the P2 classification can be found in Figure 6-3 on page 64. As can be seen, the *parameter clustering* improves the results for all measurements.

It is important to note that for the regression over tackle P1, the *parameter clustering* technique provided a more significant improvement than the *fast forward* technique. When it comes to the classification of tackle P2, the results show no advantage for any technique. As expected, the manual cluster (*CluserPerfect*) was more significant than the automatic cluster (*CluserAUTO*).

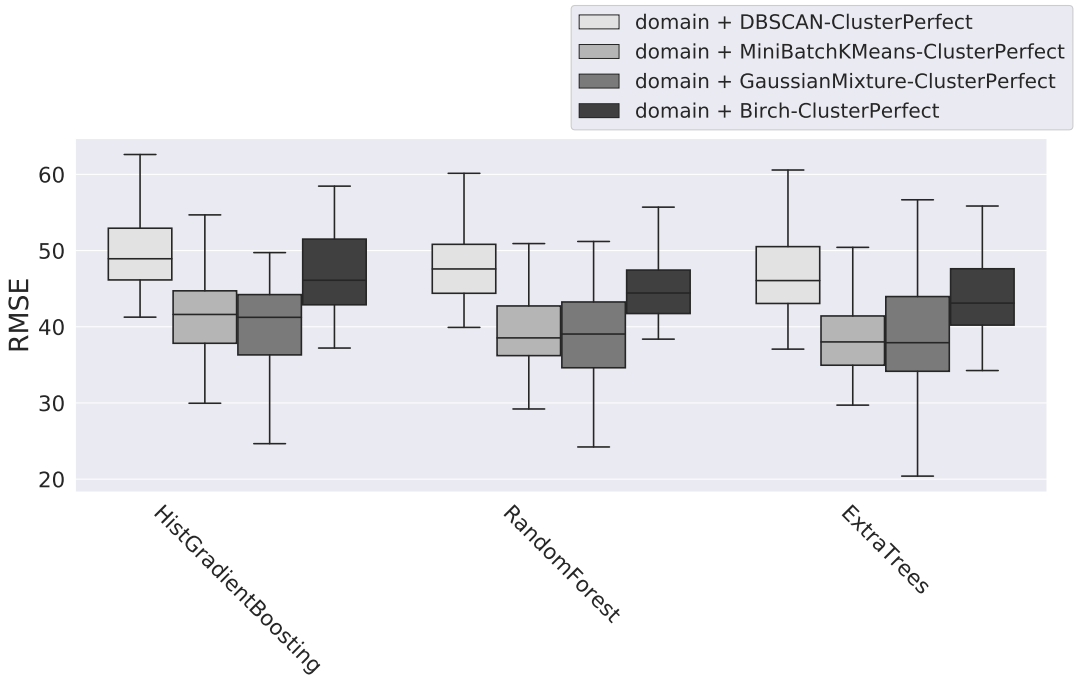


(a) R^2 score, higher results are better.

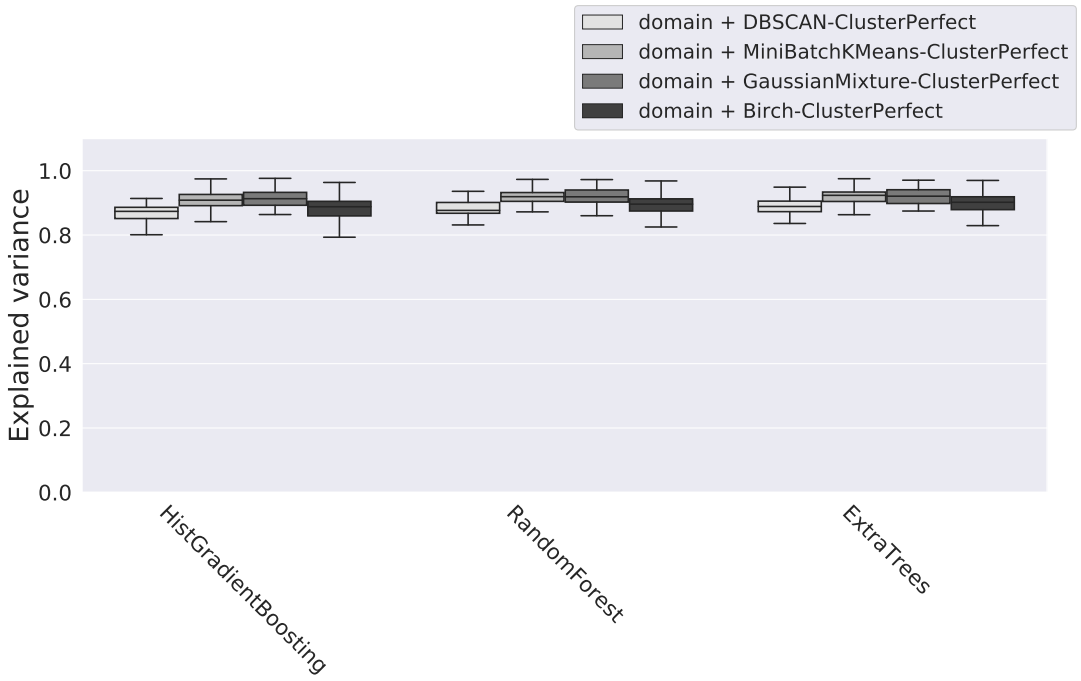


(b) R^2 score, higher results are better. Range is from 0.8 to 1 zoom of Figure D-2a

Figure D-2: Tackle regression over P1, comparison of ClusterPerfect

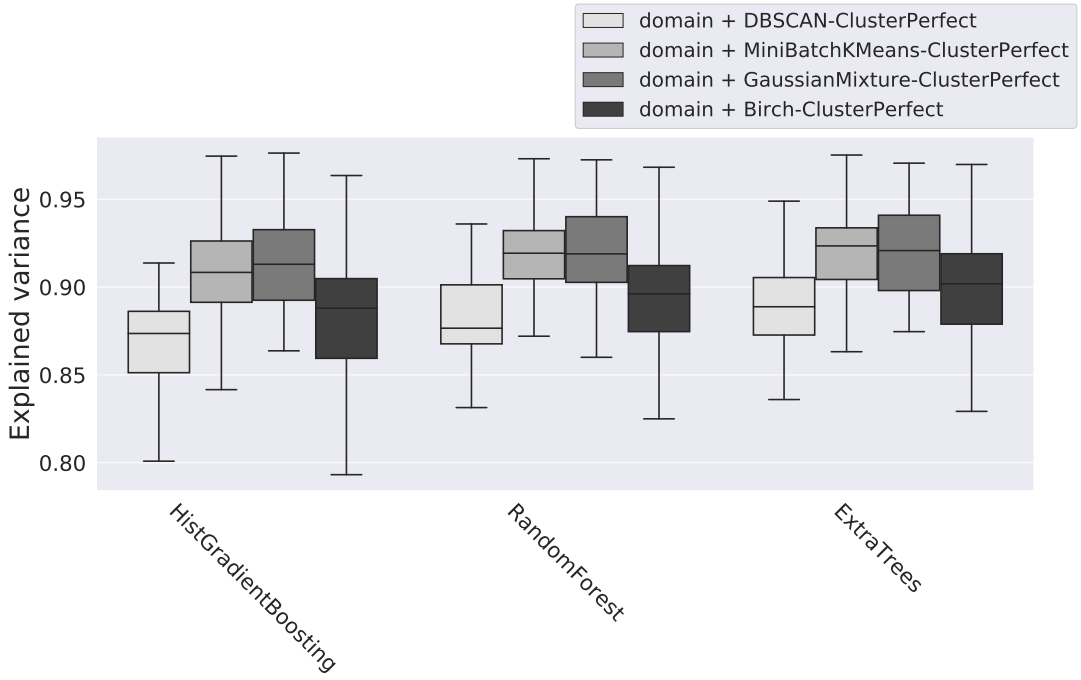


(c) RMSE, lower results are better.



