Technical University of Crete
School of Electrical and Computer Engineering

Collaboration Pattern Detection in Hedonic Cooperative Games with Externalities

Diploma Thesis

Dimitrios Troullinos

Chania, September 2019

Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ανίχνευση Μοτίβων Συνεργειών σε Ηδονικά Συνεργατικά Παίγνια με Εξωτερικές Επιδράσεις

Διπλωματική Εργασία

Δημήτριος Τρουλλινός

Επιτροπή

Επιβλέπων :	Γεώργιος Χαλκιαδάκης, Αναπληρωτής Καθηγητής
Μέλος Επιτροπής :	Μιχαήλ Γ. Λαγουδάκης, Αναπληρωτής Καθηγητής
Μέλος Επιτροπής :	Βασίλης Σαμολαδάς, Αναπληρωτής Καθηγητής

Χανιά, Σεπτέμβριος 2019

Abstract

Cooperative games model the formation of coalitions of rational agents that come together to gain some form of utility which they would have otherwise been unable to collect by acting alone. *Hedonic games*, then, constitute the class of cooperative games that models agents with hedonic preferences, that is, agents who have preferences over their very coalitional membership, i.e. the identities of others in their coalition. Thus, an agent's utility in such settings mirrors the satisfaction yielded from its assembled coalition. Now, *cooperative games with externalities*, or *in partition function form*, consider that agent utility is influenced by the partition of the agents space, i.e., the set of all disjoint coalitions currently in place. Existing studies, however, have not so far addressed hedonic games with externalities.

At the same time, *uncertainty* is prevalent in most realistic cooperative game environments, and hence intra-agent collaboration under uncertainty is a topic widely studied. However, uncertainty in hedonic game settings has received only limited attention in the literature to date.

Against this background, in this diploma thesis we first extend the formal definition of two well-known classes of hedonic games, namely additively separable hedonic games and boolean hedonic games, to partition function form. Then, we combine the aforementioned paradigms, and focus on agents in hedonic games with externalities who are unaware of their own preferences over partitions. We demonstrate how to extract these hidden preferences by employing well-established supervised learning methods—namely linear regression, linear regression with basis functions, and feed forward neural networks—and adapting them to the problem at hand. In the process, we make use of an evaluation metric specifically designed to evaluate the prediction accuracy of machine learning methods used to infer the underlying hedonic preferences over partitions. In addition, we show how an agent can use Gaussian mixture models to generate sets of potentially satisfactory partitions to propose in multi-agent negotiations. Finally, we put forward two novel coalition formation protocols that engage agents with hidden and conflicting preferences; and which are designed with the aim of maximizing social welfare, without the presence of a centralized entity or the ability to share information among agents.

Abstract in Greek

Τα *συνεργατικά παίγνια* μοντελοποιούν την δημιουργία συνασπισμών από ορθολογικούς πράκτορες, οι οποίοι ενώνουν τις δυνάμεις τους για να απολαύσουν κάποιας μορφής ωφέλεια, η οποία θα ήταν αδύνατον να αποκομιστεί με ατομική δράση. Τα *ηδονικά παίγνια*, από την άλλη, είναι μια κατηγορία συνεργατικών παιγνίων που μοντελοποιεί πράκτορες με ηδονικές προτιμήσεις, δηλαδή πράκτορες που έχουν προτιμήσεις σχετικά με την ταυτότητα των υπόλοιπων στο συνασπισμό τους. Ουσιαστικά λοιπόν, οι απολαβές ενός πράκτορα σε τέτοια περιβάλλοντα αντικατοπτρίζουν την ικανοποίηση που αποκομίζει από τον σχηματισμένο συνασπισμό του. Τώρα, τα *συνεργατικά παίγνια με εξωτερικές επιδράσεις* ή *σε μορφή συνάρτησης διαμέρισης* θεωρούν ότι οι απολαβές επηρεάζονται από τη διαμέριση του χώρου των πρακτόρων, δηλαδή το ποιό είναι το σύνολο των ήδη σχηματισμένων συνασπισμών. Μέχρι σήμερα, δεν υπάρχουν εργασίες που να έχουν εξετάσει ηδονικά παίγνια με εξωτερικές επιδράσεις.

Την ίδια στιγμή, η *αβεβαιότητα* σε περιβάλλοντα μπορεί να παρατηρηθεί σε πολλά ρεαλιστικά περιβάλλοντα συνεργατικών παιγνίων, και ως εκ τούτου η συνεργασία μεταξύ πρακτόρων υπό αβεβαιότητα είναι ένα θέμα που έχει μελετηθεί ευρέως.

Υπό το πρίσμα αυτό, στην παρούσα Διπλωματική Εργασία αρχικά επεκτείνουμε τον τυπικό ορισμό δύο πολύ γνωστών κλάσεων ηδονικών παιγνίων, συγκεκριμένα των additively separable hedonic games και των boolean hedonic games, σε μορφή συνάρτησης διαμέρισης. Κατόπιν, συνδυάζουμε τις προαναφερθείσες ιδέες και εστιάζουμε σε πράκτορες σε ηδονικά παιχνίδια με εξωτερικές επιδράσεις που δεν έχουν επίγνωση των ίδιων προτιμήσεων. Δεικνύουμε πώς να αποσπάσουμε αυτές τις κρυφές προτιμήσεις, χρησιμοποιώντας καθιερωμένες μεθόδους επιβλεπόμενης εκμάθησης—και συγκεκριμένα γραμμική παλινδρόμηση, γραμμική παλινδρόμηση με συναρτήσεις βάσης, και προωθητικά νευρωνικά δίκτυα—αφού τις προσαρμόσουμε στο τρέχον πρόβλημα. Επιπλέον, αξιοποιούμε μια μετρική ειδικά σχεδιασμένη για την αξιολόγηση της απόδοσης μεθόδων μηχανικής μάθησης που προσπαθούν να εκμαιεύσουν τις υποβόσκουσες ηδονικές προτιμήσεις σχετικά με διαμερίσεις. Στη συνέχεια, εξοπλίζουμε τους πράκτορες με την ικανότητα να δημιουργούν νέες εν δυνάμει ικανοποιητικές για αυτούς διαμερίσεις με χρήση μίξης Γκαουσιανών μοντέλων, με στόχο την πρόταση διαμερίσεων σε διαπραγματεύσεις πολλαπλών πρακτόρων. Τέλος, προτείνουμε δύο καινοφανή πρωτόκολλα σχηματισμού συνασπισμών, που εμπλέκουν πολλαπλούς πράκτορες με κρυφές και συγκρουόμενες προτιμήσεις, και τα οποία επιχειρούν να μεγιστοποιήσουν την κοινωνική ευημερία, χωρίς την παρουσία μίας κεντρικής οντότητας.

Acknowledgments

First and foremost, I would like to deeply thank my supervisor, Professor Georgios Chalkiadakis, for all his guidance and trust throughout this thesis, and for motivating me to continue my studies on topics related to Multi-Agent Systems. I also thank the members of the Intelligent Systems Laboratory, Thalia and Antonis, for our fruitful discussions, and especially Athina for her help and our excellent collaboration.

Second, I wish to thank the members of the committee, Professor Michael G. Lagoudakis and Professor Vasilis Samoladas, for their helpful comments and time.

Moreover, I want to express my gratitude towards my dear friends and colleagues Kostas, Michalis and Mike. We matured together, both academically and personally, and had moments I will always remember. I also feel very grateful to Angeliki for being by my side and always encouraging me, and to Spyros, Manolis and Chris for our friendship and all the good times we spent together.

Last but not least, I wish to thank my family for their unconditional love and support, not only throughout my studies, but also in every part of my whole life. Without them, I would not be able to fulfill my ambition of studying on this field.

Contents

List of Figures

List of Abbreviations

ASHG	Additively Separable Hedonic Game. 7, 23, 24
BHG	Boolean Hedonic Game. 7, 23, 27
EM	Expectation Maximization. 19–21
GMM	Gaussian Mixture Model. 18, 19, 23, 36–38, 40, 41, 43, 44, 46, 47, 53, 57, 58, 65, 69, 70
LR	Linear Regression. 9, 10, 12, 31, 50, 52
LR-RBF	Linear Regression with Radial Basis Function. 9, 31, 50
NN	Feed-Forward Neural Network. 10–12, 31, 35, 50, 58
PFF-ASHG	Additively Separable Hedonic Game in Partition Function Form. 25, 28, 30, 35, 39, 42, 50, 52, 53, 58, 65, 69, 70
PFF-BHG	Boolean Hedonic Game in Partition Function Form. 28, 30, 35, 39, 40, 42, 50, 52, 53, 58, 65, 69, 70
QP	Qualitative Proximity. 32, 33, 35, 36, 47, 50, 52, 57, 58
TPE	Tree-structured Parzen Estimator. 17, 35, 36, 40, 50, 57, 58

Chapter 1

Introduction

Hedonic games (Aziz, Savani, and Moulin, 2016) are a class of *cooperative* games that model agents with preferences over the composition of their own coalition, specifically the identities of other agents in the same group. In this context, agents receive payoff that corresponds to the satisfaction gained by a formed coalition. This payoff cannot be transferred, therefore *hedonic games* are a class of *non transferable utility games*. Agents can be modelled with *hedonic preferences* in a plethora of real-life applications, such as *scheduling group activities* (Darmann et al., 2012), *task allocation for a swarm of multiple agents* (Jang, Shin, and Tsourdos, 2018), *network partitioning algorithms as cooperative games* (Avrachenkov et al., 2018) among others. In this thesis, we study *hedonic games with externalities*, i.e. games with hedonic preferences affected by the composition of all coalitions. We are interested in agents in unknown environments, that attempt to acquire knowledge over their underlying preferences through interaction with others. We also consider challenging environments with high complexity of preferences, where agents are not solely interested in their own coalition.

1.1 Motivation

In cooperative game theory, existing studies on *games with externalities*, or *games in partition function form* (Chalkiadakis, Elkind, and Wooldridge, 2011), (Thrall and Lucas, 1963), where the valuation of a coalition depends on all formed coalitions in a given set of agents N, do not consider hedonic game settings. To the best of our knowledge, all studies on hedonic games so far, suggest that the utility an agent i assigns to a coalition C depends only on the structure of C. Agents with such preferences are affected by all synergies in a collaboration scheme, and examine collaborations among other agents, even beyond their own coalition.

Potential applications can emerge in settings such as competitive teams (e.g. a coding competition), where contestants can have preferences over a competitive team, as they reckon that each team's performance will depend on its composition. Furthermore, we can consider settings where the various synergies in a set of players affect each individual differently. For example, consider a class course that includes a group project, and students are classified either as hard-working or lazy. Both types of students want to collaborate with hard-working students for obvious reasons. Naturally, for a group to be formed, both parties need to comply. Therefore, this will result in groups either with multiple hard-working students, or comprised of a majority of lazy students that could not collaborate. Evidently, this is not a game with externalities, as each student cares only for its own group. However, a professor who wishes the utmost participation and knows from experience that hard-working students motivate the lazy ones, would rather prefer a more uniform distribution of both types into groups. So, his preferences are hedonic with externalities, and also conflicting with hard-working students. We can also consider settings such as task allocation strategies among agents, in which they collaborate to perform various tasks and are interested in achieving load balance across all different tasks, but can also have potentially different objectives, or valuations of collaboration schemes.

In many realistic applications, complete information about hedonic preferences is not available, and current studies have focused in hedonic games with incomplete information. In Sliwinski and Zick, 2017, the authors examine the ability of agents to learn their hidden preferences in various hedonic game classes, with a probabilistic model (Probably Approximately Correct (PAC)) that approximates the "true" utility function with a polynomial number of samples, and also attempt to find a stable coalition structure. Additionally, in Georgara, Ntiniakou, and Chalkiadakis, 2018, the authors propose the use of Probabilistic Topic Modeling (via the online version of Latent Dirichlet Allocation) to capture an agent's underlying utility function. However, to the best of our knowledge, no studies exist on extracting underlying preferences over partitions in hedonic games.

1.2 Contributions

Against this background, we combine the aforementioned paradigms, that of hedonic games in partition function form and of uncertainty. This thesis focuses on agents with *hidden preferences over partitions*. We attempt to acquire an approximation of their utility function via supervised learning techniques, and equip them with the ability to propose new partitions from past observations via generative models. Additionally, we put forward two coalition formation protocols specifically designed for partition function games, aiming to maximize the social welfare of participating agents, considering that no centralized entity exists. In our approach, we:

- Extend two well-known *hedonic games* classes, namely *additively separable hedonic games* and *boolean hedonic games* to *partition function form*
- Demonstrate how supervised learning methods can be utilized in order to extract an agent's hidden preferences for both game settings, propose an input encoding of partitions and employ an evaluation metric, specifically designed for this problem
- Show for the first time how an agent can employ a *generative model* in order to propose new partitions. We examine *Gaussian Mixture Models* for generating new partitions that attempt to satisfy an agent's preferences
- Propose two *coalition formation protocols* that contain multiple agents with *hidden* and *conflicting* preferences over partitions

Part of this work has already been published in *"Extracting Hidden Preferences over Partitions in Hedonic Cooperative Games"*, coauthored by Athina Georgara, Dimitrios Troullinos and Georgios Chalkiadakis, and published *In Proc. of the 12th International Conference on Knowledge Science, Engineering, and Management (KSEM-2019), Athens, Greece, August 2019.*

1.3 Thesis Outline

In Chapter 2 we provide the necessary theoretical background regarding both hedonic games and machine learning. Next, in Chapter 3 we formally define two hedonic games classes in partition function form, and also discuss our approach regarding these new game settings. Then, in Chapter 4 we demonstrate our experimental evaluations, discuss results and various trade-offs that emerge. And finally, in Chapter 5 we summarize our approach and consider potential future work.

Chapter 2

Theoretical Background

In this chapter we discuss the Theoretical Background of topics regarding this Thesis. Discussion topics involve *hedonic games*, *regression models*, *k-means* & *Gaussian mixture models* (*unsupervised learning models*) and *hyper-parameter optimization*.

2.1 Non-Transferable Utility Games

For a non-empty set of agents $N = \{1, \cdots, n\}$ and a set of all possible coalitions, i.e. all subsets of agents $\forall S \subseteq N$ that collaborate for a task or a common goal, a *characteristic function game* G assigns a utility (payoff) $v(S)$ for any coalition S of a set of agents N. When the distribution of this payoff $v(S)$ to agents i, $\forall i \in S$, is not determined, then we refer to *transferable utility* games. However, there are instances where the utility is not transferable among agents who collaborate. Games with these settings are referred to as *non-transferable utility* games (NTU games).

Instead of a *characteristic function game*, NTU games involve a set of *choices* $\Lambda = \{\lambda, \lambda_1, \cdots\}$ for each coalition S, and agents have *preferences* over these choices, captured from *preference relations* (Chalkiadakis, Elkind, and Wooldridge, 2011).

Definition 2.1. (Chalkiadakis, Elkind, and Wooldridge, 2011) A *preference relation* on Λ is a binary relation $\succsim \subseteq \Lambda \times \Lambda$, which is required to satisfy the following properties:

1. *Completeness:* For every $\{\lambda, \lambda^{'}\} \subseteq \Lambda$, we have $\lambda \succsim \lambda^{'}$ or $\lambda^{'} \succsim \lambda$;
2. *Reflexivity:* For every $\lambda \in \Lambda$, we have $\lambda \succsim \lambda$; and
3. *Transitivity:* For every $\{\lambda_1, \lambda_2, \lambda_3\} \in \Lambda$, if $\lambda_1 \succsim \lambda_2$ and $\lambda_2 \succsim \lambda_3$, then $\lambda_1 \succsim \lambda_3$.

