The Role of Model Selection in Preference Learning

Michael Huelsman and **Mirosław Truszczyński**
University of Kentucky
michael.huelsman@uky.edu, mirek@cs.uky.edu

Abstract

Learning preferences of an agent requires choosing which preference representation to use. This formalism should be expressive enough to capture a significant part of the agent's preferences. Selecting the right formalism is generally not easy, as we have limited access to the way the agent makes her choices. It is then important to understand how "universal" particular preference representation formalisms are, that is, whether they can perform well in learning preferences of agents with a broad spectrum of preference orders. In this paper, we consider several preference representation formalisms from this perspective: lexicographic preference models, preference formulas, sets of (ranked) preference formulas, and neural networks. We find that the latter two show a good potential as general preference representation formalisms. We show that this holds true when learning preferences of a single agent but also when learning models to represent consensus preferences of a group of agents.

Introduction

When predicting an agent's behavior it is important to understand her preferences. To this end, we aim to build preference representation models for agent's preferences, often exploiting learning. While a simple list of the alternatives from most to least preferred seems like a good way to represent preference orders, it is impractical when the number of alternatives is large. This work deals with cases where alternatives come from a *combinatorial domain*, which represents objects in terms of values of their attributes. Since such domains are exponential in size, wrt the number of attributes, a compact preference representation is needed. Hence, compact preference representation languages have received much attention in economics/operational research (Fishburn 1974; Dombi, Imreh, and Vincze 2007; Kohli and Jedidi 2007) and in AI (Boutilier et al. 2004; Brewka, Niemelä, and Truszczynski 2003; Kaci 2011). These languages have their own advantages and disadvantages, with no particular model being best in all situations.

Preference learning has been studied for many preference representation languages, including lexicographic preference models (Bräuning et al. 2017; Yaman et al. 2010; 2008; Dombi, Imreh, and Vincze 2007; Schmitt and Martignon 2006), and conditional preference networks (Lang and Mengin 2009; Alanazi, Mouhoub, and Zilles 2016). That work ignores the question of selecting a target formalism for learning. Our results show that learning works best when the preference model we learn is the same as or similar to that of the agent. Identifying a preference model that matches the agent's preferences may be difficult, especially when we have limited data on how the agent makes her choices. Thus, it is desirable to select a preference model to learn that shows a good degree of "universality," that is, offers acceptable performance for a wide variety of possible preference models an agent may be using.

We study four preference models for their universality: lexicographic preference models (LPMs), preference formulas (PFs), preference theories (PTs), and artificial neural networks (ANNs). The latter are not explicitly studied as preference representations but have proved their mettle as a universal model for learning. We consider both single and multiagent cases. In the single agent case we attempt to learn a model with good accuracy in predicting an agent's choices. In the multiagent case we seek to learn a "fair" joint preference model as a consensus preference model for the group.

In this work, we propose a simulated annealing algorithm for learning (ranked) PFs. We evaluate experimentally this algorithm, as well as algorithms from the literature for learning LPMs (Schmitt and Martignon 2006) and neural networks (Rumelhart, Hinton, and Williams 1986), and analyze the ability of the models these algorithms learn to represent a wide range of preference orders.

The results show a good performance of our algorithm for learning PFs and RPTs when compared with LPM and ANN models. Further, we find that of the four types of models, RPTs and ANNs provide structures flexible enough to support learning accurate preference models for data sets from a broad range of preference representations.

The remainder of this paper consists of five sections. The first of them provides background information. The next one addresses the complexity of problems involved. The third section defines the algorithms and experimental setup, the fourth one contains the results and discussion, and the last section offers concluding comments.

Background

We assume that preferences on alternatives from a set U are a *preorder* on U, a binary relation on U that is reflexive and transitive. We denote preorders by symbols such as $\succeq$ or $\geq$, possibly with annotations. A preorder $\succeq$ is an *order* (on U) if it is antisymmetric. A preorder $\succeq$ is *total* if for every pair $a, b \in U$ either $a \succeq b$ or $b \succeq a$. If $\succeq$ is a preorder (order) on U, we define its *strict* counterpart, $\succ$, by setting $a \succ b$ if $a \succeq b$ and $b \not\succeq a$, its associated *incomparability* relation $\bowtie$ by setting $a \bowtie b$ if $a \not\succeq b$ and $b \not\succeq a$, and its associated *indifference* relation $\approx$ by setting $a \approx b$ if $a \succeq b$ and $b \succeq a$.