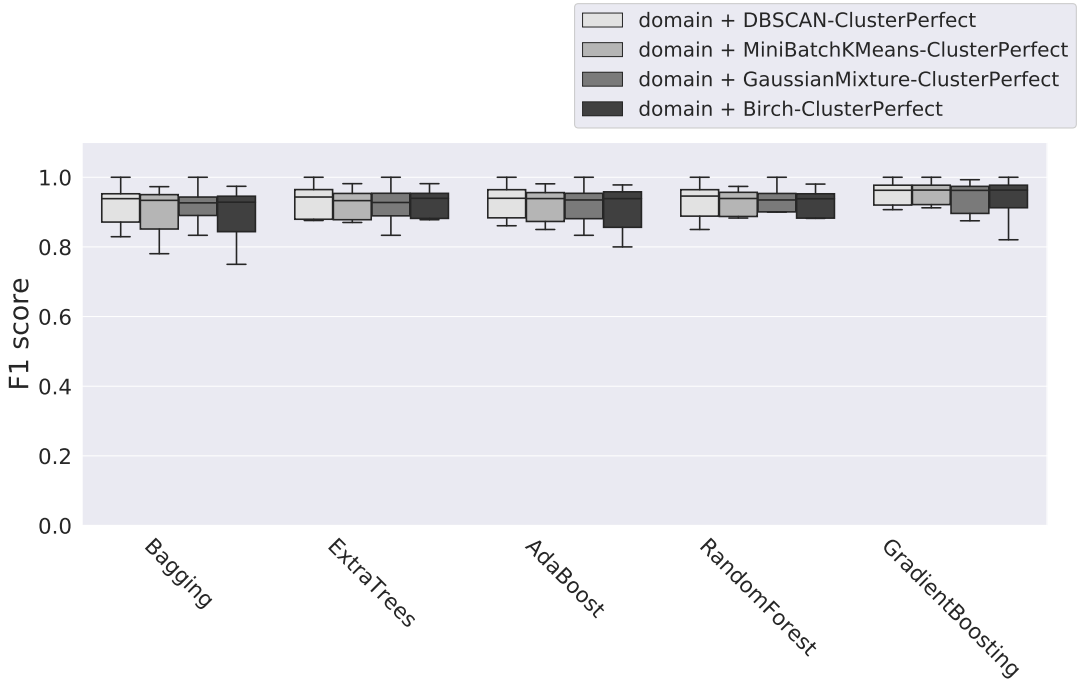
(d) Explained variance, higher results are better.

Figure D-2: Tackle regression over P1, comparison of ClusterPerfect

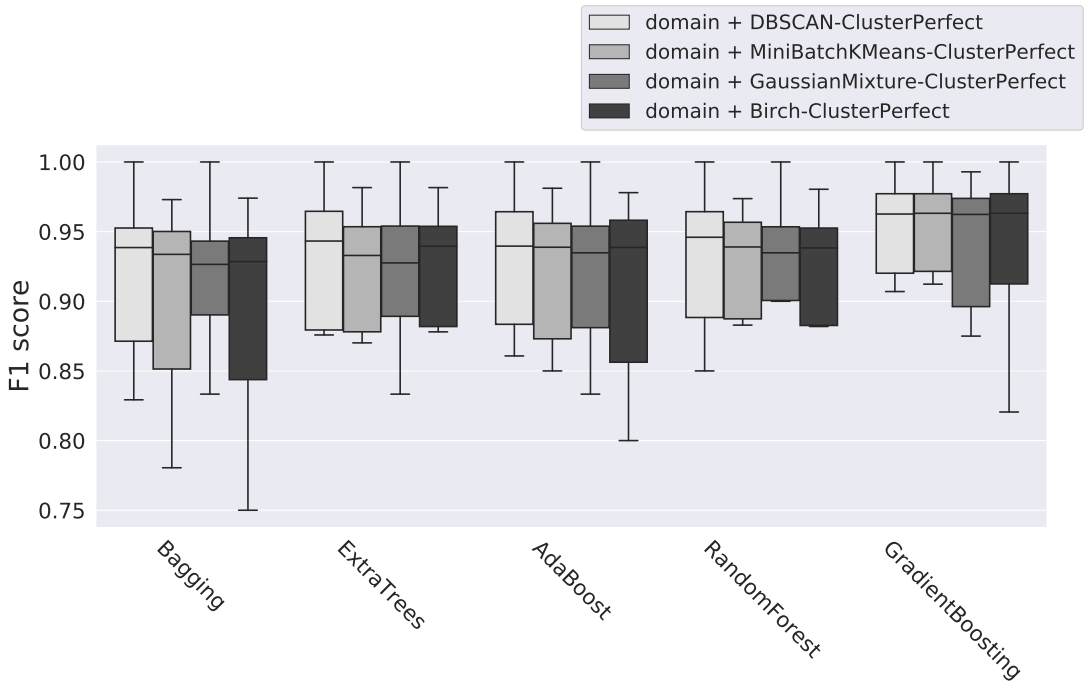


(e) Explained variance, higher results are better.Range is from 0.8 to 1 zoom of Figure D-2d

Figure D-2: Tackle regression over P1, comparison of ClusterPerfect

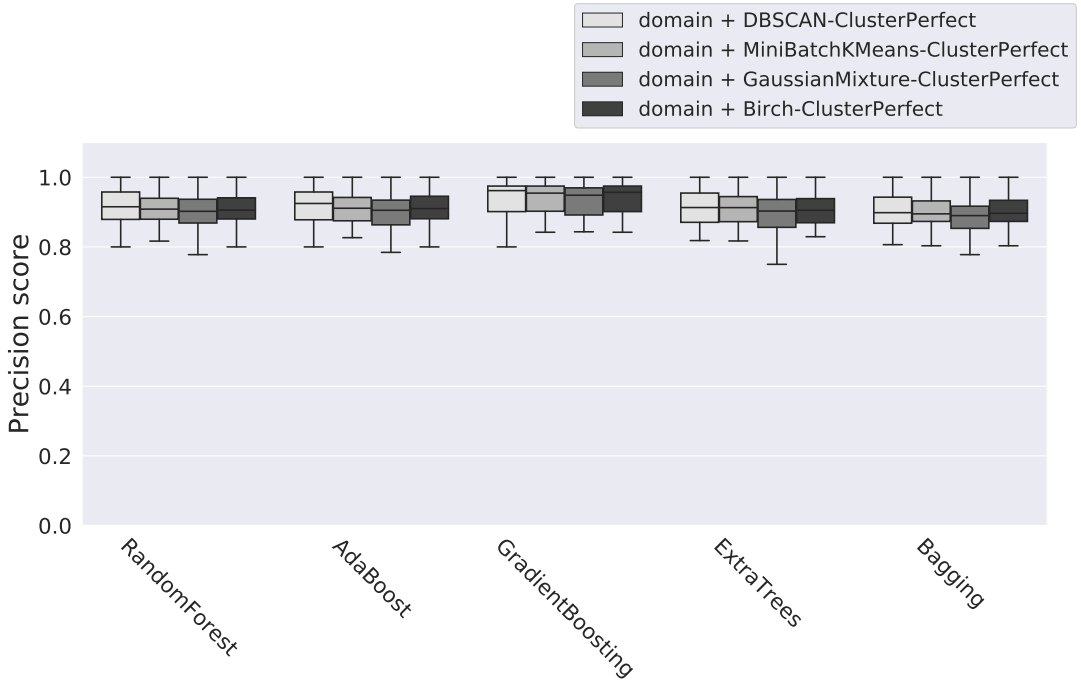


(a) F1 score, higher results are better.