2.1.1 Non-Transferable Utility Games in Partition Function Form

NTU-games can be extended to games in *partition function form* or games with *externalities*, as there are settings where the choices Λ can be affected by the coalition structure formed by all agents in N, that is, the set of all coalitions that form a *partition* π of the game.

Definition 2.2 (NTU games in Partition Function Form (Saad et al., 2012))**.** A coalitional game in partition function form (PFF) with non-transferable utility (NTU) is defined by a pair $\langle N, V\rangle$, where N is the set of players, and V is a mapping such that for every $\pi \in \Pi$ and every coalition $S \subseteq N,\ S \in \pi$, $V(S, \pi)$ is a closed convex subset of $\mathbb{R}^{|S|}$ that contains the payoff vector that players in S can achieve. Alternatively, if we consider a payoff vector in $\mathbb{R}^n$ for every coalition $S \subseteq N$ (let for any $i \notin S$ the corresponding payoff be 0 or $-\infty$), then V can be viewed as a mapping $V : \mathcal{E}_N \rightarrow \mathbb{R}^n$ that assigns to n-vector of real numbers to each embedded coalition (S, π).

2.2 Hedonic Games

Hedonic games (Banerjee, Konishi, and Sönmez, 2001) are a subclass of NTU-games that aim to capture an agent's *hedonic* preferences over coalitions S. In this setting, agents have preferences only over the identities of other agents they collaborate with, and their preference relations reflect this.

Definition 2.3. (Aziz, Savani, and Moulin, 2016) Let N be a finite set of agents. A *coalition* is a non-empty subset of N. Let $N_i = \{S \subseteq N : i \in S\}$ be the set of all coalitions (subsets of N) that include agent $i \in N$. A *coalition structure* is a partition π of agents N into disjoint coalitions. A *hedonic coalition formation game* is a pair $(N, \succsim)$, where $\succsim$ is a *preference profile* that specifies for every agent $i \in N$ a reflexive, complete, and transitive binary relation $\succsim_i$ on N_i. We call $\succsim_i$ a *preference relation.*

2.2.1 Additively Separable Hedonic Games

Separability of preferences in cooperative game settings states that adding an agent i in a particular coalition S, then $S^{'} = S \cup \{i\}$ preference to S is dependent on the preference for agent i.

Definition 2.4. (Separability (Aziz, Savani, and Moulin, 2016)) A game $(N, \succsim)$ is called *separable* if for every agent $i \in N$, coalition $S \in N_i$, and agent j not in S, we have the following:

- $S \cup \{j\} \succ_i S$ if and only if $\{i, j\} \succ_i \{i\}$;

- $S \cup \{j\} \prec_i S$ if and only if $\{i, j\} \prec_i \{i\}$; and
- $S \cup \{j\} \sim_i S$ if and only if $\{i, j\} \sim_i \{i\}$.

Additive separable preferences are a subclass of separable preferences. In an *Additively Separable Hedonic Game* (ASHG) , an agent's utility for a given coalition is the sum of the utilities it assigns to other members of that coalition. Formally, each agent $i \in N$ assigns to each agent $j \in N$ a value $b_i^j \in \mathbb{R}$ and the utility of coalition S is defined as $v_i(S) = \sum_{j \in S} b_i^j$ (Elkind and Wooldridge, 2009).

Utility of an agent i in a singleton coalition is $v_i(\{i\}) = 0$, $\forall i \in N$. For coalitions $S, T \in N_i$, we have $S \succsim_i T$ if and only if $v_i(S) \geq v_i(T)$. ASHGs are *symmetric* if $b_i^j = b_j^i$, and *non-symmetric* otherwise (Aziz, Savani, and Moulin, 2016).

2.2.2 Boolean Hedonic Games

Boolean Hedonic Games (Aziz, Harrenstein, et al., 2016) (BHGs) are Hedonic Games with *dichotomous preferences*. For agent i, any coalition $S \subseteq N$, where $i \in S$, is either *satisfactory* or *unsatisfactory*. So, there are two disjoint sets N_i^+ and N_i^-, and $S \in N_i^+$ if S is satisfactory, or $S \in N_i^-$ otherwise. Agent i strictly prefers any coalition in N_i^+ in relation to any coalition in N_i^-, and is indifferent between coalitions that belong to the same set.

Example 1. Consider a BHG with 4 agents, whose dichotomous preferences are as follows (coalitions separated by comma indicate that the agent is indifferent):

- agent 1: $\{1, 2\}, \{1, 3\}, \{1, 4\} \succ_1 \{1\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 2, 3, 4\}$
- agent 2: $\{2, 1\}, \{2, 3\}, \{2, 4\}, \{2, 3, 4\} \succ_2 \{2\}, \{2, 1, 3\}, \{2, 1, 4\}, \{2, 1, 3, 4\}$
- agent 3: $\{3\}, \{3, 1\}, \{3, 1, 2\}, \{3, 1, 4\} \succ_3 \{3, 2, 4\}, \{3, 2\}, \{3, 4\}, \{3, 1, 2, 4\}$
- agent 4: $\{4\}, \{4, 1, 2, 3\} \succ_4 \{4, 1\}, \{4, 2\}, \{4, 3\}, \{4, 1, 2\}, \{4, 1, 3\}, \{4, 2, 3\}$

We can observe that agent 1 strictly prefers coalitions of size 2 to any other coalitions and agent 2 strictly prefers coalitions of size 2 or a coalition with players 3 and 4. Furthermore, agent 3 prefers either a singleton coalition or any coalition with agent 1 (except the grand coalition) and agent 4 strictly prefers to either being a singleton or be in the grand coalition.

Modelling of Boolean Preferences

Instead of providing a complete preference profile (as showcased in example 1) by aggregating all possible coalitions with their pairwise relations, we can adopt a compact representation, introduced by Aziz, Harrenstein, et al., 2016 and as described by Georgara, Ntiniakou, and Chalkiadakis, 2018. The proposed formula γ_i consists of two subsets of agents, the "must be included" set $included_i$, and "must be excluded" set $excluded_i$. Agent i is satisfied with a coalition S when $\forall j \in included_i, j \in S$ and $\forall j \in excluded_i, j \notin S$. Agent i can have multiple pairs of $\langle included_i, excluded_i \rangle$, and a coalition $S \in N_i^+$ if it satisfies at least one pair. Thus, the DNF [1] form of a *formulae* with multiple γ_i's is:

$$\boldsymbol{\gamma}_i = \bigvee_{l=1:L} \gamma_{i,l} = \gamma_{i,1} \bigvee \gamma_{i,2} \bigvee \cdots \bigvee \gamma_{i,L} \tag{2.1}$$

where l denotes the index of formula $\gamma_{i,l}$, L is the size of pairs, and:

$$\gamma_{i,l} = \langle included_{i,l}, excluded_{i,l} \rangle \tag{2.2}$$

2.3 Supervised Learning

In the context of Machine Learning, Supervised Learning (SL) methods learn from available labeled data. The goal of SL is to approximate the function that labels data. It is called "supervised" because of the presence of the outcome variable to guide the learning process (Hastie, Tibshirani, and Friedman, 2001). Examples such as labeling an email as spam or not spam, in which the aim is to assign each input vector to one of a finite number of discrete categories, are called *classification* problems. If the desired output consists of one or more continuous variables, then the task is called *regression* (Bishop, 2006).

2.3.1 Linear Regression

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i \tag{2.3}$$

This model is a linear function of the input variables x_i, $i = 1, ..., N$, where $\mathbf{x} = (x_1, ..., x_N)^T$. Weight parameters $\mathbf{w} = (w_0, ...w_N)^T$ are estimated based on training data. w_0 is a bias parameter that allows for any fixed offset on target values. $y(\mathbf{x_k}, \mathbf{w}) = y_k$ is the approximation value for a given input $\mathbf{x_k}$, and attempts to predict the target value t_k. The model creates the same linear function for all training data by adjusting its weight parameters.

[1] Disjunctive normal form, a logical formula consisting of a disjunction of one or more conjunctions, i.e. an OR of multiple ANDs.

For an one-dimensional input vector x, and a set of samples containing (x_k, t_k), figure 2.1 shows the approximated function $y(\mathbf{x}, \mathbf{w})$:

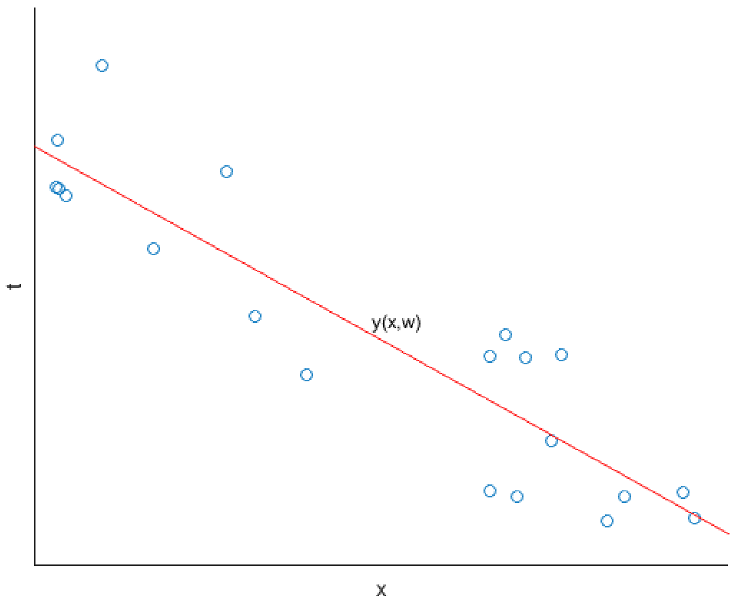

Figure 2.1: Linear Regression with an one-dimensional input vector x

Weights are estimated by applying the normal equations for the least squares problem for LR: (Bishop, 2006):

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} \tag{2.4}$$

- $\mathbf{W}$: weights' vector, with dimensions $M \times 1$ (M = number of weight parameters)
- $\mathbf{X}$: vector of $\mathbf{x_k}$ values, with dimensions $K \times M$ (K = number of samples)
- $\mathbf{t}$: target values' vector, with dimensions $K \times 1$

2.3.2 Linear Regression with Radial Basis Functions

To overcome the problem of bias (assumption of linearity) that LR introduces, radial basis functions can be used to approximate non-linear functions, resulting in Linear Regression with Radial Basis Function (LR-RBF).

$$y(\mathbf{x}, \mathbf{w}) = \sum_{i=0}^{M-1} w_i\phi_i(\mathbf{x}) \tag{2.5}$$

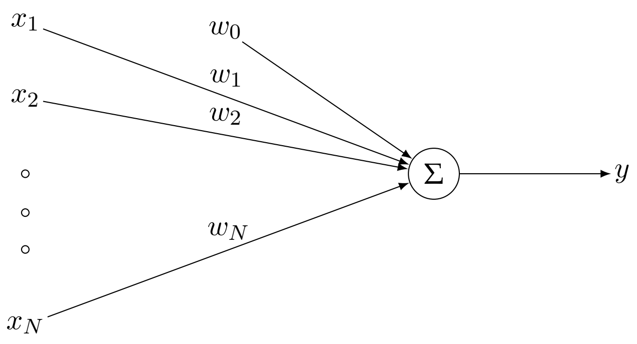

Figure 2.2: Visual Representation of a Linear Regression Model

$\phi_0(\mathbf{x}) = 1$ is defined in order to have w_0 as a bias parameter, regardless of the form of ϕ. M is the number of basis functions that the model uses. This model's efficiency depends on the form of the basis function and the size of M. The determination of the set of the basis functions depends on the problem at hand.

Gaussian Basis Functions are frequently used, with the form:

$$\phi_i(\mathbf{x}) = exp\Big\{ - \frac{\|\mathbf{x} - m_i\|^2}{2\sigma^2} \Big\} \tag{2.6}$$

where m_i reflects the location of each ϕ_i in the N-dimensional space, and σ the scale of each value, which is common for every ϕ_i. Now, each basis function ϕ_i, is an exponential, related with a centre vector $m_i \in \mathbb{R}^{\|X\|}$, and a standard deviation $\sigma \in \mathbb{R}^+$. For computing the centre vector m_i of each ϕ_i, clustering methods such as k-means (discussed below, (2.5)) are used, while the standard deviation is usually the same for all ϕ_i.

Weights estimation Equation (2.4) is modified for this model, in order to integrate $\phi_i(\mathbf{x})$:

$$\mathbf{W} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{t} \tag{2.7}$$

- $\mathbf{\Phi}$: vector of $\phi_i(\mathbf{x}_k)$ values, with dimensions $K \times M$ (K = number of samples)

2.3.3 Feed-Forward Neural Networks

Thinking of a *Feed Forward Neural Network* (NN), one can easily interpret the mathematical expression of one LR model as a simple NN. But the structure of a NN can have more layers. The basic form of a neural network model is a series of layers

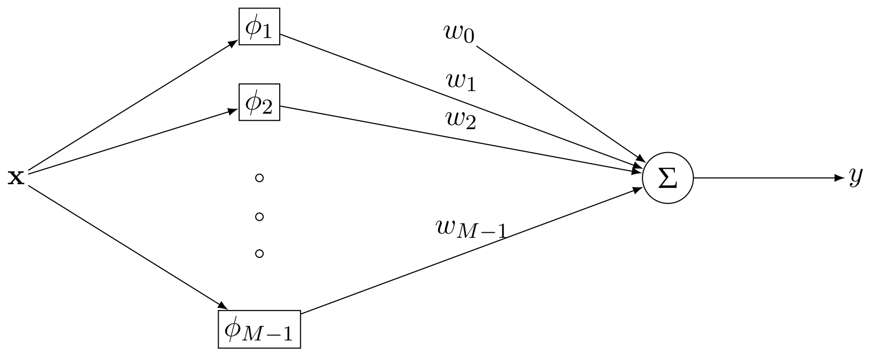

Figure 2.3: Illustration of Linear (with Basis Function) Regression Model

of computational units (Goodfellow, Bengio, and Courville, 2016). First we have M linear combination of the inputs, which take the form:

$$a_j = w_{j0}^{[1]} + \sum_{i=1}^{N} w_{ji}^{[1]} x_i \tag{2.8}$$

where $j = 1, ..., M$ and M is the number of combinations. $i = 1, ..., N$ is the index of vector $\mathbf{x}$ and w_0 is the bias parameter. [1] indicates that these parameters are in the first layer of the network. This equation is identical to Equation (2.3). An activation function h (possibly non-linear) can be used for all a_j. Outputs $h(a_j)$ are the input for the next layer of computations:

$$b_l = h_2\Big(w_{j0}^{[2]} + \sum_{j=1}^{M} w_{lj}^{[2]} h_1(a_j)\Big) \tag{2.9}$$

where $l = 1, ..., L$ is the number of output values and [2] indicates that these parameters are in the second layer of the network. We can express the relationship between input of the first layer and output of the second layer by combining Equations (2.8),(2.9):

$$b_l = h_2\Big(w_{j0}^{[2]} + \sum_{j=1}^{M} w_{lj}^{[2]} h\Big(w_{j0}^{[1]} + \sum_{i=1}^{N} w_{ji}^{[1]} x_i\Big)\Big) \tag{2.10}$$

For a 2-layered NN (1 hidden layer and 1 output layer), b_l are the output values, so: $y_l(\mathbf{x}, \mathbf{w}) = b_l$.

$$y_l(\mathbf{x}, \mathbf{w}) = h_2\Big(w_{j0}^{[2]} + \sum_{j=1}^{M} w_{lj}^{[2]} h_1\Big(w_{j0}^{[1]} + \sum_{i=1}^{N} w_{ji}^{[1]} x_i\Big)\Big) \tag{2.11}$$

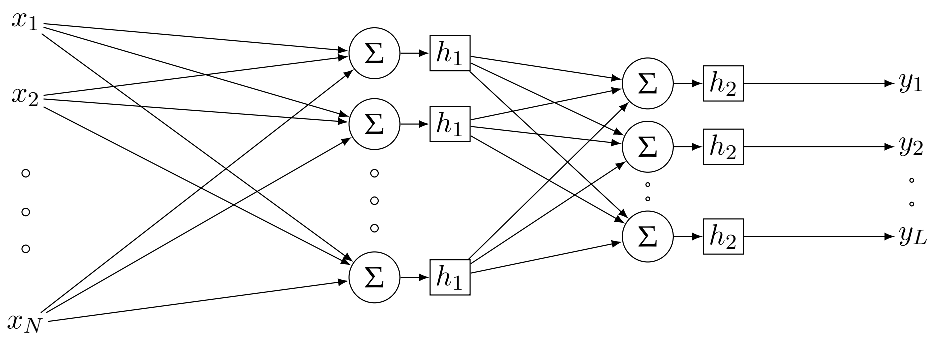

Figure 2.4: Illustration of a 2-layered Feed Forward Neural Network (1 hidden layer). Weights on the edges and bias weights are not visible.