In this work, the domain U of *alternatives*, is a *combinatorial domain*. A combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ is defined by a set of *attributes*, $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and the set of the domains of those attributes $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ (the finite sets of values the attributes can take). We write $D(v_i)$ to denote the domain of an attribute v_i. Alternatives from a combinatorial domain are then n-tuples of values — for each attribute one value from the domain of that attribute.

Combinatorial domains are exponential in size, wrt the number of attributes. Thus, representing preorders over a combinatorial domain by listing alternatives from most to least preferred is not practical. Instead, preference orders are represented implicitly by expressions (preference models) from some preference representation language. When constructing these preference models by learning, we assume only partial information about the preference order given as a set of examples of correctly ordered pairs of alternatives. Formally, an example is a triple (α, β, R) where α and β are alternatives and R is the relation between them, any of the possible "comparing" relations introduced above. This work's goal is to understand how learning preference representations depends on the model used by the agent to capture her preferences and on the model that is learned to approximate them.

Problem 1 (Preference Learning)**.** *Given a set of examples ε, consistent with the preferences of an agent A over alternatives from $\mathcal{C}(\mathcal{V}, \mathcal{D})$, find a preference model $\succ_A$ in a chosen preference representation language that satisfies (decides in the same way) the maximum number of examples in ε.*

Extending this problem to multiple agents is called *rank aggregation*. It is a natural extension of the preference learning problem. There are, however, two key differences. First, in the case of a single agent, inconsistencies in the set of examples can only occur (assuming we ignore data collection errors) when the agent does not have a coherent picture of her own preferences. In the multi-agent case, example sets consisting of preferences elicited from different agents may naturally contain inconsistencies, as agents may have opposing preferences. Second, one has to select a *fairness* criterion to optimize. In this work, we have chosen to focus on two fairness criteria: utilitarian and maximin.

Under the utilitarian criterion, we aim to satisfy as many examples as possible. This means that utilitarian rank aggregation is the same as learning from a single agent, but one which provides inconsistent examples.

Problem 2 (Utilitarian Rank Aggregation)**.** *Given a set of agents $\boldsymbol{A} = \{a_1, \ldots, a_k\}$, with each agent a_i having a preference orders $\succ_i$ over a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$, and a family of sets of examples $\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_l\}$, with each $\varepsilon_i \in \boldsymbol{\varepsilon}$ representing the preference order $\succ_i$ of agent a_i, find a preference model $\succ_{\boldsymbol{A}}$ in a chosen preference representation language that satisfies the maximum number of examples in $\cup_{i=1}^{k} \epsilon_i$.*

The maximin fairness criterion maximizes the satisfaction of the least satisfied agent.

Problem 3 (Maximin Rank Aggregation)**.** *Given a set of agents $\boldsymbol{A} = \{a_1, \ldots, a_k\}$ with preference orders over a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$, and a family of sets of examples $\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_l\}$, with each $\varepsilon_i \in \boldsymbol{\varepsilon}$ representing a preference order $\succ_i$ of agent a_i, find a preference model $\succ_{\boldsymbol{A}}$ in a chosen preference representation language that maximizes $\min_{\varepsilon \in \boldsymbol{\varepsilon}}$ satisfaction$(\succ_{\boldsymbol{A}}, \varepsilon)$.*

The important difference between maximin and utilitarian criterions is that maximin cares about the performance of individual agents, and expresses a need for every agent to be as satisfied as possible.