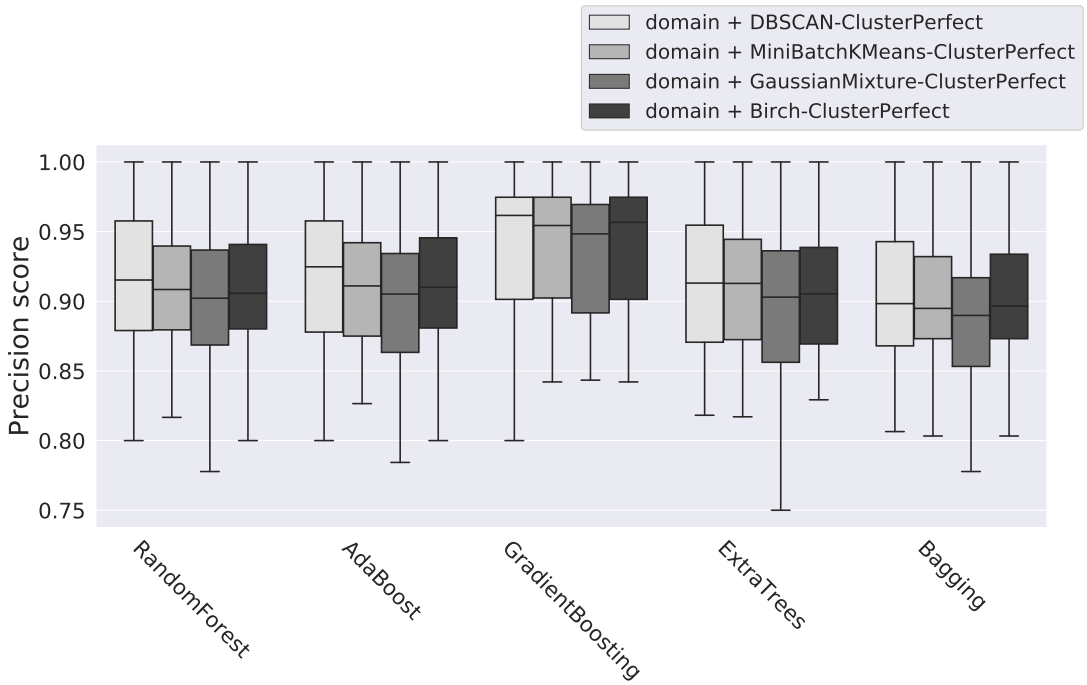


(b) F1 score, higher results are better. Range is from 0.75 to 1 zoom of Figure D-3a

Figure D-3: Tackle regression over P2, comparison of ClusterPerfect

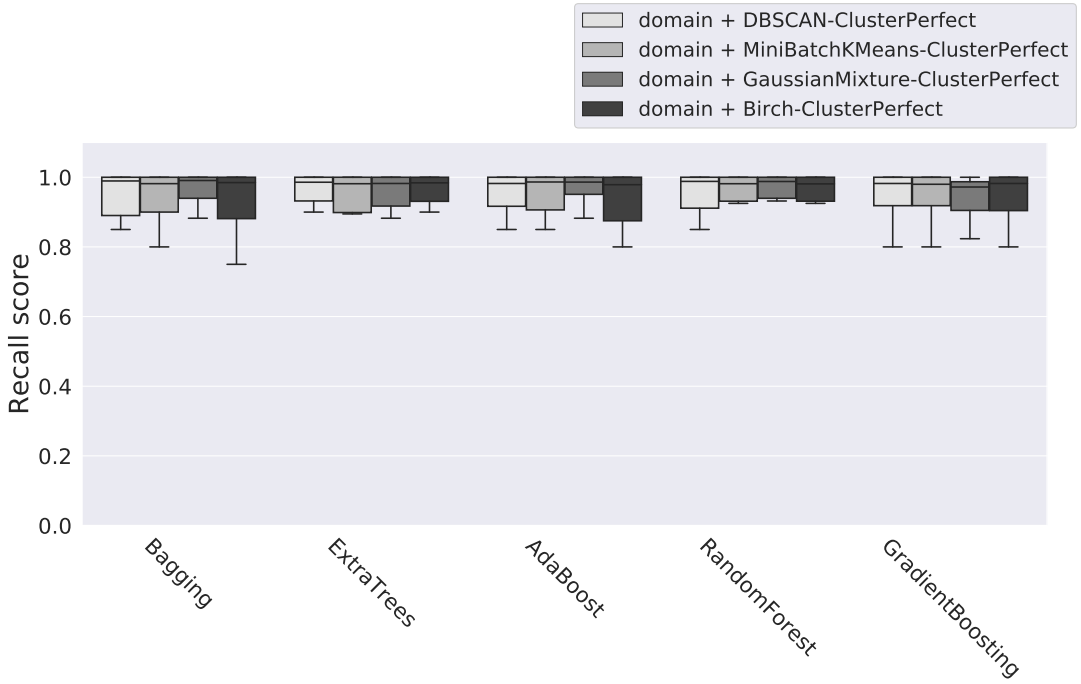


(c) Precision, higher results are better.

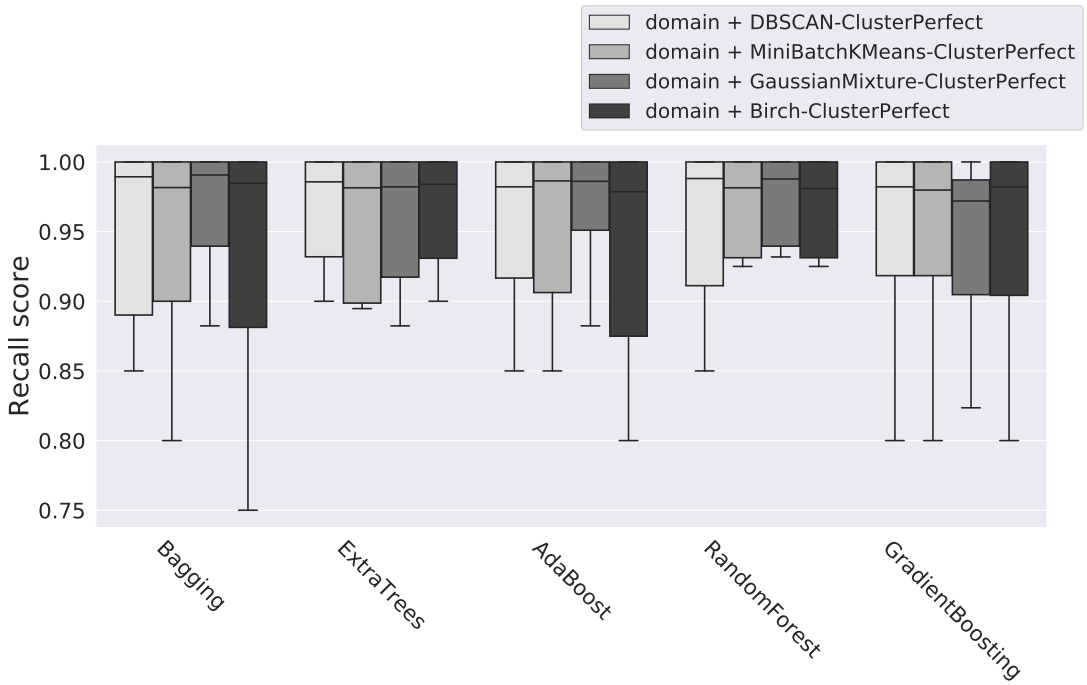


(d) Precision, higher results are better. Range is from 0.75 to 1 zoom of Figure D-3c

Figure D-3: Tackle regression over P2, comparison of ClusterPerfect

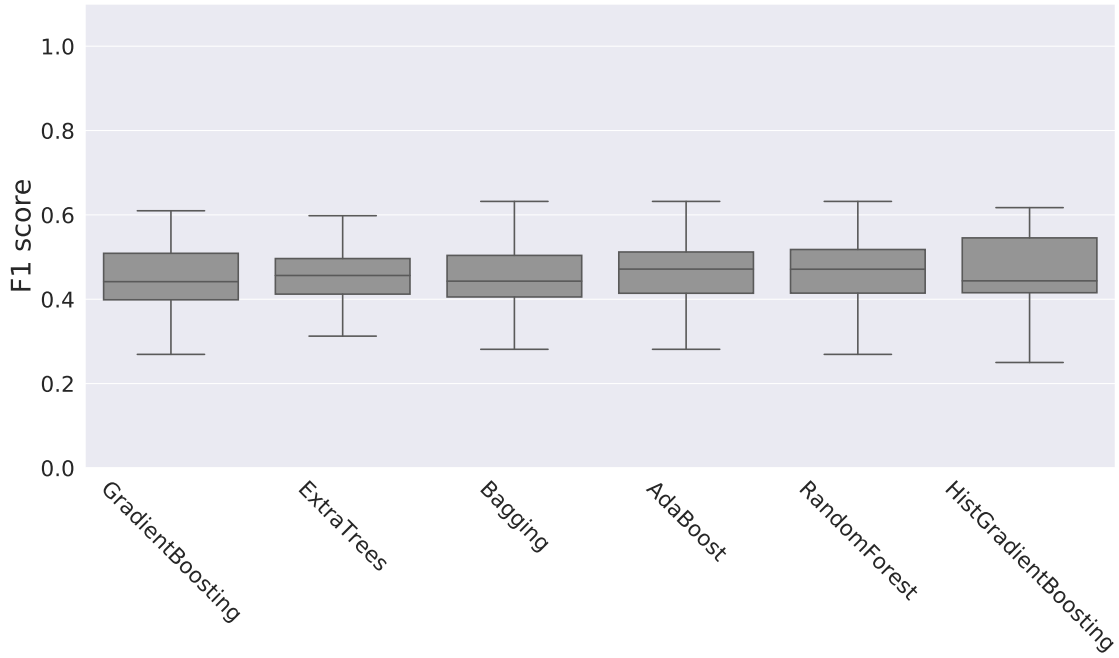


(e) Recall, higher results are better.