In essence, this structure is a combination of multiple LR models with non-linear activation functions that can approximate a wide range of functions. Beyond two layers, the same philosophy is extended for a much more complex network.

Activators of hidden layers h_i need to be non-linear, since linear activators will result in a network with capabilities equivalent to a LR model. Popular choices for activator functions are:

- Sigmoid:
$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{2.12}$$
- Rectified Linear Unit (ReLU):
$$\text{ReLU}(x) = max(0, x) \tag{2.13}$$
- Hyperbolic tangent (tanh):
$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.14}$$

NNs that serve as regression models have linear output activators (identity activation function), as they approximate target values $\mathbf{t} \in \mathbb{R}^d$, whereas in classification models, sigmoid or hyperbolic tangent functions are preferred.

When a NN is trained, its weights are actually "tuned" in order to approximate the desired function. This task is not trivial, and NNs rely primarily on Gradient Descent Optimization Algorithms for training.

2.3.4 Gradient Descent Optimization Algorithms

Gradient Descent (GD) is a way to minimize an objective function $J(\theta)$ with parameters $\theta \in \mathbb{R}^d$ by updating θ in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$. In all variants of GD, a parameter η is used as the learning rate, that determines the size of the steps we make in each update to reach a (local) minimum (Ruder, 2016).

Batch gradient descent

Batch gradient descent (BGD) computes parameters θ by calculating the gradient of the cost function $J(\theta)$ for the entire training dataset.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.15}$$

This means that only one update of the parameters is performed. Although this method is guaranteed to reach a (global or local, depending on the form of function) minimum of $J(\theta)$, this computation can be very slow for a large dataset, and very demanding if it cannot fit in main memory (Ruder, 2016). Furthermore, it cannot be used for *online training*.

Stochastic gradient descent

Unlike BGD, stochastic gradient descent (SGD) performs an update of parameters θ for each training sample $(x^{(i)}, y^{(i)})$.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.16}$$

SGD is much faster than BGD, as it calculates gradients for each sample, and is suited for *online training*. Characteristic of SGD is that it fluctuates, i.e. does not monotonically decrease the value of $J(\theta)$, and it can result in big changes in θ (especially for large values of η). This can potentially be beneficial, as it might result in a new and better local minima. However, fluctuation also makes convergence much more difficult (Ruder, 2016).

Mini-batch gradient descent

Mini-batch gradient descent (MGD) combines the benefits of BGD and SGD by performing updates for every mini-batch of n training samples.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}, y^{(i:i+n)}) \tag{2.17}$$

MGD has more stable convergence than SGD, and computes gradients much faster than BGD, as it requires a batch of fixed number of samples n for each update.

A big challenge for the aforementioned algorithms is the choice of learning rate η. Performance is highly affected by this choice, and our data. Especially for sparse data, choosing the same learning rate for all parameters might not be effective. Thus, algorithms with adaptive learning rate have been developed and are discussed below.

Adaptive Gradient Algorithm (Adagrad)

Adagrad was introduced by Duchi, Hazan, and Singer, 2011 and adapts its learning rate to the parameters, by performing larger updates for infrequent, and smaller updates for frequent parameters (Ruder, 2016).

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_{\theta_t} J(\theta_{t,i}) \tag{2.18}$$

Adagrad updates at each time step t its parameters θ. Index i denotes the learning rate for different parameters in each time step. $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix, and each diagonal value $G_{t,ii}$ is the sum of squares of gradients up to time step t. ϵ is a smoothing term that avoids division by zero.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla_{\theta_t} J(\theta_t) \tag{2.19}$$

Equation 2.19 is a vectorized form of Equation 2.18 for all i.

The main issue of Adagrad is that by adding the sum of squares of gradients for $G_{t,ii}$, this value increases as t increases. This results in making the fraction $\frac{\eta}{\sqrt{G_{t,ii}+\epsilon}}$ (the adaptive learning rate) smaller after each update. Thus, after many timesteps t, new updates are extremely small. The following algorithm overcomes this weakness.

Root Mean Square Propagation (RMSProp)

RMSProp was introduced in a lecture by Hinton, Srivastava, and Swersky, n.d. and is an unpublished method. It overcomes Adagrad's drawback of monotonically decreasing learning rate by defining the decaying average of squared gradients as:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2 \tag{2.20}$$

where $g_t = \nabla_{\theta_t} J(\theta_t)$. $E[g^2]_t$ is the decaying average that is dependent only on the previous one $E[g^2]_{t-1}$ and the current squared gradients g_t^2, using a parameter γ as momentum, that is suggested by Hinton to be set to 0.9 (Ruder, 2016).

Replacing Adagrad's G_t with $E[g^2]_t$ we obtain:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \odot \nabla_{\theta_t} J(\theta_t) \tag{2.21}$$

Adaptive Moment Estimation (Adam)

Adam optimization algorithm was introduced by Kingma and Ba, 2014 and attempts to further improve upon the prior methods by computing the mean m_t of gradients with a decaying parameter β_1, and the squared mean υ_t of gradients with a decaying parameter β_2.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.22}$$

$$\upsilon_t = \beta_2 \upsilon_{t-1} + (1 - \beta_2) g_t^2 \tag{2.23}$$

β parameters, ranging from 0 to 1, introduce bias to the above estimates due to the recursive nature of Equations (2.22), (2.23). This leads to ignoring past gradients as the distance from current timestep t increases. However, bias-correction is performed to avoid this phenomenon.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.24}$$

$$\hat{u}_t = \frac{u_t}{1 - \beta_2^t} \tag{2.25}$$

Thus, Adam's parameters' update takes the form:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} \odot \nabla_{\theta_t} J(\theta_t) \hat{m}_t \tag{2.26}$$

2.4 Hyper-Parameter Optimization

With Hyper-Parameter Optimization, we want to find the value x_i that minimizes a function $f(x)$, i.e. $argmin_x(f(x))$. When f is computationally expensive, *Sequential Model-Based Global Optimization* (SMBO) algorithms can be used (J. S. Bergstra et al., 2011).

Algorithm 1 SMBO algorithm

```
1: function SMBO($f, M0, T, S$)  ▷ M0:initial model, T:timesteps and S:surrogate function
2:     $H \leftarrow \emptyset$  ▷ H:observations history
3:     for $t = 1$ to $T$ do  ▷ for every timestep
4:         $x^* \leftarrow argmin_x S(x, M_{t-1})$
5:         Evaluate $f(x^*)$
6:         $H \leftarrow H \cup (x^*, f(x^*))$
7:         Fit a new model $M_t$ to $H$
8:     return $H$
```

In every iteration, SMBO chooses a x^* based on a surrogate function or a model, then evaluates it with f and stores the set $(x^*, f(x^*))$ to observation history H. The final step before the next iteration is to create a new model based on H. As t increases, M_t models become more accurate, which results to new values for x^* that tend to minimize f.

Knowing that evaluation of f for a single x^* is computationally expensive, x^* values are obtained using a surrogate function or a model of f, and using observation history H. Algorithms in (J. S. Bergstra et al., 2011) optimize the Expected Improvement (EI), that is:

$$EI_{y^*}(x) = \int_{-\infty}^{\infty} max(y^* - y, 0) p_M(y|x) dy \tag{2.27}$$

"Expected improvement is the expectation under some model $M : X \rightarrow \mathbb{R}^N$ that $f(x)$ will exceed (negatively) some threshold y^*."(J. S. Bergstra et al., 2011)

2.4.1 Tree-structured Parzen Estimator Approach (TPE)

This approach has the advantage that it can be used in cases where our hyper-parameters do not have a strict number of values, and when there are dependencies, e.g. for a neural network that we choose the number of hidden layers and number of nodes as a hyper-parameter, we first choose the number of layers and then the number of nodes for each layer. Another example is when we want to compare different models that might use a different set of hyper-parameters. The TPE algorithm instead of modeling $p(y|x)$, it models $p(x|y)$ using two different densities:

$$p(x|y) = \begin{cases} l(x) \text{ if } y < y^* \\ g(x) \text{ if } y \geq y^* \end{cases} \tag{2.28}$$

This means that $l(x)$ is the distribution formed using observations x^*, where $f(x^*) < y^*$ and $g(x)$ is formed with observations that satisfy $f(x^*) \geq y^*$. TPE has a parameter $\gamma = p(y < y^*)$ in order to choose y^*. Applying TPE to (2.27), we get:

$$EI_{y^*}(x) = \frac{\gamma y^* l(x) - l(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma l(x) + (1-\gamma)g(x)} \propto (\gamma + \frac{g(x)}{l(x)}(1-\gamma))^-1 \tag{2.29}$$

To maximize the EI, we need to minimize the ratio $\frac{g(x)}{l(x)}$, i.e. the algorithm chooses values of x that maximize $l(x)$ and minimize $g(x)$ in every iteration.

2.5 k-means Clustering

k-means is an Unsupervised Learning [2] method that creates a specified number of k clusters from training data (unlabeled) and was first introduced by MacQueen et al., 1967.

For a given dataset $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ of N observations of a random variable $\mathbf{x}$, where each $\mathbf{x}_n \in \mathbb{R}^D$, D is the dimensionality of $\mathbf{x}$, we want to partition this dataset to a specified number of clusters K. Mean values $\boldsymbol{\mu}_k \in \mathbb{R}^D, 1 \leq k \leq K$ are assigned to corresponding clusters C_k, and they are calculated from the assigned data points $\forall \mathbf{x}_n \in C_k$. The objective function to be minimized is:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \tag{2.30}$$

Variable $r_{nk} \in \{0, 1\}$ is an indicator that describes in which cluster a data point $\mathbf{x}_n$ is assigned. If $\mathbf{x}_n \in C_k$, then $r_{nk} = 1$, else $r_{nk} = 0$. Equation (2.30) is the sum of squares of the distances of each data point to its assigned vector $\boldsymbol{\mu}_k$ (Bishop, 2006).

[2]Unsupervised Learning methods attempt to find patterns on unlabeled data and capture their internal structure.

The intention is to acquire values for r_{nk} and $\boldsymbol{\mu}_k$, that is to compute the clusters C_k, that minimize J (Bishop, 2006). For this optimization problem, an iterative algorithm can be used, in which each iteration contains two successive steps, the Expectation Step (E-step), and the Maximization Step (M-step). First, choose randomly $\boldsymbol{\mu}_k$ values. Then follow the E-step, M-step and repeat until convergence, i.e. no further changes after update, or a maximum number of iterations is exceeded. Although convergence is assured, as each update of variables reduces the value of J, it converges to a local minimum of J, that might not be global.

- E-step: update r_{nk} values from Equation (2.31), i.e. assign data points to clusters
- M-step: update $\boldsymbol{\mu}_k$ values from Equation (2.32)

$$r_{nk} = \begin{cases} 1, & \text{if } k = arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \\ 0, & \text{otherwise} \end{cases} \tag{2.31}$$

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}} \tag{2.32}$$

2.6 Gaussian Mixture Models

Gaussian Mixture Models (GMMs), introduced by McLachlan and Basford, 1988, can be used either as a generative or clustering model. They can be viewed as an extension of k-means clustering in the context of Unsupervised Learning. Every cluster generated from GMMs, apart from the mean value m_i, contains a covariance matrix Σ_i. This allows for more flexible cluster shapes compared to k-means, i.e. can cluster data with higher complexity more efficiently. *One of the powerful attributes of the GMM is its ability to form smooth approximations to arbitrarily shaped densities* (Reynolds, 2009).

A GMM is a weighted sum of Gaussian component densities. They are commonly used as a parametric model of the probability distribution of continuous measurements or features in biometric systems. (Reynolds, 2009)

The Gaussian mixture distribution is defined with the form:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.33}$$

where index k denotes each Gaussian component and the multivariate Gaussian distribution takes the form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right\} \tag{2.34}$$

Mixing coefficients π_k correspond to probabilities of each Gaussian component k and must satisfy both (2.35) and (2.36).

$$\sum_{k=1}^{K} \pi_k = 1 \tag{2.35}$$

$$0 \leq \pi_k \leq 1 \tag{2.36}$$

$\gamma(z_k)$ is defined as the conditional probability $p(z_k = 1|\mathbf{x})$, where z_k is the random variable that denotes which Gaussian component is active. Therefore $z_k = \{0, 1\}$ and $\sum z_k = 1$.

$$\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) = \frac{p(z_k = 1)p(z_k = 1|\mathbf{x})}{\sum_l p(z_l = 1)p(\mathbf{x}|z_l = 1)} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_l \pi_l \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} \tag{2.37}$$

For a given dataset $D = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ and $N = |D|$, the log of the likelihood function takes the form:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{2.38}$$

Expectation Maximization for Gaussian Mixture Models

Expectation Maximization (EM) was introduced by Dempster, Laird, and Rubin, 1977 and applied to GMMs by G. Saari and Merlin, 1996.

In GMMs, for a given set of unlabeled data $\mathbf{X}$, we want to maximize (2.38). Parameters of (2.38) are $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{\pi}$ arrays. When maximizing a differentiable function $f(x)$, we obtain the derivative $\frac{df(x)}{dx}$ and set it to zero. Maximizing the log-likelihood function, with respect to the $\boldsymbol{\mu}_k$ values, we obtain:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{2.39}$$

Using the conditional probability $\gamma(z_{nk})$ for each point $\mathbf{x}_n$, we obtain N_k, that expresses the size of cluster k.

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{2.40}$$

Setting the derivative of 2.38 to zero with respect to $\mathbf{\Sigma}_k$, we obtain:

$$\mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \tag{2.41}$$

Finally, for the mixing coefficients π_k, a Lagrange multiplier is used in order to satisfy the constraint in 2.35.

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{\Sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) \tag{2.42}$$

We then maximize by setting the derivative of 2.38 to zero with respect to $\mathbf{\Sigma}_k$. Thus, we obtain:

$$\pi_k = \frac{N_k}{N} \tag{2.43}$$

These results (2.39), (2.41), (2.43) can be used for the iterative EM algorithm to maximize (2.38). First, choose some initial values for all parameters ($\boldsymbol{\mu}$, $\mathbf{\Sigma}$ and $\boldsymbol{\pi}$). Then, repeat the following two steps until convergence (or a maximum number of iterations).