We conclude this section by recalling the preference representation formalisms we study. A *lexicographic preference model* (LPM) $\pi = (r, \succ)$ over the domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ consists of a one-to-one function $r : \mathcal{V} \rightarrow [1..n]$ (a *ranking*) and a collection of total orders $\succ = \{\succ_{v_1}, \succ_{v_2}, .., \succ_{v_n}\}$, with each $\succ_{v_i}$ a total order on the domain D_i of the attribute v_i (Fishburn 1974). Given alternatives $\alpha, \beta \in \mathcal{C}(\mathcal{V}, \mathcal{D})$, α is preferred to, or *dominates*, β, $\alpha \succ_\pi \beta$, if for some attribute a, $\alpha[a] \succ_a \beta[a]$, and for all attributes b such that $r(b) < r(a)$, $\alpha[b] = \beta[b]$. That is, one alternative is preferred to another if it has a more preferable value for the most important attribute where the two alternatives differ. Preference orders defined by LPMs are total orders.

A *preference formula* (PF) is an ordered finite tuple of boolean formulas $\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \ldots, \varphi_k)$ built over an alphabet of propositional variables (or atoms) $x_{v,d}$, where $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$ (Brewka, Niemelä, and Truszczynski 2003). Alternatives are represented by special truth assignments to these atoms: if an alternative α has value d on attribute v, the corresponding truth assignment assigns *true* to the atom $x_{v,d}$ and *false* to all atoms $x_{v,d'}$, where $d' \in \mathcal{D}(v) \setminus \{d\}$. We denote this truth assignment by I_α. Propositional formulas over this alphabet represent properties of truth assignments and so, in particular, of alternatives from $\mathcal{C}(\mathcal{V}, \mathcal{D})$. An alternative α has a property φ if I_α satisfies φ according to the standard definition of satisfiability.

The *satisfaction degree* of an alternative α wrt to a PF $\boldsymbol{\varphi}$, denoted $s_{\boldsymbol{\varphi}}(\alpha)$, is the smallest i such that I_α satisfies φ_i, or $k + 1$, if no such i exists. This induces a preference order on alternatives: an alternative α is at least as preferred as an alternative β wrt $\boldsymbol{\varphi}$, denoted $\alpha \succeq_{\boldsymbol{\varphi}} \beta$, if $s_{\boldsymbol{\varphi}}(\alpha) \leq s_{\boldsymbol{\varphi}}(\beta)$. The relation $\succeq_{\boldsymbol{\varphi}}$ is a total preorder.

A *preference theory* (PT) is a collection of preference formulas. Given a PT $P = \{\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_n\}$, we use Pareto dominance to define a preorder on $\mathcal{C}(\mathcal{V}, \mathcal{D})$. Namely, we define $\alpha \succeq_P \beta$ if for every $\boldsymbol{\varphi} \in P$, $s_{\boldsymbol{\varphi}}(\alpha) \leq s_{\boldsymbol{\varphi}}(\beta)$.

A PT can be extended by assigning ranks to preference formulas. Formally, a *ranked* PT (or an RPT) is a collection P of preference formulas and a mapping that assigns a rank $r(\boldsymbol{\varphi})$ to each preference formula $\boldsymbol{\varphi} \in P$. Dominance between alternatives α and β in an RPT is determined by first comparing the alternatives on the lowest ranked PFs. If these formulas determine that α dominates β, then α dominates β in the order defined by P. Otherwise, we consider the next lowest rank group of PFs and proceed in the same way.

We also represent preference orders by ANNs. While ANNs may have many different structures (Rosenblatt 1958; Hopfield 1982; Elman 1990), we chose to use simple feed-forward networks which take as input two alternatives, in a canonical order, and use a softmax output layer to determine a class label for the pair of alternatives describing how they compare. The labels used are $\succ, \succeq, \approx, \prec, \preceq, \bowtie$.

Finally, to model agents in experiments, in addition to LPMs and PFs, we also use CP-nets, a well known formalism in research about preferences (Boutilier et al. 2004). We omit formal details on CP-nets due to space limits.

Problem Complexity

In this section we consider computational complexity of learning PFs and (R)PTs, as well as the power of PFs and (R)PTs in expressing preorders and total preorders. We focus the discussion on combinatorial domains $\mathcal{C}(\mathcal{A})$ over a set $\mathcal{A} = \{X_1, \dots, X_n\}$ of n binary attributes, where the domain of every attribute $X_i \in \mathcal{A}$ is binary and has values x_i and $\bar{x}_i$. Resttiction to binary domains is common and not limiting, as combinatorial domains with non-binary domains can be effectively reduced to those with binary attributes only. We outlined this reduction earlier, when discussing PFs and PTs as preference models on arbitrary combinatorial domains.