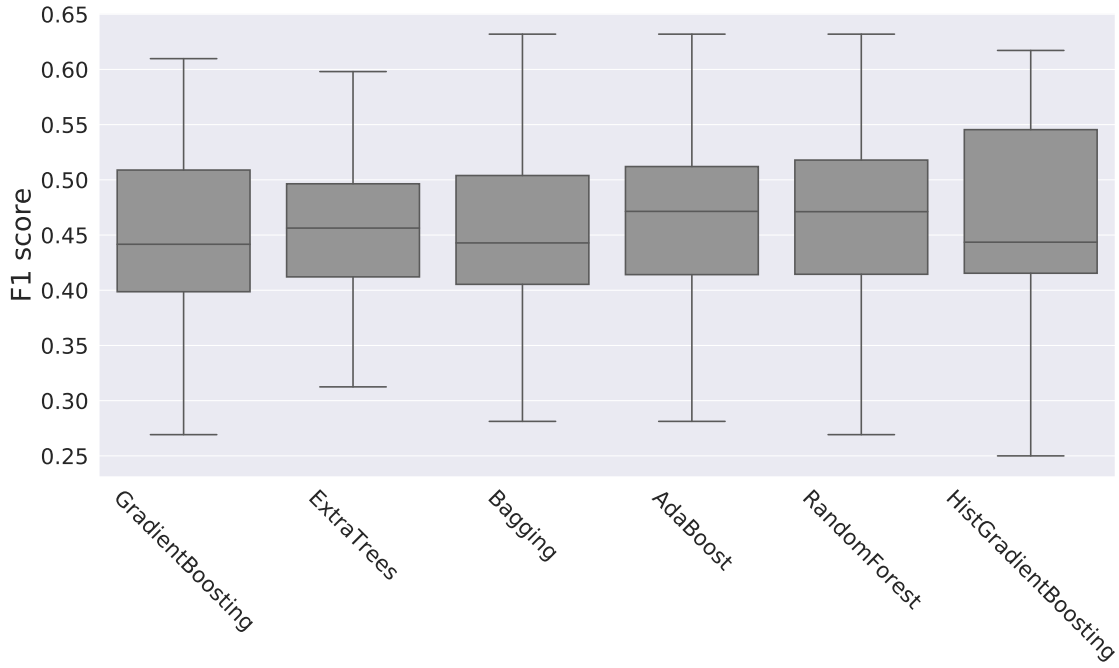


(f) Recall, higher results are better. Range is from 0.75 to 1 zoom of Figure D-3e

Figure D-3: Tackle regression over P2, comparison of ClusterPerfect



(a) F1 score range is from 0 to 1



(b) F1 score range is from 0.25 to 0.65 zoom into Figure D-4a

Figure D-4: Tackle distribution automatic classification step, higher results are better.

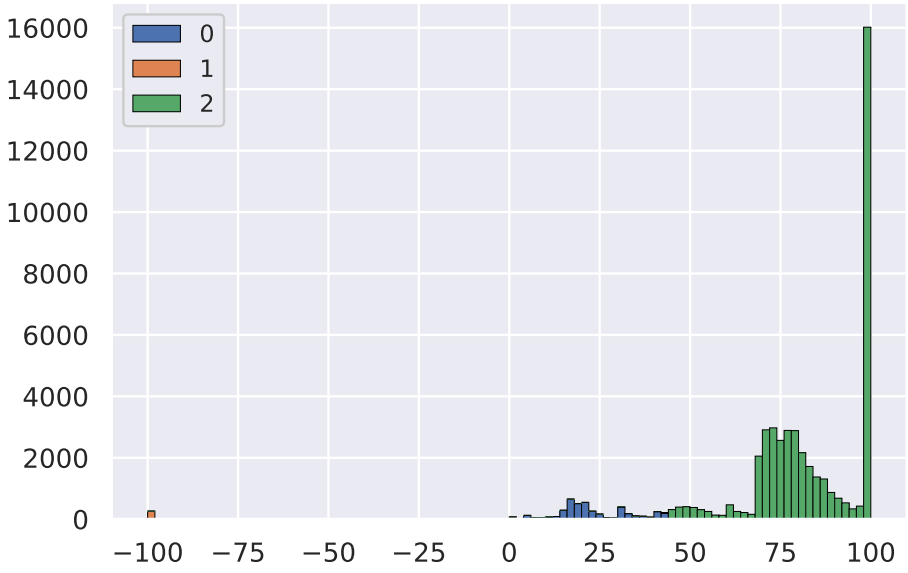
D.2 Dash Action

We tested the parameter cluster technique over the same data in an attempt to improve the regression of dash P1.

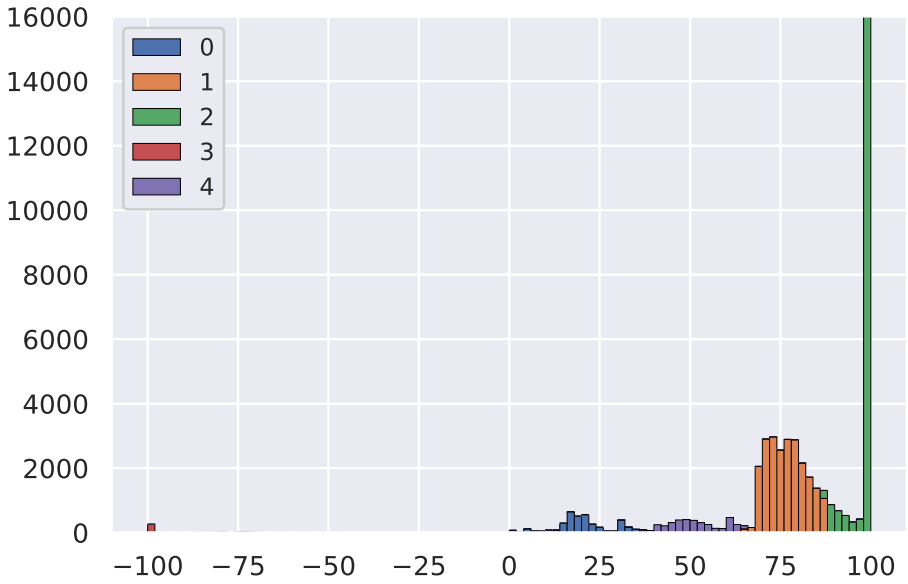
We plotted P1 with manual clustering to find a good distribution cluster, and generated the parameter's histograms using some of the standard off-the-shelf cluster algorithms such as: *MiniBatchKMeans, AffinityPropagation, SpectralClusteringtral, Ward, AgglomerativeClustering, DBSCAN, Hdbscan, Optics, Birch, and GaussianMixture*. The best results are shown in Figure D-5 on page 133. We chose the *GaussianMixture* cluster algorithm shown in Figure D-5d.

To automate the clustering process, we looked for an off-the-shelf classification algorithm, as describe in Section 5.3, to automate classification of the manual cluster. The classification results shown in Figure D-6 on page 135 indicate that the *HistGradientBoostClassifier* has the best results for all metrics and target clusters.

The final results presented in Figure 6-4 on page 66 show that the *parameter clustering* technique improves the results around all measurements, with a severe gap in relation to the other actions. Also, as expected, the manual cluster($ClusterPerfect$) was more significant than the automatic cluster($ClusterAUTO$).