- E-step (expectation step): Calculation of the posterior probabilities (2.37) for every cluster k, using the assigned values of all parameters.
- M-step (maximization step): Re-estimate parameters ($\boldsymbol{\mu}$, $\mathbf{\Sigma}$ and $\boldsymbol{\pi}$) from Equations (2.39), (2.41), (2.43), with the assigned values of parameters. Finally, calculate the log-likelihood (2.38) and check for convergence (log-likelihood increase is below a threshold e). After each iteration, it is certain that the value of the log-likelihood will not decrease.

EM resembles a hill-climbing technique. It reaches a local maximum, and it is not right to assume that this is the global maximum value. However, a global maximum for log-likelihood does not necessarily provide us with a better model when taking singularities into account. Singularities can occur when any Gaussian component k has

$\boldsymbol{\mu}_k = \mathbf{x}_n$ ($\mathbf{x}_n$ can be any point of the training set), $\boldsymbol{\Sigma}_k$ is a zero matrix and $\pi_k \neq 0$. By observing (2.34) and (2.38), one can see that the log-likelihood will tend to infinity. EM avoids singularities, contrary to a simple maximum likelihood approach, as noted by Bishop, 2006. The use of EM for mixture models instead of a *gradient-based* optimizer is preferred due to the need for constraints in such models, as discussed in Murphy, 2012.

Chapter 3

Our Approach

In this chapter we present our approach for enabling collaboration pattern detection in hedonic games with externalities. Topics include *extending* hedonic games in partition function form for two particular classes, namely ASHGs and BHGs; *encoding* partitions and utility for supervised learning; employing a novel metric for evaluating approximations of preference relations; "auto-tuning" learning algorithms' parameters with hyper-parameter optimization; *generating* new partitions via GMMs and *engaging* multiple agents to two coalition formation protocols.

3.1 Hedonic Games in Partition Function Form

We extend the formal definition of Hedonic Games to Partition Function Form (Georgara, Troullinos, and Chalkiadakis, 2019). Essentially, the structure of all coalitions affects each agent's preferences.

Definition 3.1. A *hedonic game* (HG) in *partition function form* (PFF) (Georgara, Troullinos, and Chalkiadakis, 2019) is defined by a pair $\langle N, \succsim \rangle$, where N is the set of players, and $\succsim = \{\succsim^{\pi_1}, \cdot, \succsim^{\pi_m}\}$, with $|\Pi| = m$; and for all $\pi_j \in \Pi$ $\succsim^{\pi_j} = \{\succsim_1^{\pi_j}, \cdot, \succsim_n^{\pi_j}\}$, and each $\succsim_i^{\pi_j} \in N_i \times N_i$ is a complete, reflexive and transitive preference relation describing agent i's preferences over coalitions it can participate in when π_j is in place.

3.1.1 Complexity in Partition Function Form Settings

Now agents have preferences over partitions π, instead of their assembled coalition $S_i \in \pi$. The partition space Π (set of all possible partitions) is broad and grows exponentially as the number of agents increases. For a non-empty set of agents $N = \{ag_1, ag_2, \cdots, ag_n\}$, the size of all possible partitions is measured by *Bell Numbers* (Bell, 1934):

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \quad B_0 = B_1 = 1 \tag{3.1}$$

All possible coalitions for $|N|$ agents are all the subsets of N (except the empty set $\emptyset$), i.e. $2^{|N|} - 1$. For an agent i that has preferences only over the coalitions that it belongs, then all possible coalitions that contain i are $2^{|N|-1}$, [1] so for 10 agents, all possible coalitions are $2^9 = 512$, and for 20 agents, all possible coalitions are $2^{19} = 524288$. By contrast, when agents have preferences over partitions, for 10 agents, $|\Pi_{|N|=10}| = B_{10} = 115975$, and for 20, $|\Pi_{|N|=20}| = 51724158235372$. Considering an agent who is completely unaware of its utility function, we assume an underlying structure of this function that can be exploited via machine learning methods without the need of a substantial proportion of partitions from Π.

3.1.2 Additively separable hedonic games in partition function form

Generalizing ASHGs to partition function form, each agent assigns a value to any agent within each partition $\pi \in \Pi$, i.e, agent i assigns the value $b_i^j(\pi)$ to agent j when partition π is formed. Thus, the utility of embedded coalition (S, π) is now defined as $v_i(S, \pi) = \sum_{j \in S} b_i^j(\pi)$. As such, it is more natural to model ASHGs like NTU games due to their properties. Therefore, there is a mapping $V : \mathcal{E}_N \to \mathbb{R}$ such that for every embedded coalition there is an n-vector of reals:

$$V(S, \pi) = \begin{bmatrix} v_1(S, \pi) \\ v_2(S, \pi) \\ \vdots \\ v_n(S, \pi) \end{bmatrix} = \begin{bmatrix} \sum_{j \in S} b_1^j(\pi) \\ \sum_{j \in S} b_2^j(\pi) \\ \vdots \\ \sum_{j \in S} b_n^j(\pi) \end{bmatrix} \tag{3.2}$$

In contrast to what is usual in Characteristic Function Games settings, some value that agent i gets from the fact that S has been formed within π. Given this, each agent forms a preference relation that refers to embedded coalitions (rather than coalitions). This preference relation is as follows: agent i prefers embedded coalition (S, π) to (T, π'), $(S, \pi) \succsim_i (T, \pi')$, where $i \in S$, $i \in T$, if and only if $v_i(S, \pi) \geq v_i(T, \pi')$, even if S and T consist of exactly the same set of agents.

As discussed above 3.1.2, utility functions in ASHGs are dependant on $b_i^j, \forall i, j \in N$ values. In a PFF setting, agent i 's values need to depend not only on its coalition

[1] Agent i considers all possible coalitions with all other agents $N \setminus i$, $|N \setminus i| = |N| - 1$, so it considers $2^{|N|-1} - 1$ possible coalitions, but i can also form a singleton coalition $S = \{i\}$. So, all possible combinations that contain i are $2^{|N|-1}$.

$N_i \in N$, but on the whole partition π. Here, we suggest that the values b_i^j are derived from a weighted (positive weights) undirected graph, reminiscent of a social network, that describes the "communication links" between the agents.

Therefore, we now consider these values as a function $b_i^j(\pi)$ of a partition π and a graph g, containing the following three properties:

1. the distance between agent i and j; and
2. the distances between agent i and all coalitions in π; and
3. the distances between agent j and all coalitions in π.

with the form:

$$b_i^j(\pi) = \kappa_i \cdot \frac{1}{dist_g(i,j)} + \lambda_i \cdot \sum_{S \in \{\pi\}} dist_g(i,S) + \xi_i \cdot \sum_{S \in \{\pi\}} \frac{1}{dist_g(j,S)} \tag{3.3}$$

where $dist_g(i,j)$ is the shortest distance between agent i and j in graph g, calculated using Dijkstra's algorithm[2]. $dist_g(i,S), S \in N$, i.e. distance between an agent i and a coalition S in graph g is defined as follows:

$$dist_g(i,S) = \frac{1}{|S|} \sum_{j \in S} dist_g(i,j) \tag{3.4}$$

This Equation (3.4) expresses the distance of an agent i from a coalition S as the sum of distances from agents $j, \forall j \in S$, divided by the size of coalition S.

Parameters $\kappa_i, \lambda_i, \xi_i$ are unique for every agent i and they serve to differentiate each agent's preferences. They are generated from a uniform distribution $\kappa_i, \lambda_i, \xi_i \sim \mathcal{U}(0, 10)$.

Example 2. Consider a PFF-ASHG game with 4 agents, i.e. $N = \{ag_1, ag_2, ag_3, ag_4\}$ and the graph $G(V,E)$ in Figure 3.1. For $g = G(V,E)$, we obtain the following:

- $dist_g(1,2) = 5$
- $dist_g(1,3) = 6$
- $dist_g(1,4) = 8$
- $dist_g(2,3) = 3$
- $dist_g(2,4) = 5$
- $dist_g(3,4) = 2$

Let:

[2]Evidently, $dist_g(i,i) = 0, \forall i \in N$.

- $\kappa_1 = 5, \lambda_1 = 2, \xi_1 = 8$
- $\kappa_2 = 10, \lambda_2 = 0, \xi_2 = 0$
- $\kappa_3 = 7, \lambda_3 = 4, \xi_3 = 1$
- $\kappa_4 = 2, \lambda_4 = 8, \xi_4 = 5$

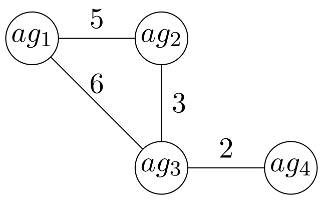

Figure 3.1: Graph G(V,E) for Example 2

Now consider a partition $\pi_k = \{\{ag_1, ag_3\}, \{ag_2, ag_4\}\}$. Values b_i^j are calculated from Equation (3.3). For agent ag_1:

$$\begin{aligned}
\upsilon_1(S, \pi_k) &= \sum_{j \in S} b_1^j(\pi_k) = b_1^3(\pi_k) \\
&= 5 \cdot dist_g(1,3) + 2 \cdot \sum_{S \in \{\pi\}} dist_g(1,S) + 8 \cdot \sum_{S \in \{\pi\}} \frac{1}{dist_g(3,S)} \\
&= 5 \cdot 6 + 2 \cdot (\frac{5+8}{2}) + 8 \cdot (\frac{2}{3+2}) \\
&= 30 + 2 \cdot 6.5 + 8 \cdot 0.2 = 30 + 13 + 1.6 = 44.6
\end{aligned}$$

While the sum contains only agents from ag_1's coalition, b_1^j values are dependant on the structure of all coalitions. For $\pi_{k,2} = \{\{ag_1, ag_3\}, \{ag_2\}\{ag_4\}\}$, payoff $\upsilon_1(S, \pi_k)$ will be different, as b_1^j values will change according to Equation (3.3).

$$\begin{aligned}
\upsilon_1(S, \pi_{k,2}) &= \sum_{j \in S} b_1^j(\pi_{k,2}) = b_1^3(\pi_{k,2}) \\
&= 5 \cdot dist_g(1,3) + 2 \cdot \sum_{S \in \{\pi\}} dist_g(1,S) + 8 \cdot \sum_{S \in \{\pi\}} \frac{1}{dist_g(3,S)} \\
&= 5 \cdot 6 + 2 \cdot (5+8) + 8 \cdot (\frac{1}{3} + \frac{1}{2}) \\
&= 30 + 2 \cdot 13 + 8 \cdot 0.83 = 30 + 26 + 6.64 = 62.64
\end{aligned}$$

3.1.3 Boolean hedonic games in partition function form

Boolean hedonic games provide a concise representation of hedonic games with dichotomous preference relations (Aziz, Harrenstein, et al., 2016). According to the dichotomous preferences model, each agent i can partition $N_i = \{S \subseteq N \setminus \{\emptyset\} : i \in S\}$ into two disjoint sets N_i^+ and N_i^-; and i strictly prefers all coalitions in N_i^+ to those in N_i^-, and is indifferent about coalitions in the same set. In boolean hedonic games, each agent i, instead of explicitly enumerating the preference relation that leads to dichotomous preferences, defines a logic formula γ_i that intuitively represents its goal of being with preferred partners; and i is satisfied if its goal is achieved, or dissatisfied otherwise. This formula can be of any form of a propositional logic language, but γ_i in (Aziz, Harrenstein, et al., 2016) involves only propositional variables *relative* to agent i, that is, denoting agents i wants or does not want to be grouped with.

Expanding BHGs to partition function form, the key idea is to partition the Π space into two disjoint sets P_i^+ and P_i^-, i.e. into a set with the partitions agent i prefers, and a set with the partitions i does not. In its generality, the use of propositional logic formulae allows us to have a compact representation, but γ_i in (Aziz, Harrenstein, et al., 2016) was meant to capture i's preferences regarding the composition of its coalition only. We extend into PFF by introducing a specific form for γ_i, which is as follows: each γ_i is *not* restricted to variables relative to agent i, but consists of multiple pairs $\langle Incl_i, \overline{Incl_i}\rangle$ connected via the logical connective *or* ($\vee$). $\langle Incl_i, \overline{Incl_i}\rangle$ is a pair of two *disjoint* sets of subcoalitions. Now, each pair is interpreted as follows: $Incl_i$ is a set of "must-include" subsets of coalitions, while $\overline{Incl_i}$ is a set of "must-not-include" subsets of coalitions. In words, the set $Incl_i$ represents desirable patterns of collaborations that a preferable to i partition must contain; while the set $\overline{Incl_i}$ indicates cooperation among agents that must be excluded from a preferable partition. Thus, a partition π satisfies the pair $\langle Incl_i, \overline{Incl_i}\rangle$ if:

- $\forall c \in Incl_i\ \exists S \in \pi : c \subseteq S$; **and**
- $\forall c \in \overline{Incl_i}\ \ \nexists S \in \pi : c \subseteq S$

that is, for every desirable pattern $c \in Incl_i$ there is a coalition $S \in \pi$ that contains c (i.e. $c \subseteq S$); while for each unwanted pattern $c \in \overline{Incl_i}$ there is no coalition $S \in \pi$ such that c is contained in S. In general, a formula γ_i is of the form: $\gamma_i = \langle Incl_{i,1}, \overline{Incl_{i,1}}\rangle \vee \langle Incl_{i,2}, \overline{Incl_{i,2}}\rangle \vee \cdots \vee \langle Incl_{i,p}, \overline{Incl_{i,p}}\rangle$, and a partition π satisfies γ_i (we write $\pi \models \gamma_i$) if there is a pair $\langle Incl_{i,j}, \overline{Incl_{i,j}}\rangle$ such that π satisfies $\langle Incl_{i,j}, \overline{Incl_{i,j}}\rangle$. Therefore, the partition space Π is the disjoint union of the sets: $P_i^+ = \{\pi \in \Pi : \pi \models \gamma_i\}$ and $P_i^- = \{\pi \in \Pi : \pi \not\models \gamma_i\}$.

Example 3. Consider a BHG of a set of 7 agents $N = \{ag_1, ag_2, ag_3, ag_4, ag_5, ag_6, ag_7\}$

in PFF settings, and partitions:

$$\pi_1 = \{\{ag_1, ag_2\}, \{ag_3, ag_4, ag_5\}, \{ag_6, ag_7\}\}$$
$$\pi_2 = \{\{ag_1, ag_3\}, \{ag_2, ag_4\}, \{ag_5, ag_6, ag_7\}\}$$

and ag_1's preferences as follows:

$$\gamma_1 = \left\langle \{\langle ag_3, ag_4\rangle, \langle ag_6, ag_7\rangle\}, \overline{\{\langle ag_5, ag_7\rangle\}} \right\rangle \vee \left\langle \{\langle ag_1, ag_2\rangle\}, \overline{\{\langle ag_5, ag_6\rangle\}} \right\rangle$$

We observe through γ_1 that ag_1 is satisfied with a partition that:

- contains a subcoalition with agents ag_3 & ag_4, a subcoalition with agents ag_6 & ag_7 **and** does not contain a subcoalition with agents ag_5 & ag_7 ($\gamma_{1,1}$); **or**
- contains a subcoa*lition with agents ag_1 & ag_2 **and** does not contain a subcoalition with agents ag_5 & ag_6 ($\gamma_{1,2}$)

Thus, we conclude that π_1 is satisfactory (via $\gamma_{1,1}$), and π_2 is not satisfactory, meaning that $\pi_1 \in P_1^+$ and $\pi_2 \in P_1^-$.