Of key interest to us is the problem of learning preference models.

Problem 4 (Preference Model Learning PML$(\mathcal{L})$)**.** *Let $\mathcal{L}$ be a preference language for a binary combinatorial domain $\mathcal{C}(\mathcal{A})$. Given an integer k and a set of examples ε representing a preference order over alternatives in $\mathcal{C}(\mathcal{A})$, decide whether there is a preference model M in $\mathcal{L}$ that satisfies at least k examples from ε.*

The complexity of PML$(\mathcal{L})$ depends on $\mathcal{L}$. The case when $\mathcal{L}$ = LPM was studied by Schmitt and Martignon (2006). They proved that the problem was NP-complete. They also proposed a greedy algorithm for learning LPMs. Their algorithm guarantees that the learned model satisfies at least half of the examples and, in the case when the example set is consistent with some LPM, *all* of them. Learning ANNs was proved to be NP-complete by Blum and Rivest (1989).

To the best of our knowledge the complexity of learning of PFs, PTs and RPTs has not been studied. We show PML is NP-complete for each of these formalisms. This holds even when we restrict ourselves to only DNF formulas.

Theorem 1. *The problem* PML$(\mathcal{L})$ *is NP-complete for $\mathcal{L}$ = PF, PT and RPT, even when formulas appearing in preference formulas are in DNF.*

We consider the rank aggregation problem under utilitarian and maximin goal functions. Formally, we state the rank aggregation model learning problem RAML$(\mathcal{L})$ as follows.

Problem 5 (RAML$(\mathcal{L})$)**.** *Let $\mathcal{L}$ be a preference language for a binary combinatorial domain $\mathcal{C}(\mathcal{A})$. Given an integer k and a collection of sets of examples $(\varepsilon_1, \dots, \varepsilon_n)$, with each ε_i representing a preference order over $\mathcal{C}(\mathcal{A})$ used by an agent a_i, $i = 1, \dots, n$, decide whether there is a preference model M in $\mathcal{L}$ that satisfies at least k examples in $\varepsilon = \bigcup_{i=1}^{n} \varepsilon_i$ (for the* utilitarian *criterion), or at least k examples in each set ε_i (for the* maximin *criterion).*

One can show that both versions of the RAML$(\mathcal{L})$ problem are in NP. Moreover, if $n = 1$ then the RAML$(\mathcal{L})$ problem (in for each fairness criterion) reduces to PML$(\mathcal{L})$. Thus, we have the following corollary to Theorem 1.

Corollary 1. *The problem* RAML*$(\mathcal{L})$ (in each of its versions) is NP-complete for $\mathcal{L}$ = PF, PT and RPT, even when formulas appearing in preference formulas are in DNF.*

The last problem we discuss briefly is the expressivity of PFs, PTs and RPTs in modeling preference orders over combinatorial domains. All preorders can be captured by PTs and all total preorders by PFs.

Theorem 2. *Let $\mathcal{C}(\mathcal{A})$ be a combinatorial domain over binary attributes from $\mathcal{A}$. Let $\succeq$ be any preorder (resp., total preorder) over $\mathcal{C}(\mathcal{A})$. Then, there is a preference theory (resp., preference formula) Φ such that $\succeq$ and the preorder (resp., total preorder) $\succeq_\Phi$ defined by Φ coincide.*

This suggests that PFs and (ranked) PTs are good candidates for general preference learning. This result has a limitation. Namely, PFs and PTs may have large sizes, often exponential in the number of attributes. However, when a preorder is specified by a set of examples, the formula (theory) Φ is of polynomial size in the size of the example set.

Algorithms and Setup

We study four formalisms for their ability to represent, with good accuracy, a wide range of possible preference orders. These formalisms are: LPMs, PFs, RPTs, and ANNs. To learn LPMs we use the greedy algorithm which was studied by Schmitt and Martignon (2006). This algorithm chooses the ranking of attributes in an LPM by iteratively selecting the attribute which produces the least number of unsatisfied examples. We modified it for use in maximin rank aggregation case by having it choose the next attribute based on the maximin score.