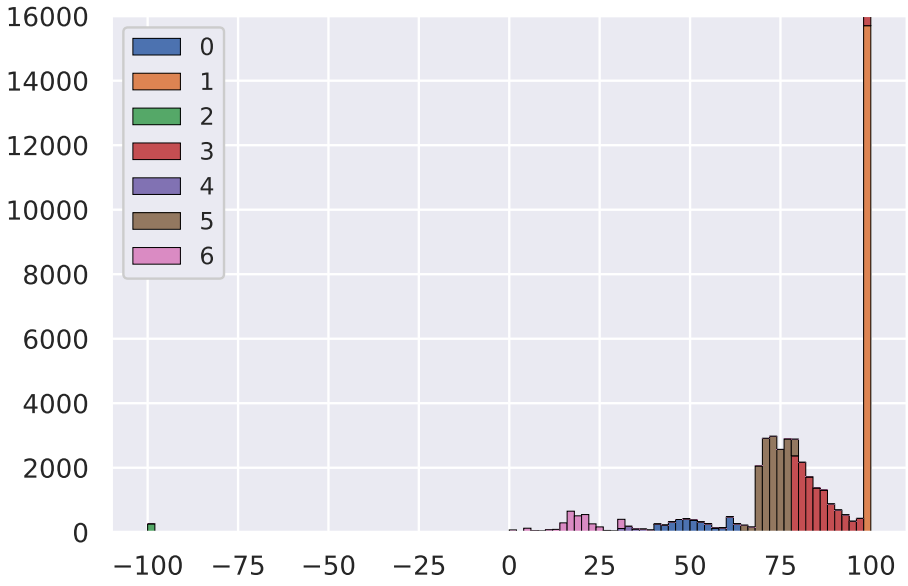


(a) Birch

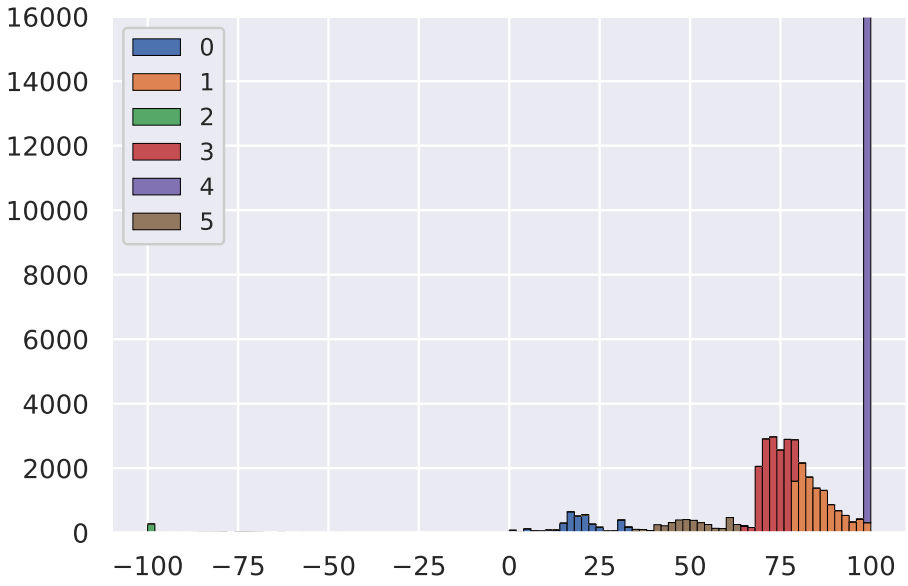


(b) MiniBatchKMeans

Figure D-5: Dash cluster algorithm plots



(c) GaussianMixture



(d) Best-GaussianMixture

Figure D-5: Dash cluster algorithm plots

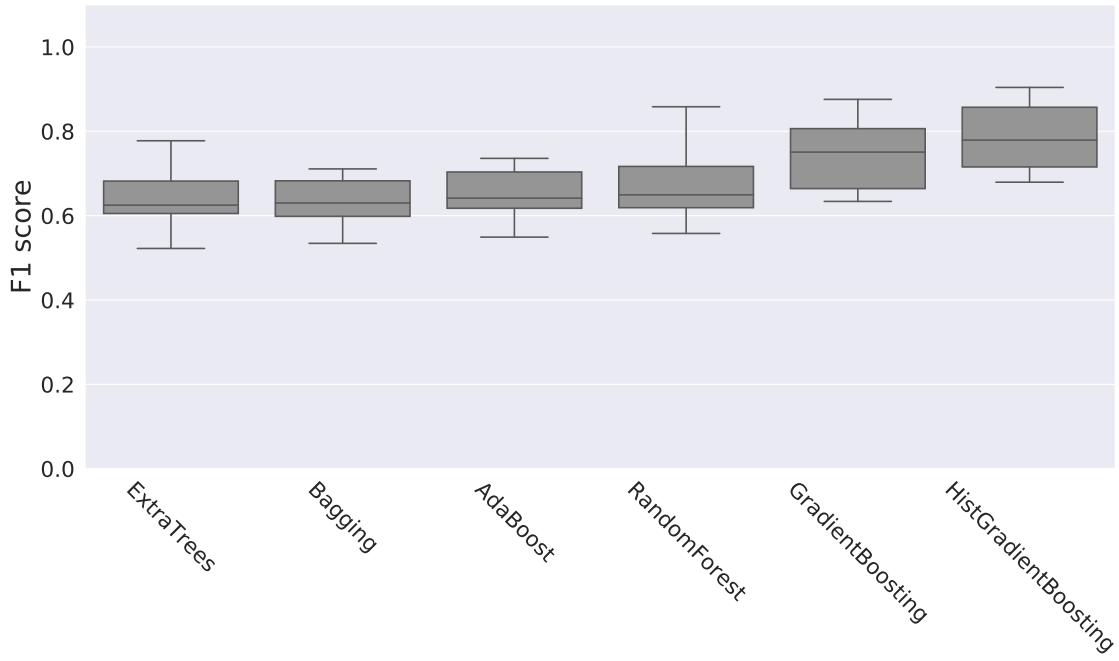


Figure D-6: Dash distribution automate classification step, higher results are better.

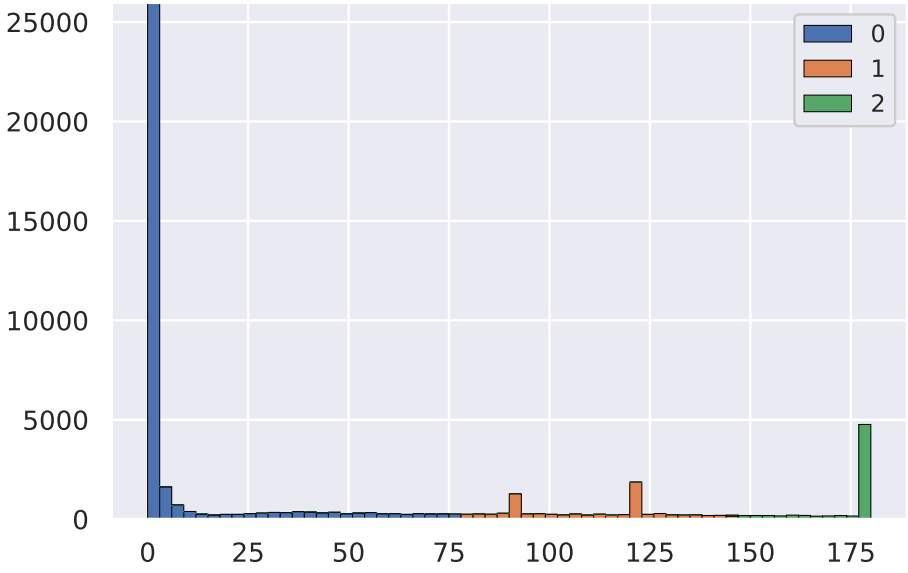
D.3 Turn Action

We tested the parameter cluster technique over the same data presented in Section 6, in an attempt to improve the regression of turn P1. We chose to generalize the P1 parameter by ignoring the right/left direction based on the *mirror transform* over the agent's goal state (G_t^i).