Henceforth, PFF-BHGs will refer to *boolean hedonic games in partition function form* and PFF-ASHGs to *additively separable hedonic games in partition function form.*

3.2 Learning agent's preferences

In this section we discuss how agents who are unaware of their own preferences can learn to predict their own utility function via supervised learning models, and evaluate them in a way that is fitting for our game settings.

3.2.1 Input encoding

In a hedonic game $\mathbf{G}$ in *PFF*, a given partition π_k is encoded to an input vector $\mathbf{x}_k$, where index k denotes a specific partition of our dataset. Our encoding uses $\binom{n}{2}$ input variables[3] (n is the number of agents), such that each variable contains information for a specific pair of agents $\langle ag_i, ag_j\rangle$, regarding their coexistence in the same coalition, and this information is depicted with boolean variables (integer values of 1 or 0). Input

[3]From (3.5), we can observe that the number of elements is $\sum_{i=1}^{i=n-1} i = \frac{n\cdot(n-1)}{2} = \frac{n!}{2!\cdot(n-2)!} = \binom{n}{2}$

size equals to the size of unordered pairs for all agents. Thus, a partition π_k is encoded to $\mathbf{x}_k$ as follows.

$$
\begin{aligned}
\mathbf{x}_k = [\, & \underbrace{\mathbb{1}_{ag_2 \in \pi(ag_1)}, \mathbb{1}_{ag_3 \in \pi(ag_1)}, \cdots, \mathbb{1}_{ag_n \in \pi(ag_1)},}_{n-1 \text{ elements}} \\
& \underbrace{\mathbb{1}_{ag_3 \in \pi(ag_2)}, \mathbb{1}_{ag_4 \in \pi(ag_2)}, \cdots, \mathbb{1}_{ag_n \in \pi(ag_2)},}_{n-2 \text{ elements}} \\
& \qquad \vdots \\
& \underbrace{\mathbb{1}_{ag_{i+1} \in \pi(ag_i)}, \mathbb{1}_{ag_{i+2} \in \pi(ag_i)}, \cdots, \mathbb{1}_{ag_n \in \pi(ag_i)},}_{n-i \text{ elements}} \\
& \qquad \vdots \\
& \underbrace{\mathbb{1}_{ag_{n-1} \in \pi(ag_{n-2})}, \mathbb{1}_{ag_n \in \pi(ag_{n-2})},}_{2 \text{ elements}} \underbrace{\mathbb{1}_{ag_n \in \pi(ag_{n-1})}}_{1 \text{ element}}]^T
\end{aligned} \tag{3.5}
$$

where $\mathbb{1}_{ag_j \in \pi(ag_i)}$, for the pair (ag_i, ag_j), contains the value of 1 if agent j is in i's coalition, otherwise the corresponding value is 0.

We can argue that $\forall \pi_i, \pi_j \in \Pi, i \neq j$, where Π is the set of all possible partitions, it holds true that their encoded inputs $\mathbf{x}_i \neq \mathbf{x}_j$ [4]. This property is essential, otherwise our models could potentially perceive two different partitions as the same one. This encoding of input $\mathbf{x}_k$ is utilized in all game scenarios.

Example 4. Consider a game with 5 agents $N = \{ag_1, ag_2, ag_3, ag_4, ag_5\}$ and a partition π_k of this set of agents $\pi_k = \{\{ag_1, ag_3\}, \{ag_2, ag_4, ag_5\}\}$. From Equation (3.5), the obtained $\mathbf{x}_k$ will be

[4]From an input vector x_k, the corresponding partition π_k can be obtained via a function that creates all correlations between agents, as specified in x_k, thus generating the initial π_k. Therefore, two different partitions cannot have the same input vector.

$$\begin{aligned}
\mathbf{x}_k = [\ & \underbrace{\mathbb{1}_{ag_2\in\pi(ag_1)}, \mathbb{1}_{ag_3\in\pi(ag_1)}, \mathbb{1}_{ag_4\in\pi(ag_1)}, \mathbb{1}_{ag_5\in\pi(ag_1)},}_{\text{4 elements}} \\
& \underbrace{\mathbb{1}_{ag_3\in\pi(ag_2)}, \mathbb{1}_{ag_4\in\pi(ag_2)}, \mathbb{1}_{ag_5\in\pi(ag_2)},}_{\text{3 elements}} \\
& \underbrace{\mathbb{1}_{ag_4\in\pi(ag_3)}, \mathbb{1}_{ag_5\in\pi(ag_3)},}_{\text{2 elements}} \underbrace{\mathbb{1}_{ag_5\in\pi(ag_4)}}_{\text{1element}}]^T \\
\mathbf{x}_k = [\ & 0 \ , \ 1 \ , \ 0 \ , \ 0 \ , \\
& 0 \ , \ 1 \ , \ 1 \ , \\
& 0 \ , \ 0 \ , \ 1 \ ]^T
\end{aligned}$$

Starting from a partition with singletons, $\pi_{k,o} = \{\{ag_1\}, \{ag_2\}, \{ag_3\}, \{ag_4\}, \{ag_5\}\}$, we examine x_k, i.e. we observe that $ag_3 \in \pi(ag_1)$, $ag_4 \in \pi(ag_2)$, $ag_5 \in \pi(ag_2)$ and $ag_5 \in \pi(ag_4)$. In order to satisfy all conditions, we merge ag_1's and ag_3's coalitions, then ag_4's and ag_2's coalitions and so forth. Finally, $\pi_{k,o} = \{\{ag_1, ag_3\}\{ag_2, ag_4, ag_5\}\} = \pi_k$ is obtained.

3.2.2 Output encoding - Regression Models

Regression Models essentially attempt to approximate a function and a labeled dataset is required for this task. In a setting where an agent ag_i attempts to acquire an approximation of its utility function via a labeled dataset $\mathrm{D} = \{\langle \mathrm{x}_1, t_1\rangle, \cdots, \langle \mathrm{x}_d, t_d\rangle\}$ of data $\mathbf{x}_k$ and labels t_k, we specify that $\mathbf{x}_k$ is the encoding for a given partition π_k and t_k captures the corresponding utility of π_k.

In PFF-ASHGs, a utility function is formally defined in 3.1.2, thus labels $t_k = \upsilon_i(S, \pi_k)$, where $ag_i \in S$ and $S \in \pi_k$. Since $S \in \pi_k$, we can simply denote the utility function as $\upsilon_i(\pi_k)$.

For agent ag_i in PFF-ASHG settings:

$$t_k = \upsilon_i(\pi_k) \tag{3.6}$$

In PFF-BHGs 3.1.3, a utility function is not formally defined, as a partition π_k belongs either in P_i^+ or in P_i^- (is satisfactory or not). Consequently, a utility value can express the payoff (satisfaction) for π_k. Considering that we will employ regression models for both game settings, we define labels for dichotomous preferences as:

$$t_k = \begin{cases} +c & \text{if } \pi_k \in P_i^+ \\ -c & \text{if } \pi_k \in P_i^- \end{cases} \tag{3.7}$$

where $c \in \mathbb{R}$ is a constant, and the regression model will map a predicted real value $\hat{t_k}$ to $+c$ if $\hat{t_k} > 0$ and to $-c$ otherwise. This mapping procedure will be applied for LR and LR-RBF.

For NNs, a sigmoid output activator is used, thus target values for $t_{k,NN}$ are:

$$t_{k,NN} = \begin{cases} 1 & \text{if } \pi_k \in P_i^+ \\ 0 & \text{if } \pi_k \in P_i^- \end{cases} \tag{3.8}$$

While from a given dataset D we obtain an approximation of the utility function $\hat{u_i}$, we implicitly extract an approximation of the *hidden* preference relation $\succsim_i^{\hat{u}_i}$ for any pair of partitions $\langle \pi_l, \pi_m \rangle$ in the partition set Π by simply comparing $\hat{u_i}(\pi_l)$ with $\hat{u_i}(\pi_m)$.

3.2.3 Evaluation of regression models

In order to evaluate our experimental results, it is essential to determine the nature of the desirable outcomes. We work with hedonic games, a class of cooperative games which, unlike others, possess some particular properties. Specifically, in such games, we care little about the actual coalitions' utilities, since our interest lies mainly on the preference relation formed. The modelling of the games studied within the scope of this thesis (i.e. the interpretation of a game instance into input data for a learning model), allows us to extract the desired preferences. Thus, a question arises: do we care to learn the best function $\hat{u_i}(\pi)$ that resembles the actual $u_i(\pi)$; or do we desire a $\hat{u_i}(\pi)$ that encodes a preference relation best matching the actual one?

The norm in evaluation of such supervised models is the *mean square error* (MSE) or *root mean square error* (RMSE) metrics, that depict the average squared distance of a model's predictions to the actual values.

$$\text{MSE}(u_i, \hat{u_i}) = \frac{1}{|D|} \sum_{\pi \in D} \Big(u_i(\pi) - \hat{u_i}(\pi) \Big)^2 \tag{3.9}$$

$$\text{RMSE}(u_i, \hat{u_i}) = \sqrt{\text{MSE}} \tag{3.10}$$

In our case, we care more about finding a function that will correctly predict pairwise relations between two partitions, and not the actual utility function. This resulted in considering the following metric.

Qualitative Proximity metric

When a utility function of an agent ag_i is "learned" based on training data, we implicitly extract a preference relation $\succsim_i^{\hat{u}_i}$, over partitions. Then we can measure the *percentage of equivalence* between u_i and $\hat{u}_i$ by counting the average of pairwise relations that are identically encoded by u_i and $\hat{u}_i$. Thus, we utilize the *Qualitative Proximity* (QP) (Georgara, Troullinos, and Chalkiadakis, 2019), which in fact could be thought of as a variant of *Kendall Tau* metric (Kendall, 1948), as follows:

$$\mathrm{QP}(u_i, \hat{u}_i) = \frac{\sum_{\pi,\pi' \in D} \mathrm{CHK}\Big(u_i, \hat{u}_i, \pi, \pi'\Big)}{|D|(|D|-1)/2} \tag{3.11}$$

$$\mathrm{CHK}\Big(u_i, \hat{u}_i, \pi, \pi'\Big) = \begin{cases} 1 & \text{if } \Big(\pi \succsim_i^{u_i} \pi' \ \wedge \ \pi \succsim_i^{\hat{u}_i} \pi'\Big) \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Essentially, QP performs all possible pairwise relations and calculates the proportion of preferences that our modelled function predicted correctly.

While Equations (3.11),(3.12) suggest that all possible pairwise relations are examined (D is our dataset), i.e. for $d = |D|$ we have $(d-1)^2$ combinations (there are $d-1$ pairwise combinations for each partition $\pi \in D$). If (3.12) also asserts whether $(\pi' \succsim_i^{u_i} \pi \ \wedge \ \pi' \succsim_i^{\hat{u}_i} \pi)$, then we only observe a pair $\langle \pi_l, \pi_m \rangle$ once and we automatically need to perform approximately half of these combination, specifically $\frac{d(d-1)}{2}$ (size of unordered pairs). Hence the denominator in (3.11).

Example 5. Consider 3 partitions $D = \{\pi_1, \pi_2, \pi_3\}$ and an agent ag_i with the following preference relation $\succsim_i^{u_i}$.

$$\pi_2 \succ_i \pi_1 \succ_i \pi_3$$

The extracted preference relation $\succsim_i^{\hat{u}_i}$ is as follows:

$$\pi_1 \succ_i \pi_2 \succ_i \pi_3$$

Listing all unordered pairs with the corresponding results gained from 3.12 (also checking for $(\pi' \succsim_i^{u_i} \pi \ \wedge \ \pi' \succsim_i^{\hat{u}_i} \pi)$)

Thus, $\mathrm{QP}(u_i, \hat{u}_i) = \frac{2}{3}$.

$\langle \pi_1, \pi_2 \rangle$	$\text{CHK}\Big(u_i, \hat{u}_i, \pi_1, \pi_2\Big) = 0$
$\langle \pi_1, \pi_3 \rangle$	$\text{CHK}\Big(u_i, \hat{u}_i, \pi_1, \pi_3\Big) = 1$
$\langle \pi_2, \pi_3 \rangle$	$\text{CHK}\Big(u_i, \hat{u}_i, \pi_2, \pi_3\Big) = 1$

Table 3.1: Distance of predicted and actual preferences

Parallelization of the QP metric

The Algorithm (2) for computing the QP metric for a set of predicted and true labels of partitions, is presented below.

Algorithm 2 Calculate the QP metric of predicted $\hat{\mathbf{t}}$ and true $\mathbf{t}$ labels for a set of partitions

```
 1: procedure CalculateQP($\hat{\mathbf{t}}, \mathbf{t}$)          ▷ predicted labels: $\hat{\mathbf{t}}$, true labels: $\mathbf{t}$
 2:     $valid \leftarrow 0$                  ▷ $valid$ counts the size of correct preference predictions
 3:     $labelsSize \leftarrow length(\mathbf{t})$
 4:     for $i$ in $[1, labelsSize - 1]$ do
 5:         for $j$ in $[i + 1, labelsSize]$ do
 6:             $isCorrect \leftarrow checkOrder(t\hat{[i]}, t\hat{[j]}, t[i], t[j])$
 7:             if $isCorrect$ then
 8:                 $valid \leftarrow valid + 1$
 9:     $QP \leftarrow valid/combinations(labelsSize)$
10:     return $QP$
```

As it can be observed, this algorithm has $O(k^2)$[5] time complexity, where k denotes the size of partitions. Since no dependencies exist on the computation for each unordered pair $\langle i, j \rangle$, this algorithm can be efficiently parallelized.

In order to avoid communication overhead, we choose to do coarse-grained parallelism (split the algorithm into large tasks). This approach raises the prospect of load imbalance, meaning that not all tasks have the same load, thus resulting in poor utilization of available computational resources. Merely dividing indices of the outer for-loop [6] results in large load imbalance on different processes, as the computation of the inner for-loop depends on the value of i. For larger i, inner for-loop has less computations, due to the smaller range of j (j always starts from $(i + 1)$ until the constant size of labels)

Algorithm 3 contains the task of each process, and Algorithm 4 is the parallel version of the initial QP computation in Algorithm 2.

[5]size of all unordered pairs is $\frac{k(k-1)}{2} = 0.5k^2 + 0.5k$

[6]E.g. for 20 partitions and 2 processes, process 0 computes indices $[0, 9]$ and process 1 computes indices $[10, 19]$

Aiming to overcome the load imbalance, *createIndices* function in Algorithm 4 attempts to evenly distribute the computations across processes, as demonstrated in Table 3.2.