For learning PFs and RPTs we developed algorithms using heuristic search techniques. We considered both *genetic algorithms* and *simulated annealing*. We observed poorer performance from the genetic algorithms and, consequently, focused on simulated annealing. We chose to learn PFs and RPTs which are built using *disjunctive normal form* (DNF) boolean formulas. The use of a normal form to represent boolean formulas is standard. The choice of DNF is motivated by common linguistic patterns used to describe preferences. For instance, preferred vacations are on the beach in Italy (conjunction: on the beach and in Italy) or in the mountains in Colorado (another conjunction).

Simulated annealing requires all objects (here, PFs and (R)PTs) have the same size, which restricts their form. Each DNF is specified by: i — the number of disjuncts in a formula and j — the number of literals in each disjunct. We also specify k, the number of formulas in a PF. For RPTs we add b, the number of PFs per rank, and a, the number of ranks. We specify the size (type) of PF using a triple

(i, j, k) and the size of an RPT using a 5-tuple (a, b, i, j, k). A PF $(x_1 \wedge \neg x_3, x_2 \wedge x_3)$ has type $(1, 2, 2)$, and a PF $(\neg x_1 \vee x_4, x_2 \vee x_3, x_1 \vee \neg x_3)$ has type $(2, 1, 3)$. Our algorithm does not avoid duplication of literals, products, or formulas. This guarantees that the class of PFs of type (i, j, k) simulates all PFs of type (i', j', k'), where $i' \leq i$, $j' \leq j$ and $k' \leq k$. For example the product $a \wedge a \wedge b$ is the same semantically as the product $a \wedge b$.

For algorithms, we convert PFs into strings of literals. Knowing the type allows us to recover the formula back. Simulated annealing algorithms require a neighbor relation in the search space. In our case, the search space is formed by strings of literals (representations of PFs of a specified type). For our algorithm, we define two PFs, of the same type, as neighbors if their atom string representations differ in only one place. For instance, ab and ac, ab and $a\neg c$, and ab and $\neg ab$ are three pairs of neighbors. This neighbor relation easily extends to RPT atom strings.

Given an example set ε our algorithm randomly selects a starting candidate solution and sets the initial temperature to 100. Each iteration we randomly select a neighbor. If that neighbor satisfies more examples we replace our current candidate solution with that neighbor. If not, we replace our current solution with that neighbor with probability $e^{\frac{-\Delta}{T}}$, where Δ is the difference in performance of the two neighboring PFs and T is the current temperature. Each iteration we cool the temperature by dividing it by 1.001 and iterate until we reach a temperature of 10^{-7}. After stopping we run a naive hill climbing algorithm, that is, we take our candidate solution and test the performance of all its neighbors, replacing the candidate solution with its best neighbor that outperforms it until no improving neighbor exists.

In experiments, we built simple, linear, feed forward ANNs using the Pytorch (Paszke et al. 2017) Python module varying the number of hidden layers but keeping the number of nodes constant at 256. The number of hidden layers is varied since that changes the functions that can be approximated by an ANN (Hornik 1991). Each node uses a ReLU activation function and the output layer consists of six classes $\succ$, $\succeq$, $\approx$, $\prec$, $\preceq$, $\bowtie$ using a softmax function to decide the output class. Each model is trained for 1000 epochs.

To build example sets, we generate a preference order (either an LPM, PF, or CP-net). For PF and CP-nets we limit their forms: PFs are labeled by their type and CP-nets are labeled with a value i which is the maximum number of conditioning attributes another attribute can have. These labels are varied through the experiments. Except for CP-nets, which are generated using a state of the art tool by Allen et. al. (2016), models are built using straightforward random generators.

Examples are built by selecting two different alternatives α, β from $\mathcal{C}(\mathcal{V}, \mathcal{D})$. If the alternatives have already been used they are discarded and another pair is generated. The pair is compared using the preference model and added to ε. In the multiagent case this process is repeated for each agent and allows pairs to show up in multiple example sets.