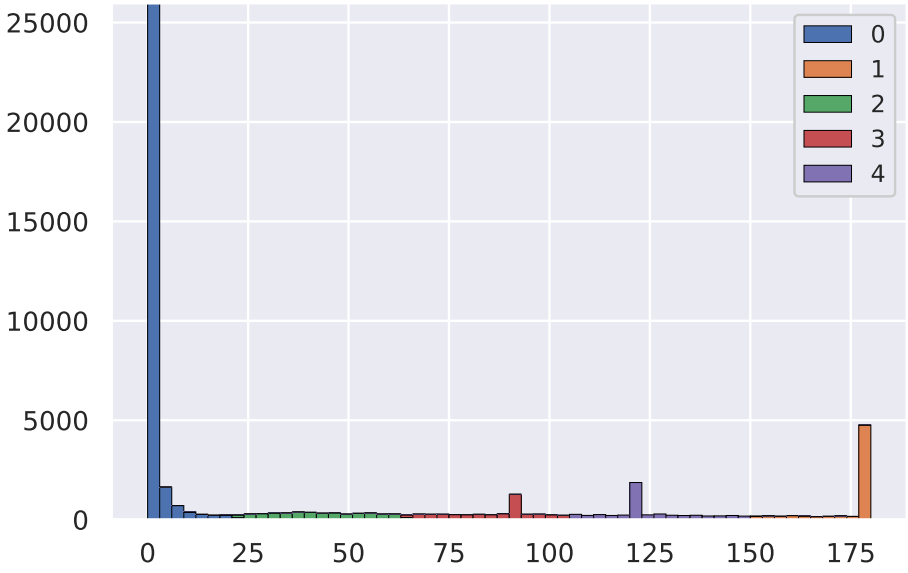
We plotted P1 with manual clustering to find a good distribution cluster that provides information over the agent's goal state. We generated the parameter's histograms over some standard off-the-shelf cluster algorithms such as: *MiniBatchK-Means, AffinityPropagation, Ward, AgglomerativeClustering, DBSCAN, Hdbscan, Optics, Birch, and GaussianMixture*. Those with the best results are shown in Figure D-7 on page 137. We chose the *GaussianMixture* cluster algorithm presented in Figure D-7d.

To automate the clustering of the agent's goal state (G_t^i) process, we looked for an off-the-shelf classification algorithm to automate classification of the manual G_t^i cluster. The classification results are shown in Figure D-8 on page 139. As can be seen, the *HistGradientBoostClassifier* earns the best results for all metrics and target clusters.

The final results shown in Figure Figure 6-5 on page 69 demonstrate that the *Parameter Clustering* technique improves the results for all measurements. As expected, the manual cluster($ClusterPerfect$) showed a more significant improvement than the automatic cluster($ClusterAUTO$).