Algorithm 3 Process to calculate the QP metric of predicted $\hat{\mathbf{t}}$ and true $\mathbf{t}$ labels for a specified subset of indices

```
1: procedure ProcessCalculateQP($\hat{\mathbf{t}}, \mathbf{t}$,indices,$valid$)    ▷ predicted labels: $\hat{\mathbf{t}}$, true labels: $\mathbf{t}$, subset of indices
2:     $labelsSize \leftarrow length(\mathbf{t})$
3:     for $i$ in indices do
4:         for $j$ in $[i+1, labelsSize]$ do
5:             $isCorrect \leftarrow checkOrder(\hat{t[i]}, \hat{t[j]}, t[i], t[j])$
6:             if $isCorrect$ then
7:                 $valid \leftarrow valid + 1$
```

Algorithm 4 Parallel computation of QP metric of predicted $\hat{\mathbf{t}}$ and true $\mathbf{t}$ labels

```
1: procedure ParallelQP($\hat{\mathbf{t}}, \mathbf{t}$,numOfProcesses)    ▷ predicted labels: $\hat{\mathbf{t}}$, true labels: $\mathbf{t}$, number of processes: numOfProcesses
2:     $labelsSize \leftarrow length(\mathbf{t})$
3:     $indicesArray \leftarrow createIndices(labelsSize, \text{numOfProcesses})$
4:     for $p$ in numOfProcesses do
5:         $process[p] \leftarrow NewProcess(ProcessCalculateQP, \hat{\mathbf{t}}, \mathbf{t}, indicesArray[p], valid[p])$
6:         $process[p].startProcess()$
7:     for $p$ in numOfProcesses do
8:         $process[p].join()$
9:     $QP \leftarrow sum(valid)/combinations(labelsSize)$
10:    return $QP$
```

Example 6. Consider $K = 19$ partitions and $N = 3$ processes, and the corresponding distribution Table 3.3. Process 0 will compute indices $indices_0 = \{1, 6, 7, 12, 13, 18\}$. Summing all computations required for these indices, we obtain:

$$\begin{aligned} computations_0 &= \sum_{1+1}^{19} 1 + \sum_{6+1}^{19} 1 + \sum_{7+1}^{19} 1 + \cdots + \sum_{18+1}^{19} 1 \\ &= 18 + 13 + 12 + 7 + 6 + 1 = 57 \end{aligned}$$

Process 1 will compute $indices_1 = \{2, 5, 8, 11, 14, 17\}$ with $computations_1 = 17 + 14 + 11 + 8 + 5 + 2 = 57$, and process 2, $indices_2 = \{3, 4, 9, 10, 15, 16\}$ with $computations_2 = 16 + 15 + 10 + 9 + 4 + 3 = 57$. Load balance across all processes is apparent.

Process 0	Process 1	$\cdots$	Process $N-2$	Process $N-1$
$K-1$	$K-2$	$\cdots$	$K-(N-1)$	$K-N$
$K-2N$	$K-(2N-1)$	$\cdots$	$K-(N+2)$	$K-(N+1)$
$K-(2N+1)$	$K-(2N+2)$	$\cdots$	$K-(3N-1)$	$K-(3N)$
.	.	...	.	.
.	.	...	.	.
.	.	...	.	.

Table 3.2: Distribution of indices in *createIndices* function for K partitions and N processes

Process 0	Process 1	Process 2
18	17	16
13	14	15
12	11	10
7	8	9
6	5	4
1	2	3

Table 3.3: Distribution of indices in *createIndices* function for 19 partitions and 3 processes

3.2.4 Hyper-Parameter Optimization with the QP metric

NNs are applied to different game scenarios, namely PFF-ASHGs and PFF-BHGs, both with ranging number of agents and complexity of hidden preferences, so manual fine tuning of hyper-parameters (number of hidden layers, size of each layer), or exhaustive search algorithms create a significant bottleneck on speed performance. Merely selecting a big network will overfit in cases where the underlying function is not rather complex (and will take significantly more time to train). Moreover, an adaptable model that can adjust its parameters to learn both simple and complicated functions is apparently quite useful and desirable.

As discussed in 2.4.1, the TPE hyper-parameter optimization algorithm is fitting for a NN with a dynamic size of hidden layers. This algorithm attempts iteratively to minimize a loss function f_{loss}. In 3.2.3, we indicate that the QP metric best fits the problem at hand. Consequently, QP best determines the loss of an approximated function (or a preference relation). Thus, the selected loss function is:

$$f_{\text{loss}} = \frac{1}{\text{QP}} \tag{3.13}$$

since TPE seeks for the parameters that minimize this loss function, while QP expresses the ratio of correct preference relations between the true and approximated function.

However, knowledge of the underlying function is absent, and agents compare true labels from labeled data with predicted labels, in order to calculate the QP. Performing an evaluation on the same data creates bias in the choices of TPE, as it evaluates on the training set. Therefore, we perform *k-fold cross validation* (Refaeilzadeh, Tang, and Liu, 2009) on our training dataset, that is, for a training dataset of size d, and a choice of k, training data are split to training and testing k times ($(k-d)/k$ for training , and d/k for testing). In each iteration a different portion of data is selected for testing. After evaluation (calculation of QP$_{kfold}$) for all splits, we obtain an average QP$_{avg,kfold}$ that is used for the TPE loss function.

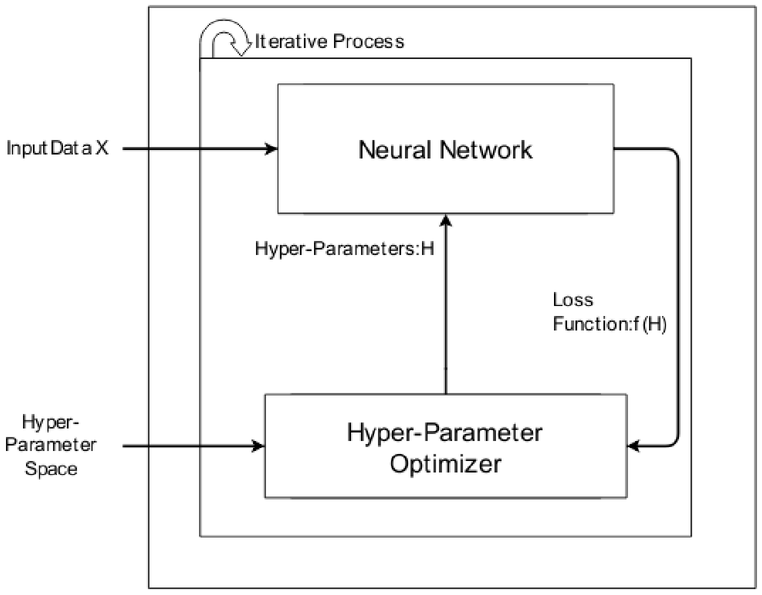

Figure 3.2: Visual representation of hyper-parameter optimization

3.3 Generating Satisfactory Partitions via Gaussian Mixture Models

With the use of a generative model, an agent can create new partitions to offer, based on partitions observed in the past. Gaussian Mixture Models (GMMs) use unlabeled data to construct (mixtures of) distributions we can then sample in order to (subsequently, via Algorithm below) generate new data (i.e., partitions). Agents utilize GMMs as generative models using a subset of observed partitions as training data. As discussed in Section 3.2, partitions are converted to a 1-dimensional input array with binary values 1 & 0, that express the correlation of two agents, i.e. whether they are on the same coalition or not. Sampling from GMMs, we obtain an array that contains real values, and thus cannot be converted to a partition. A function is created for this task, that converts a generated sample to a partition.

As stated before, GMMs can also be used as generative models. After training, a number k of Gaussian distributions are generated by unlabeled training data. A random sample $\mathbf{S}$ can be drawn from this mixture of distributions, hence generating a new value $\mathbf{x}$. As discussed in Subsection 3.2.1, a partition is converted to a vector containing binary values that express whether two agents are in the same coalition. GMMs view training data as real values, thus the obtained vector $\mathbf{S}$ that is generated, contains real values. Simply rounding values to one or zero is not suitable as it may generate conflicts. For example, for 3 agents $N = \{ag_1, ag_2, ag_3\}$, where a generated sample S's values are rounded, if $\mathbf{S}_{rounded}$ specifies that $ag_1 \in \pi(ag_2)$ and $ag_2 \in \pi(ag_3)$, but $ag_1 \notin \pi(ag_3)$, then we have an obvious conflict, as ag_l and ag_3 must be in the same coalition if the first two expressions hold true.

We propose Algorithm 5 that takes $\mathbf{S}$ as input, and creates a partition. $\mathbf{S}$'s values are in the range $[0, 1]$. The key idea here is that each value expresses how likely it is that agent i needs agent j and k to be in the same coalition or not. Values close to 1 indicate that agent i needs the corresponding agents in the same coalition, while values close to 0 show the opposite. Thus, the first step is to order correlation values in descending order and split them between correlations that agent i "desires" and correlations that agent i does not[7]. The second step is to start from the partition that contains only singletons, and then modify this partition according to the $\mathbf{S}$ array. Iterating the *desirable* correlations in descending order, for a *desirable* correlation $\langle ag_j, ag_k\rangle$, the agent attempts to merge two coalitions C_1 and C_2, where $ag_j \in C_1$, $ag_k \in C_2$. This merging will be realised only if no pair of agents $\langle ag_l, ag_m\rangle$, where $ag_l \in C_1, ag_m \in C_2$ is in the *undesirable* list. This ensures that there will be no subcoalitions of pairs that agent i does not want, according to $\mathbf{S}$ vector. At the end of this process, a partition π_g is generated from $\mathbf{S}$.

Example 7. Consider a PFF-HG with 4 agents $\{ag_1, ag_2, ag_3, ag_4\}$. Input samples are $\mathbf{x} \in \mathbb{R}^6$, as size of input for 4 agents is $3 + 2 + 1 = 6$. From Equation (3.5), we can observe that every sample $\mathbf{x}_k$ has information about the correlation between two agents in partition π_k. For 4 agents, we know from Subsection 3.2.1 that the input will take the form:

$$\mathbf{x}_k = [\langle ag_1, ag_2\rangle, \langle ag_1, ag_3\rangle, \langle ag_1, ag_4\rangle, \langle ag_2, ag_3\rangle, \langle ag_2, ag_4\rangle, \langle ag_3, ag_4\rangle]^T \tag{3.14}$$

For example, if $\mathbf{x}_k = [1, 0, 0, 1, 0, 1, 0]^T$, then the corresponding correlations of

[7]If input value is $\mathbf{S}_l > 0.5$, then the correlation of agents is considered desirable, else undesirable

Algorithm 5 Convert a generated sample $\mathbf{S}$ to a partition π_g

```
1: procedure CONVERTSAMPLETOPARTITION($\mathbf{S}, agent, N$)    ▷ $N$ = number of agents
2:     $desirable, undesirable \leftarrow sampleToPriorityList(\mathbf{S}, agent, N)$    ▷ $desirable, undesirable$ contain pairs $< ag_j, ag_k >$ of agents
3:     $\pi_g \leftarrow \pi_{singletons}$    ▷ $\pi_{singletons} = \Big\{\{ag_1\}, \{ag_2\}, ..., \{ag_N\}\Big\}$
4:     for $\{ag_a, ag_b\}$ in $desirable$ do
5:         $coalition_a \leftarrow findAgentCoalition(ag_a, \pi_g)$
6:         $coalition_b \leftarrow findAgentCoalition(ag_b, \pi_g)$
7:         if $canBeMerged(coalition_a, coalition_b, undesirable)$ then
8:             $\pi_g \leftarrow mergeCoalitions(coalition_a, coalition_b, \pi_g)$
9:     return $\pi_g$
```

agents that exist in π_k are $\langle ag_1, ag_2\rangle, \langle ag_1, ag_4\rangle, \langle ag_2, ag_4\rangle$. So, the encoded partition had the form $\pi_k = \{\{ag_1, ag_2, ag_4\}, \{ag_3\}\}$.

After training a GMM with a number of samples $\mathbf{x}_k$, for agent i, we can sample from the model, thus obtaining a generated sample S^i_{gen}:

$$S^i_{gen} = [0.98, \quad 0.81, \quad 0.42, \quad 0.04, \quad 0.51, \quad 0.68]^T \tag{3.15}$$

We now show step by step how our algorithm works. From S^i_{gen}, two lists are created, a *desirable* (list of correlations with values $S^i_{gen,l} > 0.5$, in descending order) and *undesirable* (list of correlations with values $S^i_{gen,} \leq 0.5$, without specified ordering) list, with the values:

$$desirable = [\langle ag_1, ag_2\rangle, \langle ag_1, ag_3\rangle, \langle ag_3, ag_4\rangle, \langle ag_2, ag_4\rangle]^T \tag{3.16}$$

and

$$undesirable = [\langle ag_1, ag_4\rangle, \langle ag_2, ag_3\rangle]^T \tag{3.17}$$

We always start with a singleton partition $\pi_g = \{\{ag_1\}, \{ag_2\}, \{ag_3\}, \{ag_4\}\}$, and then iterate sequentially for all sets in *desirable* list.

- For $\langle ag_1, ag_2\rangle$, no conflicts will occur by merging coalitions that contain ag_1 and ag_2, so, new partition is now $\pi_g = \Big\{\{ag_1, ag_2\}, \{ag_3\}, \{ag_4\}\Big\}$.
- For $\langle ag_1, ag_3\rangle$, if we merge coalitions containing ag_1 and ag_3, our new partition will contain coalition $\{ag_1, ag_2, ag_3\}$, but *undesirable* list contains the subcoalition $\langle ag_2, ag_3\rangle$, thus creating a conflict. So, partition π_g will remain as is in this iteration.

- For $\langle ag_3, ag_4 \rangle$, if we merge coalitions containing ag_3 and ag_4, our new partition will contain coalition $\{ag_3, ag_4\}$, thus creating no conflict. So, partition π_g will take the form: $\pi_g = \Big\{\{ag_1, ag_2\}, \{ag_3, ag_4\}\Big\}$.
- For $\langle ag_2, ag_4 \rangle$, if we merge the corresponding coalition, new partition will be the grand coaliition, that creates conflicts for both sets of *undesirable*, thus not updating π_g.

Finally, the new partition obtained from this process is:

$$\pi_g = \Big\{\{ag_1, ag_2\}, \{ag_3, ag_4\}\Big\} \tag{3.18}$$

Note that the order of sets in *desirable* will affect π_g, as different conflicts will occur from merging the same coalitions in a different order.

3.3.1 Data Selection

For PFF-BHGs, observed partitions that belong to P_i^+ are selected as training data. For PFF-ASHGs, preferences are not dichotomous, and a utility threshold r_t needs to exist that defines “good” partitions. This threshold is a real value in the range $[0, 1]$. For a given partition’s utility u_k, the following inequality needs to be true in order to add it to training data:

$$r_t <= \frac{u_k - u_{min}}{u_{max} - u_{min}} \tag{3.19}$$

where:

$$u_{min} = \min_{\forall u_k \in U_d}(u_k), u_{max} = \max_{\forall u_k \in U_d}(u_k) \tag{3.20}$$

U_d is the set of all utilities from observed partitions.

This can potentially result in too few samples for training. A greater number of samples that contain a small subset of partition below r_t is better, when few labeled data are available. As a result, a minimum number of samples is an essential requirement, even if for some partition this inequality does not hold true.

3.3.2 Number of Components

GMMs have as a parameter the number of components, i.e. the number of Gaussian Distributions (or the number of clusters). For a small number of components, generated partitions are very likely to be unobserved, and their true utility is not likely to be in the desired range (primarily in PFF-BHGs); while for a very large number of components, the opposite is true. However, in the case that we gravitate towards exploitation in a protocol by which someone makes proposals, generating already observed partitions is not a drawback.

Three different metrics are examined in order to determine the number of components.

- Akaike information criterion (AIC) (Akaike, 1974):

$$\text{AIC} = -2(\text{loglikelihood}) + 2K \tag{3.21}$$

  where K is the number of input parameters
- Bayesian information criterion (BIC) (Schwarz et al., 1978):

$$\text{BIC} = -2(\text{loglikelihood}) + \ln(n) \cdot K \tag{3.22}$$

  where K is the number of input parameters and n the number of samples
- Silhouette (SIL) (Rousseeuw, 1987):

$$SIL = \frac{1}{M} \sum_{i=0}^{M-1} \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.23}$$

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i,j) \tag{3.24}$$

$$b(i) = min_{i \neq j} \frac{1}{|C_i|} \sum_{j \in C_i} d(i,j) \tag{3.25}$$

  where $a(i)$ is the average distance of a data point $i \in C_i$, from all other data points in cluster C_i. $b(i)$ is the minimum average distance of a data point $i \in C_i$ from other clusters $C_j, j \neq i$.