We use several default parameters in our experiments. Example sets consist of 100 examples, derived from preference orders over a domain of eight binary attributes, for each agent. The multiagent case has 5 agents, thus 5 example sets. We use 5-fold cross validation to test learned models on unseen examples. I.e. each time a learning algorithm is run a fifth of the examples are set aside for validation. Each experiment is run 25 times and results are averaged over the runs. Data are reported as the proportion of examples satisfied.

Results and Discussion

Learning the preferences of a single agent, see Table 1, yielded two broad results. First, training accuracy is maximized when the model being learned matches the model producing the examples. We see this with LPMs, but a 2,2,7 PF is just as effective at learning 1,1,3 PF preferences as 1,1,3 PFs. This is due to 2,2,7 PFs reproducing 1,1,3 PFs since 2,2,7 PFs can contain duplicate atoms.

	LPM	1,1,3 PF	2,2,3 PF	0 Layer NN
LPM	1.00	0.66	0.92	0.93
1,1,3 PF	0.64	1.00	1.00	0.89
2,2,7 PF	0.61	0.73	0.99	0.84
0 CP-net	0.19	0.16	0.21	0.94
7 CP-net	0.73	0.54	0.75	0.86

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.83	0.97	0.75
1,1,3 PF	0.78	0.72	0.85
2,2,7 PF	0.73	0.70	0.66
0 CP-net	0.92	0.96	0.80
7 CP-net	0.76	0.94	0.75

Table 1: Learning accuracy in single agent cases.

Secondly, both ANNs and PTs learn highly accurate preference models across the different representations, that is, ANNs and RPTs show a potential for “universality”. This is theoretically supported for PTs 2 and also expected for ANNs, which are powerful learning models.

Training accuracy is a metric of how well the models fit the training exmaples, but it is also important that we test the predictive power of our learned models. For this we perform cross validation. The results are shown in Table 2.

	LPM	1,1,3 PF	2,2,3 PF	0 Layer NN
LPM	0.98	0.60	0.75	0.85
1,1,3 PF	0.62	1.00	0.98	0.80
2,2,7 PF	0.55	0.68	0.94	0.59
0 CP-net	0.18	0.10	0.13	0.79
7 CP-net	0.65	0.49	0.63	0.63

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.76	0.80	0.61
1,1,3 PF	0.71	0.62	0.84
2,2,7 PF	0.52	0.54	0.58
0 CP-net	0.79	0.69	0.63
7 CP-net	0.56	0.71	0.63

Table 2: Validation accuracy in single agent cases.

As expected, accuracy drops when applying learned models to unseen examples. This decrease is less than 0.2, with many models performing better. The observed ability to generalize means that our algorithms are learning models capturing well the original preference orders, not simply satisfying examples in the example set. Note that non-ANN models performed better in terms of difference between training and validation accuracy. This might be due to the added structure which is inherent in LPMs, PFs, and PTs.

These results show the viability of learning preferences using both traditional models and ANNs. Learning LPMs and PFs to approximate CP-net defined orders, is less effective than learning ANNs and RPTs. We conjecture this is due to the presence of incomparabilities in CP-net orders that cannot be reproduced by LPMs and PFs.

The next question is how well these learning processes perform when we are trying to learn the preferences of multiple agents. We start with the case of learning under the utilitarian fairness criterion.

	LPM	1,1,3 PF	2,2,7 PF	0 Layer NN
LPM	0.67	0.45	0.63	0.76
1,1,3 PF	0.46	0.56	0.59	0.68
2,2,7 PF	0.46	0.49	0.58	0.66
0 CP-net	0.17	0.10	0.15	0.89
7 CP-net	0.51	0.37	0.51	0.70

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.79	0.69	0.49
1,1,3 PF	0.67	0.51	0.51
2,2,7 PF	0.66	0.51	0.46
0 CP-net	0.90	0.85	0.70
7 CP-net	0.69	0.60	0.48

Table 3: Utilitarian joint preference learning accuracy.