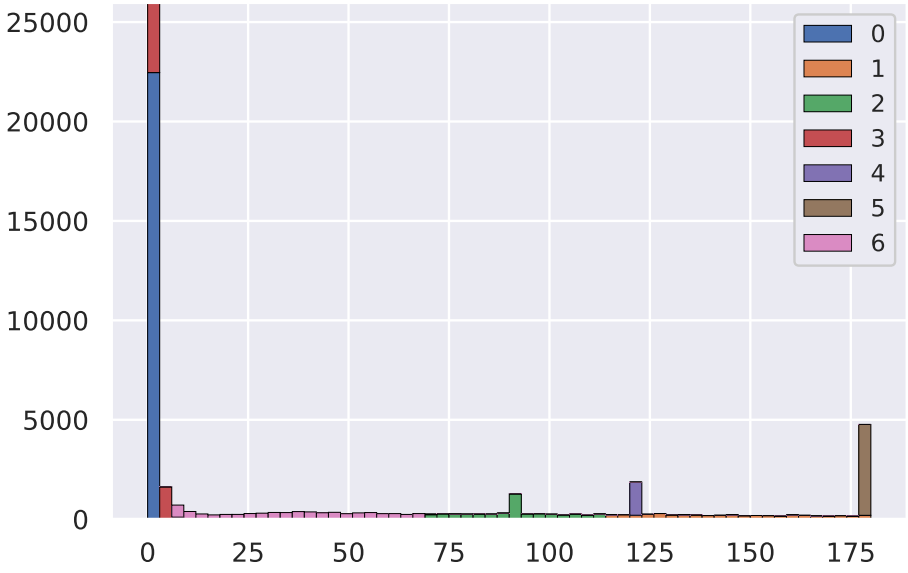


(a) Birch

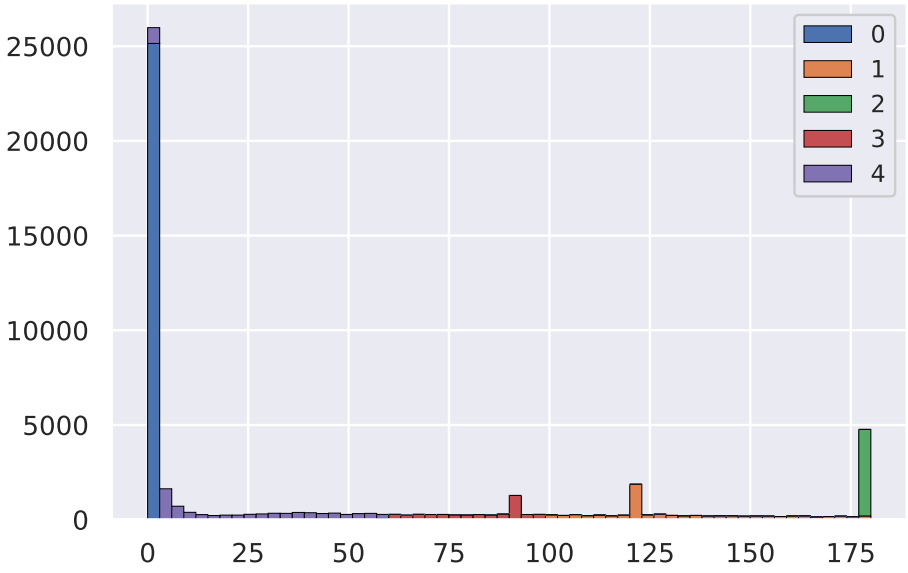


(b) MiniBatchKMeans

Figure D-7: Turn cluster algorithm plotting tries

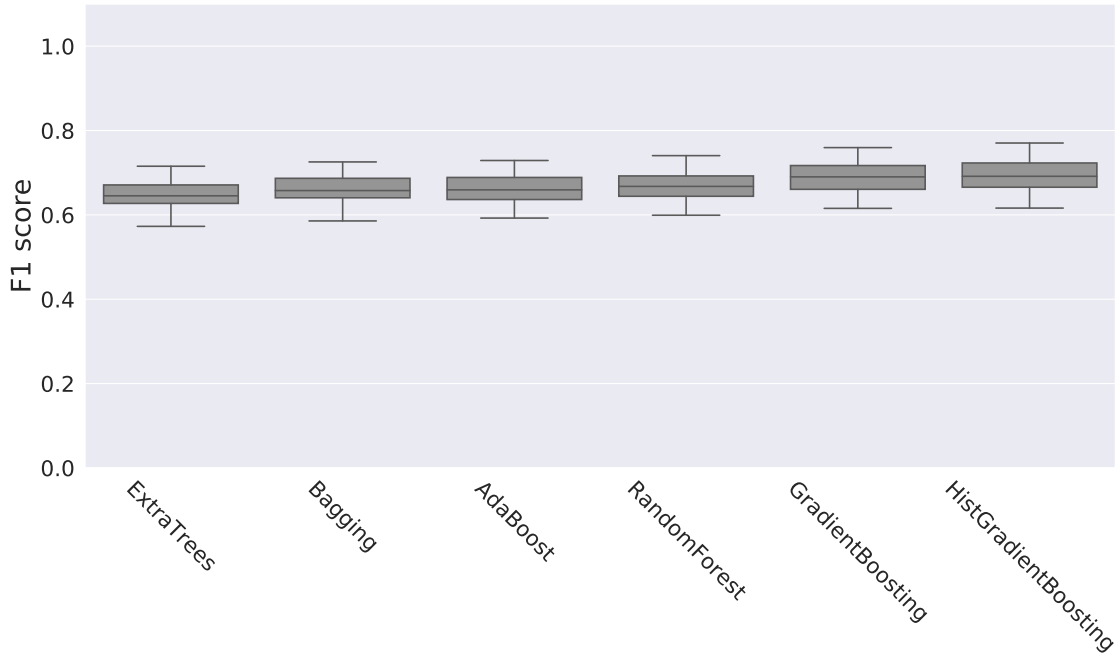


(c) GaussianMixture

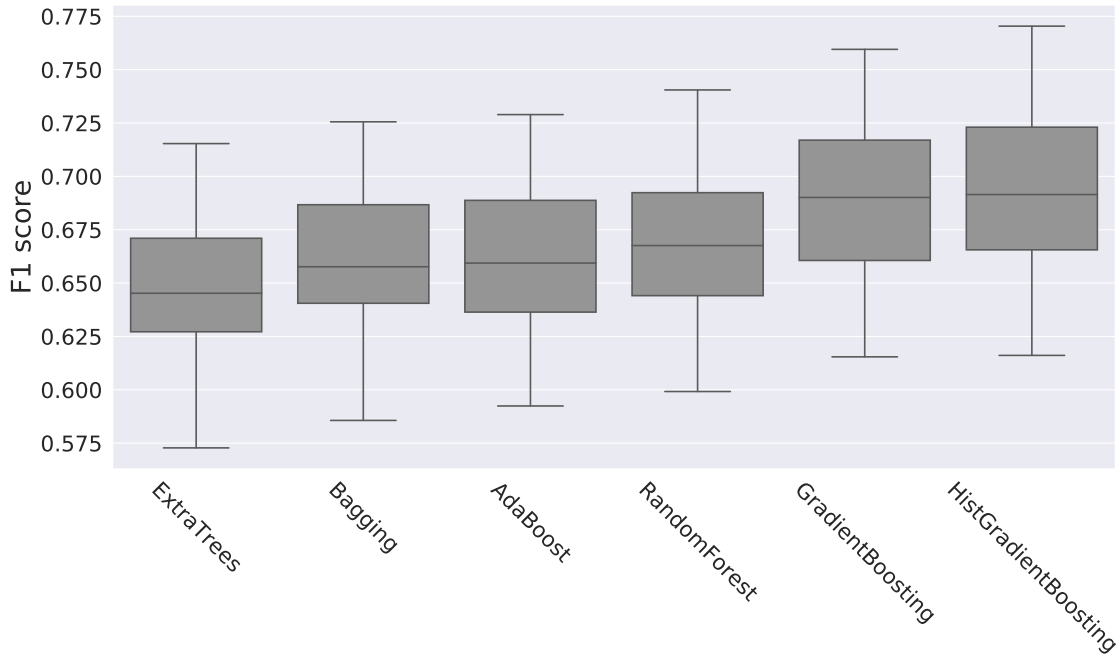


(d) Best-GaussianMixture

Figure D-7: Turn cluster algorithm plots



(a) F1 score range is from 0 to 1



(b) F1 score range is from 0.5 to 0.8 zoom into Figure D-8a

Figure D-8: Turn distribution automate classification step, higher results are better.

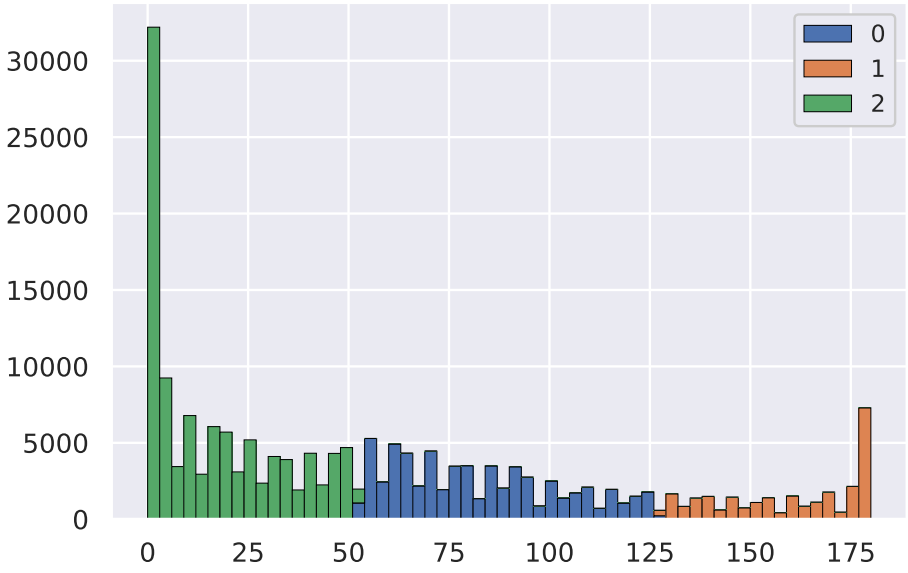
D.4 Turn Neck Action

We tested the parameter cluster technique over the same data presented in Section 6 to improve the regression of turn-neck P1.

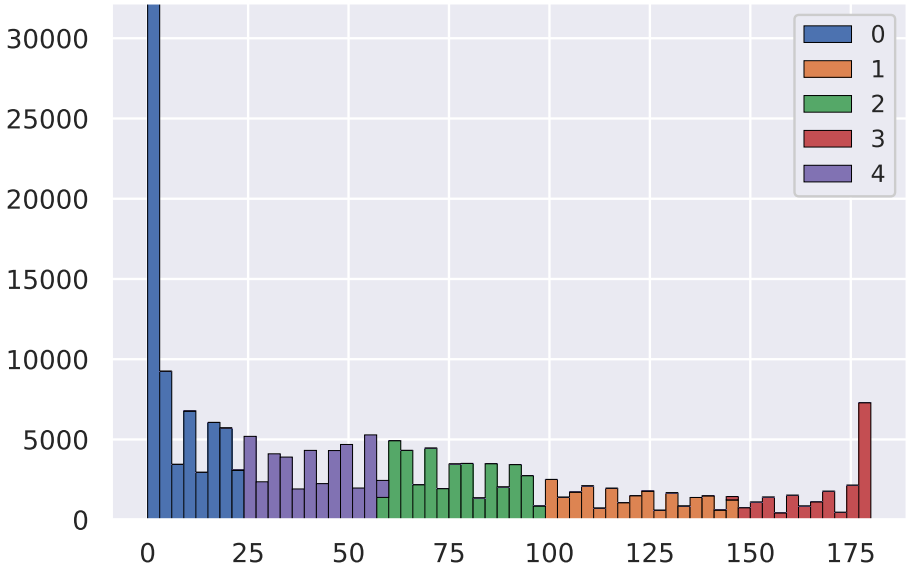
Here too, we chose to generalize the P1 parameter by ignoring the right/left direction based on the *mirror transform* over the agent's goal state. We plotted P1 with manual clustering to find a good distribution cluster. We generated the parameter's histograms over some standard off-the-shelf cluster algorithms such as: *MiniBatchKMeans, AffinityPropagation, Ward, AgglomerativeClustering, DBSCAN, Hdbscan, Optics, Birch, or GaussianMixture*. We present those with the best results in Figure D-9 on page 141. We chose the *GaussianMixture* cluster algorithm noted in Figure D-9d.

To automate the clustering of the agent's goal state process, we looked for an off-the-shelf classification algorithm. The classification results shown in Figure D-10 on page 143 indicate that the *HistGradientBoostClassifier* earns the best results for all metrics and target G_t^i clusters.

The final results in Figure 6-6 on page 72 show that the *parameter clustering* technique improves the results around all measurements. As expected, the manual cluster($ClusterPerfect$) showed a more significant improvement than the automatic cluster($ClusterAUTO$).