For a given metric value $m_{GMM}(c)$, where c is the number of components, we can benefit from the TPE algorithm, and avoid an exhaustive search for finding the appropriate c, by using the loss function $f_{loss} = \frac{1}{m_{GMM}(c)}$.

Ideally, for an agent i, its GMM model will generate partitions that satisfy the following two properties.

- ag_i's utility for its generated partitions π_g is in the desired values range (as discussed in 3.3.1); and
- π_g are unobserved, i.e. ag_i has not yet observed the generated partition.

The first property is self-evident as agents want to offer partitions that satisfy themselves. Second property exists considering that we want to use GMMs as generative models. If obtained partitions are already observed, then GMMs are unnecessary, as a simple random selection of observed partitions would be adequate and equivalent to our method.

3.4 Coalition Formation in Hedonic Games with Externalities

In this section we discuss how multiple agents with conflicting preferences can collaborate by employing the supervised & unsupervised methods discussed above. Two different coalition formation approaches are discussed, with the purpose of discovering partitions that maximize the *social welfare*. The motivation behind our approaches is to construct a general protocol that can be applied to any game settings, which can be modelled as one of the two considered classes of hedonic games with *externalities*. Thus, our formation protocols are generic, as they are not tied to a specific application domain. Literature on formation protocols in hedonic games does not address games with externalities, as existing approaches do not consider agents that are interested in the structure of all coalitions, as seen in (Taywade, Goldsmith, and Harrison, 2019) and (Janovsky and Deloach, 2016) among others.

Since agents are affected by all coalitions on a partition, we consider it essential that agents should affect the structure of all coalitions. This led us to approach this problem as a negotiation game, where agents create offers (partitions of all participating agents) and vote on them. Offers can be generated via GMMs as in Section 3.3 and agents can vote on partitions based on their preference profile, that can be obtained by a regression model (Section 3.2).

3.4.1 Coalition Formation Protocol with Single Deviations

The aforementioned approach regarding an agent's ability to learn a function that approximates the true utility of any partition, and the use of GMMs to create partitions is utilized in the following coalition formation protocol (CFP-SD). This protocol is created with the purpose of combining each agent's hidden preferences and form coalitions that maximize the *social welfare* (SW), where SW for a given partition π_k is the sum of utilities of all agents for π_k.

$$\mathrm{SW}(\pi_k) = \sum_{i=0}^{N-1} \upsilon_i(\pi_k) \tag{3.26}$$

The SW of a partition π_k in *Boolean Hedonic Games* reflects the number of agents who consider their coalition satisfactory ($S_i \in N_i^+$), that is $\mathrm{SW} = |\{i \in N : \pi_k(i) \in N_i^+\}|$[8] (Aziz, Harrenstein, et al., 2016). The same philosophy can be extended to PFF-BHGs, considering partition preferences instead of coalitional preferences, i.e. $\mathrm{SW} = |\{i \in N : \pi_k \in P_i^+\}|$. Hence the reason we choose the utility function as defined in 3.27 for PFF-BHGs. In PFF-ASHGs, we could choose the utility function as defined in 3.1.2, but agents do not have the same utility range. Thus, a sum of these utilities would not be equally representative of all agents, as agents with wider values' ranges would influence SW more than agents with narrower ranges. Therefore, we define a normalized utility function for PFF-ASHGs in order to overcome this effect.

$$\upsilon_{i,\mathrm{BHG}}(\pi_k) = \begin{cases} 1 & \text{if } \pi_k \in P_i^+ \\ 0 & \text{if } \pi_k \in P_i^- \end{cases} \tag{3.27}$$

$$\upsilon_{i,\mathrm{ASHG}}(\pi_k) = \frac{\upsilon_i(\pi_k) - \min_{\mathrm{obs}}(\upsilon_i)}{\max_{\mathrm{obs}}(\upsilon_i) - \min_{\mathrm{obs}}(\upsilon_i)} \tag{3.28}$$

where $\min_{\mathrm{obs}}(\upsilon_i), \max_{\mathrm{obs}}(\upsilon_i)$ are the minimum and maximum utility value that agent i has observed.

As discussed, our game settings include agents with conflicting preferences, thus the same set of coalitions will satisfy each agent differently. In this protocol, all agents are negotiating in each iteration and are arranged in a circular layout, such that in iteration t, agent $ag_{t(\mathrm{mod} n)}$ (n:number of agents) starts negotiation by offering a partition π. Then, the next agent ($ag_{t+1(\mathrm{mod} n)}$) modifies π_{offer} to π^*_{offer}, and negotiating partition π_{offer} is updated to π^*_{offer} only if:

$$\sum_{i=0}^{n-1} \mathrm{choice}(ag_i, \pi^*_{\mathrm{offer}}, \pi_{\mathrm{offer}}) > 0 \tag{3.29}$$

where:

[8] $\pi_k(i) = S_i$: coalition of π_k that contains agent i

$$\text{choice}(ag_i, \pi^*_{\text{offer}}, \pi_{\text{offer}}) = \begin{cases} +1 & \text{if } \pi^*_{\text{offer}} \succ_i \pi_{\text{offer}} \\ -1 & \text{if } \pi_{\text{offer}} \succ_i \pi^*_{\text{offer}} \\ 0 & \text{if } \pi^*_{\text{offer}} \sim_i \pi_{\text{offer}} \end{cases} \tag{3.30}$$

This procedure is performed for each agent, until agent $ag_{t+n-1(\text{mod}n)}$. Thus, each agent can potentially change the resulting partition according to its preferences.

The initial partition π_{offer} in every iteration is derived by the GMM of the corresponding agent. Then, agents form counter-proposals π^*_{offer} based on π_{offer}, aiming to improve their own satisfaction by modifying the structure of π to an extend. Essentially, each agent takes π_{offer} as an input, examines all possible single deviations [9], and selects the partition that best satisfies its preferences according to its regression model, i.e. it chooses the partition with the highest utility.

Example 8. Consider 4 agents and a partition $\pi = \{\{ag_1, ag_4\}, \{ag_2\}, \{ag_3\}\}$. All possible single deviations are

- For agent ag_1

$$\{\{ag_1\}, \{ag_4\}, \{ag_2\}, \{ag_3\}\}$$
$$\{\{ag_4\}, \{ag_1, ag_2\}, \{ag_3\}\}$$
$$\{\{ag_4\}, \{ag_2\}, \{ag_1, ag_3\}\}$$

- For agent ag_2

$$\{\{ag_1, ag_2, ag_4\}, \{ag_3\}\}$$
$$\{\{ag_1, ag_4\}, \{ag_2, ag_3\}\}$$

- For agent ag_3

$$\{\{ag_1, ag_3, ag_4\}, \{ag_2\}\}$$
$$\{\{ag_1, ag_4\}, \{ag_2, ag_3\}\}$$

- For agent ag_4

$$\{\{ag_1\}, \{ag_2\}, \{ag_3\}, \{ag_4\}\}$$
$$\{\{ag_1\}, \{ag_2, ag_4\}, \{ag_3\}\}$$
$$\{\{ag_1\}, \{ag_2\}, \{ag_3, ag_4\}\}$$

Note that agents examine single deviations not only for themselves, as we focus on games with externalities.

[9]For N agents, all possible single deviations have Worst-case time complexity of $O(N^2)$ when partition contains only singletons, i.e. $\pi = \{\{ag_1\}, \{ag_2\}, \cdots, \{ag_N\}\}$ and Best-case time complexity of $\Omega(N)$ when partition is the grand coalition $\pi = \{\{ag_1, ag_2, \cdots, ag_N\}\}$.

By performing a modest modification, it is less likely to affect a substantial number of agents. Using the criterion for update from Equation (3.29), we know that the majority of agents prefer π^*_{offer} to π_{offer}. So, when an update is performed, we move towards partitions that satisfy more agents. In this sense, we attempt to greedily improve the SW of the initial partition π_{offer} without a centralized entity that has information about the utility of all agents. This approach is demonstrated in the following example.

Example 9. Consider 4 agents $N = \{ag_1, ag_2, ag_3, ag_4\}$ in the above-mentioned protocol. At the first iteration we start from agent ag_1, who proposes a partition $\pi_{\text{offer},1}$ from its own GMM model, which is the current candidate, thus $\pi_{\text{offer}} = \pi_{\text{offer},1}$.

- Then ag_2 offers a counter proposal with partition $\pi^*_{\text{offer},2}$, and all agents have to choose between π_{offer} and $\pi^*_{\text{offer},2}$. Let:

$$\sum_{i=0}^{n-1} \mathit{choice}(ag_i, \pi^*_{\text{offer},2}, \pi_{\text{offer}}) = 1$$

  Now the proposed partition π_{offer} is updated to $\pi^*_{\text{offer},2}$.
- Agent ag_3 now proposes partition $\pi^*_{\text{offer},3}$, but:

$$\sum_{i=0}^{n-1} \mathit{choice}(ag_i, \pi^*_{\text{offer},3}, \pi_{\text{offer}}) = -2$$

  thus resulting in no change in π_{offer}.
- Finally, ag_4 offers partition $\pi^*_{\text{offer},4}$, and

$$\sum_{i=0}^{n-1} \mathit{choice}(ag_i, \pi^*_{\text{offer},4}, \pi_{\text{offer}}) = 3$$

  thus updating π_{offer} to $\pi^*_{\text{offer},4}$.

The first iteration is complete and partition π_{offer} is formed. In the next iteration, the same process is performed, but now ag_2 proposes the first offer.

3.4.2 Coalition Formation Protocol with Copeland's Method

We now view formation under a social choice theory (Brandt et al., 2016) perspective, and propose an alternative coalition formation protocol that makes use of the *Copeland's Method* (G. Saari and Merlin, 1996) on offering partitions (CFP-CM). This method allows agents with pairwise preferences to select a partition that satisfies most agents.

Condorcet Criterion

In environments with multiple agents and multiple choices (more than two), the *Condorcet Winner* is the choice that defeats all other candidate choices in pairwise elections, by gaining a majority of votes from agents. While a *Condorcet Winner* does not always exist, *Copeland's Method* serves as a voting rule that satisfies the *Condorcet Criterion*; that is, when a *Condorcet Winner* exists, it will be the prevailing choice of *Copeland's Method* (Procaccia, 2008).

Copeland's Method

The *Copeland's Method* voting rule chooses the candidate who beats the highest number of other candidates in pairwise elections (Procaccia, 2008). In this method, each pair of candidates is compared, and all agents vote to determine which of the two is more preferred. The winner of each pair earns 1 point. In case of a tie, both candidates earn half a point. Finally, when all pairwise comparisons are performed, the winner candidate is the one with the highest number of points (Lippman and College, 2013).

Example 10. Consider Table 3.4, with 3 choices $\{a, b, c\}$ and 4 players $\{1, 2, 3, 4\}$.

Players:	1	2	3	4
1st choice	b	a	b	c
2nd choice	c	b	c	a
3rd choice	a	c	a	b

Table 3.4: Ranking of choices for 4 players

Comparing all pairs of choices $\{a, b, c\}$:

- $\langle a, b\rangle$: 2 out of 4 players prefer choice a over b, so both a and b earn half a point.
- $\langle a, c\rangle$: 3 out of 4 players prefer c over a, so c earns a point.
- $\langle b, c\rangle$: 3 out of 4 players vote for choice b over c, resulting in 1 point for b.

Overall, choice a received 0.5 points, choice b received 1.5 points and choice c received 1 point. Thus, choice b is the winner, based on this voting rule.

According to *Copeland's Method*, scores are assigned to partitions as follows. In a pairwise relation between π_m and π_n, score s_m^n is assigned for π_m, where:

$$s_m^n = \begin{cases} 1 & \text{if } \pi_m \succ \pi_n \\ \frac{1}{2} & \text{if } \pi_m \sim \pi_n \\ 0 & \text{if } \pi_n \succ \pi_m \end{cases} \tag{3.31}$$

Example 11. Consider an example of 5 agents $N = \{ag_1, ag_2, ag_3, ag_4, ag_5\}$, a set of 4 partitions $\Pi_{CM} = \{\pi_1, \pi_2, \pi_3, \pi_4\}$ and the following preference profiles.

Agent	Preference Profile
ag_1	$\pi_1 \succ_1 \pi_3 \sim_1 \pi_2 \succ_1 \pi_4$
ag_2	$\pi_2 \succ_2 \pi_3 \succ_2 \pi_4 \succ_2 \pi_1$
ag_3	$\pi_1 \succ_3 \pi_4 \succ_3 \pi_2 \succ_3 \pi_3$
ag_4	$\pi_2 \succ_4 \pi_1 \succ_4 \pi_3 \succ_4 \pi_4$
ag_5	$\pi_3 \succ_5 \pi_2 \succ_5 \pi_1 \succ_5 \pi_4$

Below is a list of all pairwise comparisons and their scores based on each agent's preferences.

Pairwise Comparisons	Score	Winner
$\pi_1 - \pi_2$	$2 - 3$	π_2
$\pi_1 - \pi_3$	$3 - 2$	π_1
$\pi_1 - \pi_4$	$4 - 1$	π_1
$\pi_2 - \pi_3$	$2.5 - 2.5$	tie
$\pi_2 - \pi_4$	$4 - 1$	π_2
$\pi_3 - \pi_4$	$4 - 1$	π_3

The final score of each partition is: $\pi_1 = 2$ points, $\pi_2 = 2.5$ points, $\pi_3 = 1.5$ point and $\pi_4 = 0$ points. Thus, $\pi_{CM} = \pi_2$ is the chosen partition.

With *Copeland's Method*, our agents are now able to select the best partition from a set of partitions Π_{CM}. This set will contain generated partitions from GMMs. Each agent is requested to offer a number of partitions q, in order to create the set Π_{CM} of $N \cdot q$ offered partitions from all agents. Then, agents vote on all partitions of this set, except their own, and the best partition is the one π_{CM} with the best score, as demonstrated in Example 11. Finally, agents collaborate and form the coalitions of π_{CM}. This process is repeated for a specified number of iterations.

3.4.3 Coalition Formation with hidden preferences

Agents that participate in both protocols need to collect labeled data in order to employ a regression model that extracts their utility function, since they are unaware of their preferences. Thus, agents offer and select partitions randomly when they are uncertain. This randomness serves the learning process, as it provides a more distributed representation of data. Each agent i learns from data, and considers itself *aware* of its preferences when it reaches a confidence level, that is when QP exceeds a threshold $\mathrm{QP}_{threshold}$.

After training a regression model, each agent initially continues to act randomly, but also performs predictions with its approximated function. When it retrains its model on new data, it also calculates the QP between predicted and observed labels. Once QP exceeds this threshold (evaluated on the last $2 \cdot \mathrm{Batch}$ (see Table 4.6) partitions observed), then this agent participates by generating partitions from a GMM model when required by the protocol, and also by evaluating and voting on partitions via its approximated preference relation $\succsim_i^{\hat{u}_i}$. Thus, our agents initially act randomly in both formation protocols, until they are certain about their approximated utility function.

Chapter 4

Experimental Evaluation

In this chapter we conduct systematic evaluation of our approach, analyze and compare the performance of our methods.