Table 3 shows that training accuracy decreases in a multiagent senario, as expected. This is most likely due to inconsistencies introduced since agents may disagree. Comparing Table 3 to Table 4 we see that in the multiagent case differences between training and validation accuracy are similar to the single agent case. When dealing with utilitarian aggregation we are attempting to satisfy as many examples as possible, with no regard to the preferences of individual agents, thus it is not surprising that we see similar performance as in the single agent case. The phenomenon of PTs and ANNs being the most general holds for utilitarian aggregation.

	LPM	1,1,3 PF	2,2,7 PF	0 Layer NN
LPM	0.63	0.42	0.51	0.56
1,1,3 PF	0.42	0.53	0.51	0.44
2,2,7 PF	0.43	0.46	0.48	0.40
0 CP-net	0.15	0.08	0.10	0.78
7 CP-net	0.47	0.34	0.42	0.43

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.57	0.58	0.43
1,1,3 PF	0.42	0.41	0.49
2,2,7 PF	0.41	0.40	0.42
0 CP-net	0.79	0.73	0.66
7 CP-net	0.43	0.44	0.39

Table 4: Utilitarian joint preference validation accuracy.

Modifying simulated annealing and the greedy LPM algorithms for maximin training is relatively easy, but it is not a trivial change for ANNs. Such a modification is beyond the scope of this work and we leave it for furture work. Instead we use the ANNs learned with the utilitarian goal function as an approximation for the maximin aggregation.

	LPM	1,1,3 PF	2,2,7 PF	0 Layer NN
LPM	0.51	0.36	0.53	0.67
1,1,3 PF	0.33	0.39	0.48	0.60
2,2,7 PF	0.35	0.39	0.49	0.59
0 CP-net	0.11	0.03	0.09	0.85
7 CP-net	0.36	0.23	0.38	0.61

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.71	0.59	0.47
1,1,3 PF	0.59	0.41	0.45
2,2,7 PF	0.57	0.42	0.43
0 CP-net	0.86	0.80	0.68
7 CP-net	0.60	0.50	0.43

Table 5: Maximin joint preference training accuracy.

Applying our methods to maximin aggregation decreases performance when compared to the utilitarian aggregation, see Table 5. Another measure of fairness is the concept of proportionality. An allocation is proportional if each agent (of n total agents) gets at least $\frac{1}{n}$ of their total possible utility. Naively applying proportionality to our problems, a method is proportional if it satisfies at least 0.20 of an agent's examples, since we have 5 agents. This means our average maximin aggregations are proportional, in fact, they significantly exceed the threshold. Another difference between utilitarian and maximin aggregations is the greater difference between training and validation, see Table 6. LPM, PF, and PT learning strategies drop by as much as 0.14, while ANNs may lose up to 0.35 in accuracy. In other words, generalizing for maximin aggregation is worse than for utilitarian aggregation, however using non-ANN models generalizes better than using an ANN.

	LPM	1,1,3 PF	2,2,7 PF	0 Layer NN
LPM	0.42	0.24	0.35	0.40
1,1,3 PF	0.24	0.26	0.30	0.28
2,2,7 PF	0.27	0.28	0.30	0.25
0 CP-net	0.06	0.01	0.02	0.66
7 CP-net	0.29	0.17	0.25	0.27

	3 Layer NN	3,3,2,2,7 RPT	1,3,1,1,3 RPT
LPM	0.41	0.51	0.43
1,1,3 PF	0.28	0.35	0.41
2,2,7 PF	0.25	0.35	0.41
0 CP-net	0.67	0.72	0.65
7 CP-net	0.28	0.39	0.36

Table 6: Maximin joint preference validation accuracy.

Some general trends emerge from the data. Larger PTs tend to have a problem with overfitting in the single agent case, as we can see by the larger disparity between training and validation accuracy for $3, 3, 2, 2, 7$ RPTs than with $1, 3, 1, 1, 3$ RPTs. Second, in the single agent setting it is best to learn a model which aligns with that agent, although RPTs and ANNs are general enough that acceptable representations can still be learned. Third, ANNs work well in training, but have a tendency to overfit more, as indicated by a larger disparity between training and validation accuracies than for the other models.