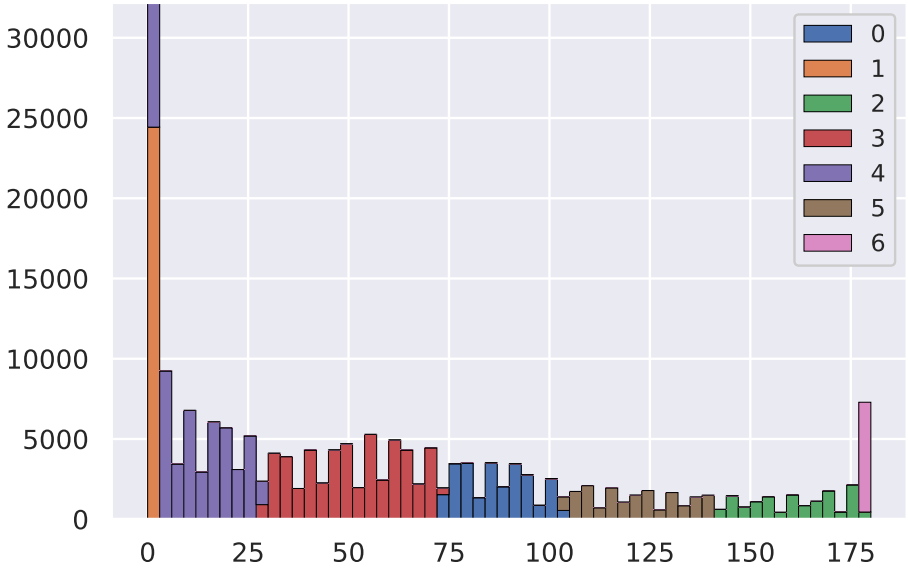


(a) Birch

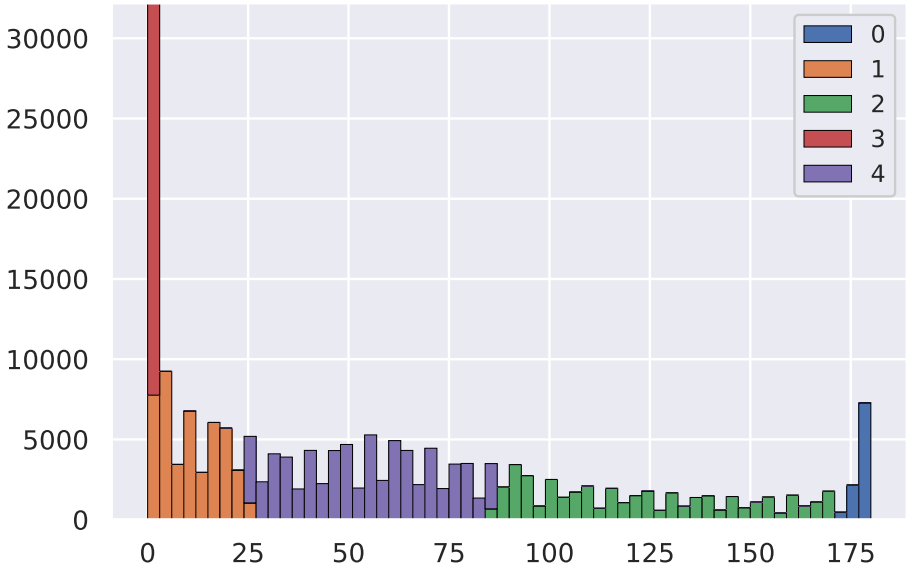


(b) MiniBatchKMeans

Figure D-9: Turn Neck cluster algorithm plotting tries

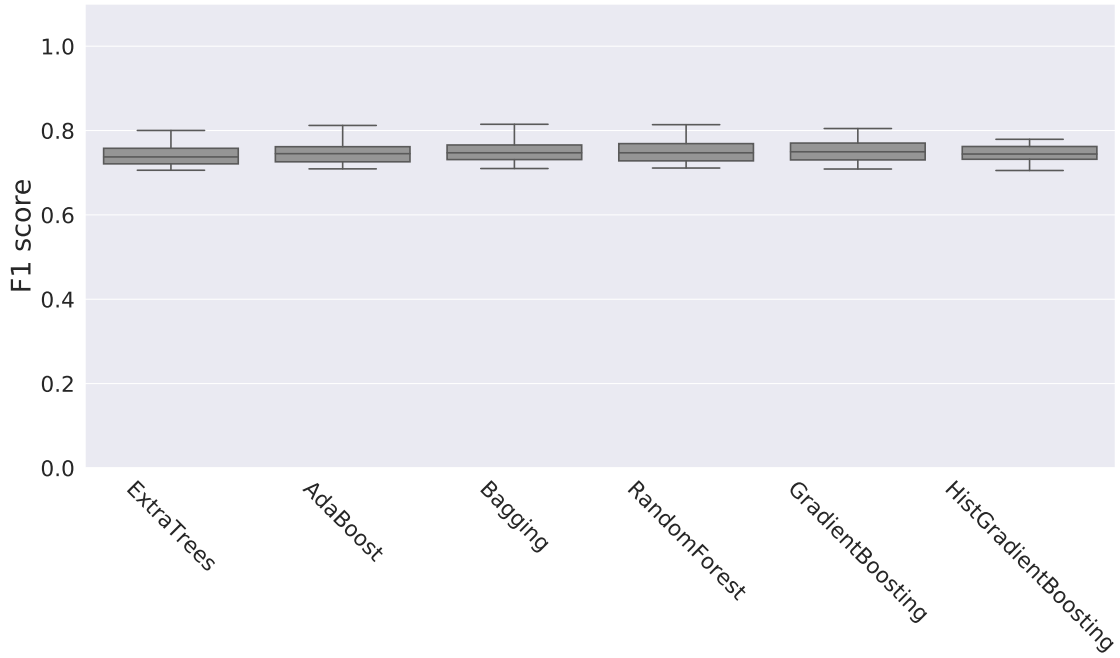


(c) GaussianMixture

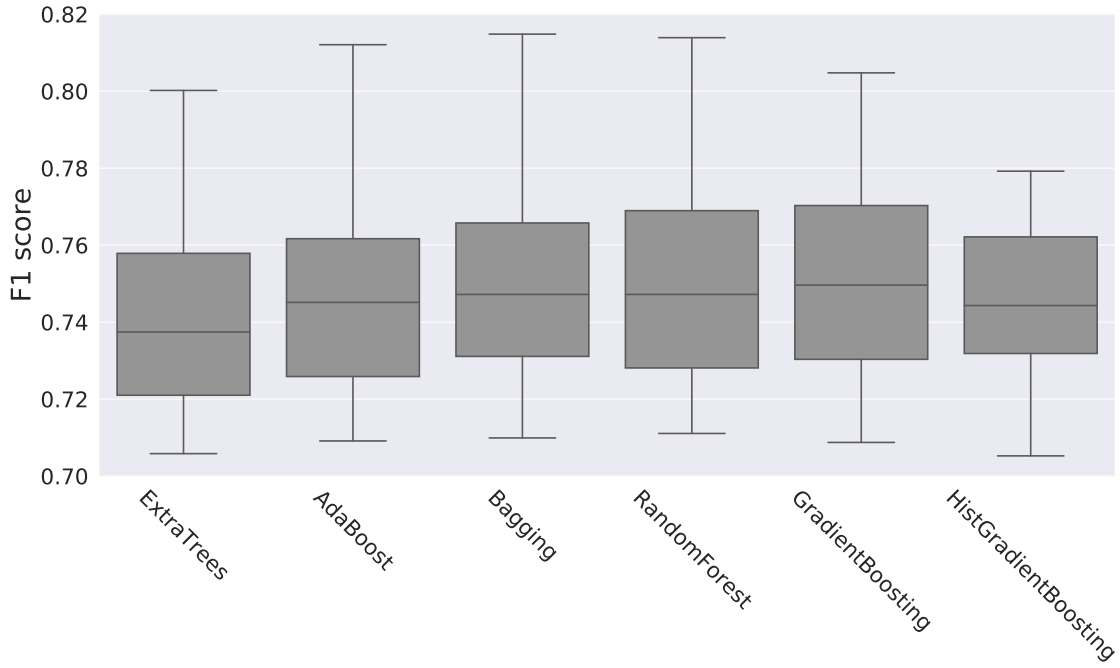


(d) Best-GaussianMixture

Figure D-9: Turn neck cluster algorithm plots



(a) F1 score range is from 0 to 1



(b) F1 score range is from 0.7 to 0.82 zoom into Figure D-10a

Figure D-10: Turn Neck distribution automate classification, higher results are better.