An ANN classifier has deficits beyond its inability to generalize well. Most importantly the learned relation may not be a preorder. In several cases, the relations learned by ANNs did not satisfy the fundamental property of transitivity, thus an alternative could be strictly better than itself.

Additionally, it is computationally infeasible to compute optimal alternatives. It is also impossible to extract key underlying characteristics of the original preference order such as which attributes are more important when comparing alternatives. Unlike learned LPMs, PFs, and RPTs that allow us to extract which properties are desirable, there is no practical way to get this information from an ANN.

Conclusion

We studied learning of preference models to represent preference orders represented by examples. When the examples come from a group of agents, the learned models can be be used as aggregated preference order.

We established the complexity of learning PFs and (R)PTs, and we obtained the results on the ability of PFs and (R)PTs to express arbitrary (total) preorders. We designed and studied algorithms learning PFs, (R)PTs and ANNs as representations of preference orders.

Heuristic search algorithms for learning PFs and RPTs perform well. For example, 2, 2, 7 PFs averaged an accuracy of 0.74 in single agent training. Learning simpler RPTs produced an average training accuracy of 0.77. ANNs provide a promising way of learning a variety of qualitative preferences. ANNs, along with RPTs, are shown to be flexible enough to learn several different types of preference well which means they are the most universal of the preference formalisms we study. RPTs and ANNs both average above 0.70 in the single agent case and above 0.50 in terms of utilitarian aggregation. We also show that learning a consensus preference in the egalitarian (maximin) setting is a viable method of preference aggregation, with our learned models producing proportional results on average.

There remains several open questions which are the direct result of this work. Can a maximin method of learning ANNs be constructed, and how well does it perform? Are there better hyperparameter settings for learning preferences? Do they change based on the representation being learned? Finally, can heuristic search be applied to more formalisms than just PFs and RPTs while achieving good results? These questions will require further work.

Acknowledgements

This work was partially funded by the NSF under the grant number IIS-1618783.

References

Alanazi, E.; Mouhoub, M.; and Zilles, S. 2016. The complexity of learning acyclic CP-nets. In *IJCAI*, 1361–1367.

Allen, T. E.; Goldsmith, J.; Justice, H. E.; Mattei, N.; and Raines, K. 2016. Generating CP-nets uniformly at random. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Blum, A., and Rivest, R. L. 1989. Training a 3-node neural network is NP-complete. In *Advances in Neural Information Processing Systems*, 494–501.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Bräuning, M.; Hüllermeier, E.; Keller, T.; and Glaum, M. 2017. Lexicographic preferences for predictive modeling of human decision making: A new machine learning method with an application in accounting. *European Journal of Operational Research* 258(1):295–306.

Brewka, G.; Niemelä, I.; and Truszczynski, M. 2003. Answer set optimization. In *IJCAI*, volume 3, 867–872.

Dombi, J.; Imreh, C.; and Vincze, N. 2007. Learning lexicographic orders. *European Journal of Operational Research* 183(2):748–756.

Elman, J. L. 1990. Finding structure in time. *Cognitive science* 14(2):179–211.

Fishburn, P. C. 1974. Lexicographic orders, utilities and decision rules: A survey. *Management science* 20(11):1442–1471.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* 79(8):2554–2558.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4(2):251–257.

Kaci, S. 2011. *Working with Preferences: Less is More*. Springer.

Kohli, R., and Jedidi, K. 2007. Representation and inference of lexicographic preference models and their variants. *Marketing Science* 26(3):380–399.

Lang, J., and Mengin, J. 2009. The complexity of learning separable ceteris paribus preferences. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65(6):386.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature* 323(6088):533–536.

Schmitt, M., and Martignon, L. 2006. On the complexity of learning lexicographic strategies. *Journal of Machine Learning Research* 7(Jan):55–83.

Yaman, F.; Walsh, T. J.; Littman, M. L.; and Desjardins, M. 2008. Democratic approximation of lexicographic preference models. In *Proceedings of the 25th International Conference on Machine Learning*, 1200–1207. ACM.

Yaman, F.; Walsh, T. J.; Littman, M. L.; and desJardins, M. 2010. Learning lexicographic preference models. In *Preference learning*. Springer. 251–272.