LR models, k-means and GMM implementations were derived from the *scikit-learn* library (Pedregosa et al., 2011), while NNs and their optimization algorithms from the *tensorflow* library (Martín Abadi et al., 2015). Hyper-parameter optimization for our models was achieved through the *hyperopt* library (J. Bergstra, Yamins, and Cox, 2013). In Table 4.1 we provide a table with our parameter choices and ranges across all experiments.

n	Graph			Formulae	
	κ, λ, ξ	$\|edges\|$	$edge_{weight}$	$\#\langle Incl_i, \overline{Incl_i}\rangle / agent$	$\#agents / \langle Incl_i, \overline{Incl_i}\rangle$
5	$\sim \mathcal{U}(0, 10)$	$\sim \mathcal{U}(4, 10)$	$\mathcal{U}(1, 5)$	$\mathcal{U}(2, 3)$	$\mathcal{U}(1, 4)$
10	$\sim \mathcal{U}(0, 10)$	$\sim \mathcal{U}(9, 45)$	$\mathcal{U}(1, 5)$	$\mathcal{U}(4, 7)$	$\mathcal{U}(4, 6)$
20	$\sim \mathcal{U}(0, 10)$	$\sim \mathcal{U}(19, 190)$	$\mathcal{U}(1, 5)$	$\mathcal{U}(9, 11)$	$\mathcal{U}(5, 7)$
50	$\sim \mathcal{U}(0, 10)$	$\sim \mathcal{U}(49, 1225)$	$\mathcal{U}(1, 5)$	$\mathcal{U}(12, 15)$	$\mathcal{U}(8, 12)$

Table 4.1: Game environment parameters

Randomness of Generated Game Parameters

Random partitions are generated by the following algorithm 6. Starting from an empty partition **P** and a set of available agents **N**, coalitions **S** are formed until all agents belong in the partition scheme. The process initiates by selecting randomly the size s of **S**, and then sampling randomly s agents from the available agents, thus creating **S**. Then, the available agents' set is updated, and new coalition is now part of the generated partition **P**. This procedure is repeated until the set of available agents is empty.

Algorithm 6 Generate a random partition **P** from a set of agents **N**

```
1: procedure GENERATERANDOMPARTITION($\mathbf{N}$)    ▷ $\mathbf{N}$ = set of agents
2:     $\mathbf{P} \leftarrow \varnothing$    ▷ start from an empty partition
3:     remainingN $\leftarrow \mathbf{N}$
4:     while $length$(remainingN) $> 0$ do    ▷ in each iteration, a new coalition is formed
5:         $s \leftarrow randomInteger(1, length(\text{remainingN}))$    ▷ size of new coalition
6:         $\mathbf{S} \leftarrow randomSample(\text{remainingN}, s)$    ▷ sample $s$ agents from remaining set
7:         remainingN $\leftarrow$ remainingN $\setminus \mathbf{S}$    ▷ update remaining Agents' set
8:         $\mathbf{P} \leftarrow \mathbf{P} \cup \{\mathbf{S}\}$    ▷ append formed coalition $\mathbf{S}$ to the new partition
9:     return $\mathbf{P}$
```

#Partitions	Average execution time of QP	
	Single Process	2 Processes
10000 Samples	22.8 sec	11.9 sec
20000 Samples	93.4 sec	56.2 sec

Table 4.2: Execution time of QP metric

Parallelization of QP

In table 4.2 average computation time is provided for the QP, in order to demonstrate the effectiveness of parallelization.

4.1 Extracting a Preference Relation

In Figures 4.1, 4.2 average QP results are provided for different game scenarios. Table 4.5 contains the size of datasets and 4.4 the average execution time.

LR models do not contain any parameters to tune, but LR-RBF, benefiting from k-means in order to acquire the centers $\mathbf{m}_k$, have a parameter k that defines the size of clusters. This parameter is calculated using the TPE algorithm, in the range of $k = [5, 25]$. NNs use the Adam (discussed in Subsection 2.3.4) optimization algorithm, as Adam was created as an improved version of AdaGrad and RMSProp by combining both, and is adopted more extensively. For all settings, the number of layers is 1 or 2, while the nodes per layer are in $[\frac{n}{2}, \frac{n(n-1)}{2}]$, both selected by the TPE algorithm. Parameters choices for NNs are presented in Table 4.3.

As we can see, NN models outperform LR and LR-RBF, both in PFF-ASHGs and PFF-BHGs, especially as the number of agents increases. This is, in fact, an anticipated result since the hyper-parameters optimization makes NN models more adaptive to the problem.

Neural Networks Parameter Choices	
Optimization Algorithm	Adam
Epochs	200
Hidden Layers Activators	ReLU
Output Activator (PFF-ASHG)	Linear
Output Activator (PFF-BHG)	Sigmoid
Error Function (PFF-ASHG)	MSE
Error Function (PFF-BHG)	binary crossentropy

Table 4.3: Parameter choices for Neural Networks

n	PFF-ASHGs			PFF-BHGs		
	LR	LR-RBF	NN	LR	LR-RBF	NN
5	0.05sec	1.2sec	14sec	0.04sec	1sec	45sec
10	2.4sec	17sec	1.7min	1.3sec	12sec	2.9min
20	10sec	1.7min	8.3min	5.5sec	1.2min	8.4min
50	3.2min	25.3min	5.5hr	1.7min	30.6min	2.5hr

Table 4.4: Approximate time needed per game for training & testing

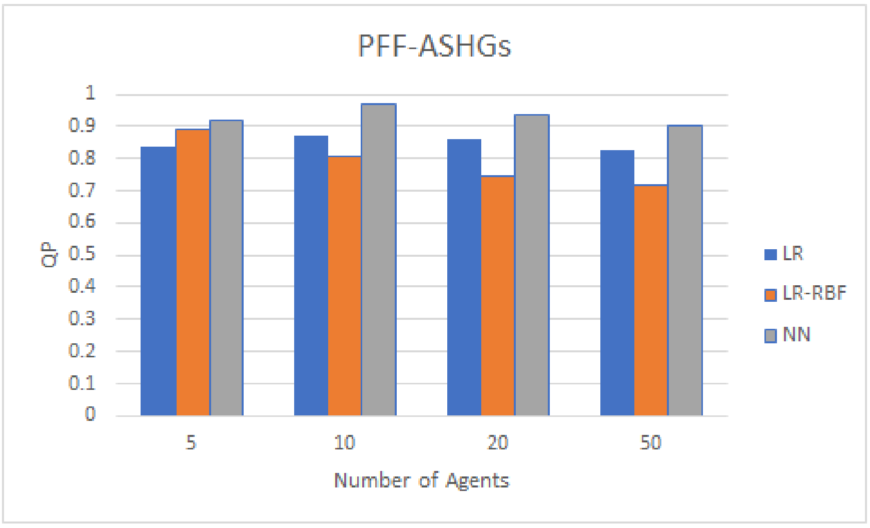

Figure 4.1: Average QP for all PFF-ASHG games with 5, 10, 20, and 50 agents

n	Partition samples	
	Training	Testing
5	200	500
10	2000	5000
20	5000	10000
50	20000	40000

Table 4.5: Samples per setting

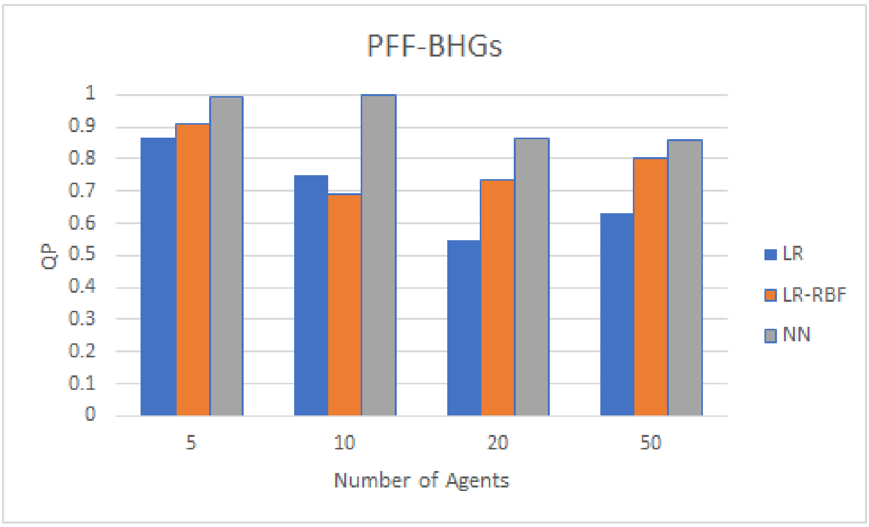

Figure 4.2: Average QP for all PFF-BHG games with 5, 10, 20, and 50 agents

Additional results for Neural Networks are showcased alongside our formation protocols (Figures 4.13,4.16,4.19,4.22), that illustrate the the average QP in an online setting.

Evaluation of our 3 different models is observed in Figures 4.1 and 4.2. As expected, the NN architecture, combined with the adaptive capabilities of TPE, outperforms the other models. However there is a trade-off between execution time and performance 4.4. As we can observe, PFF-ASHGs scale better, while in PFF-BHGs, there is a noticeable drop in performance as the number of agents increase. This can be attributed to the higher complexity of Formulae as the number of agents increase. Comparing the results of LR, they give us more insight about the higher complexity in PFF-BHGs, especially beyond 20 agents.

4.2 Generating new Partitions

In Figures 4.3,4.4,4.5,4.7,4.10 we showcase the performance of our method, specifically the average utility and unobserved ratio of generated partitions. Especially in PFF-BHG settings, the trade off between ”good” generated partitions and them being unobserved is apparent. In PFF-ASHG settings, as discussed in 3.3.1, a threshold needs to exist in order to classify a partition as ”good”. In our experiments we set $r_t = 0.8$.

Experimental results show that the number of components does not affect the average utility of generated partitions. But, for higher number of training samples, average utility tends to increase, even for the same threshold $r_t = 0.8$. This is expected, as for a small number of samples, we will choose more samples that might be below the defined threshold. A requirement of a minimum number of samples (20) exists for each model. In each setting, GMM generates 1000 samples for evaluation/computation of the average utility and unobserved ratio.

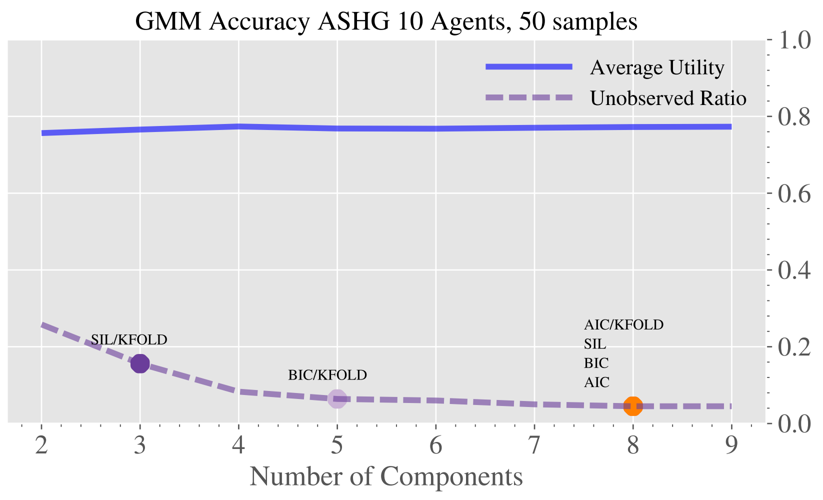

Figure 4.3: Average utility, Unobserved ratio of generated samples for a PFF-ASHG game with 10 agents and 50 samples.

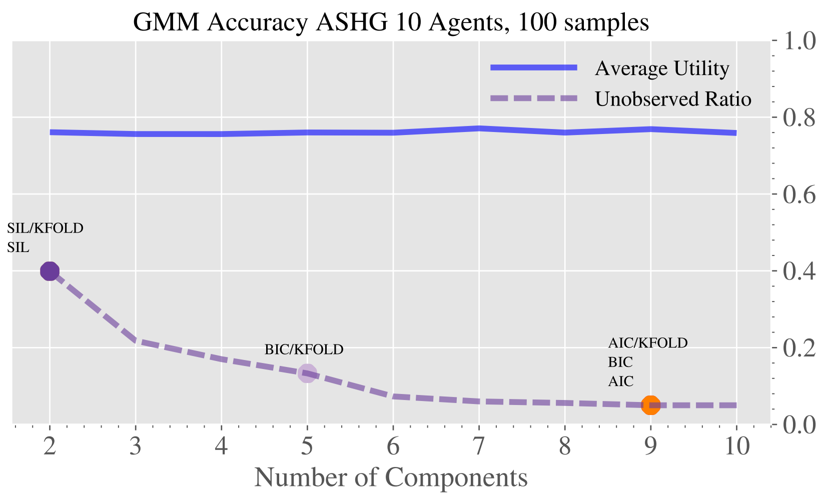

Figure 4.4: Average utility, Unobserved ratio of generated samples for a PFF-ASHG game with 10 agents and 100 samples.

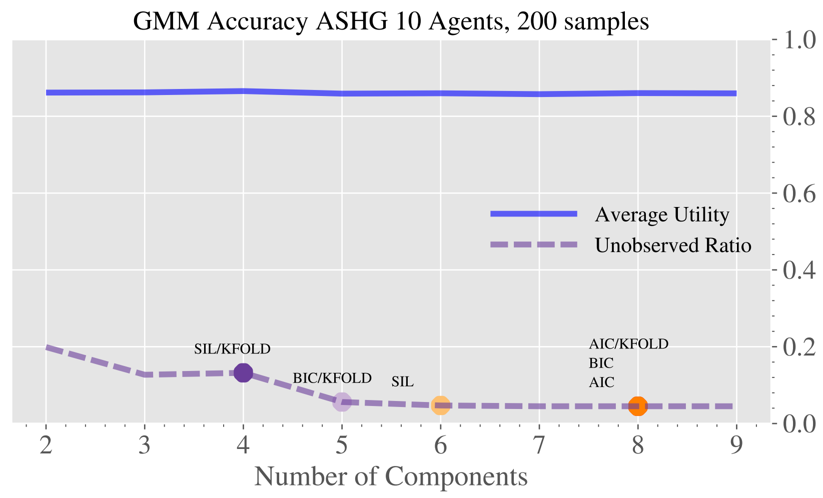

Figure 4.5: Average utility, Unobserved ratio of generated samples for a PFF-ASHG game with 10 agents and 200 samples.

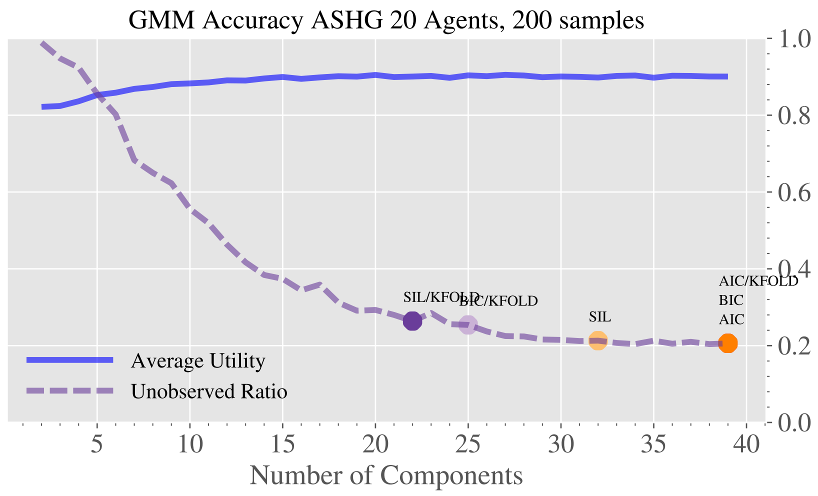

Figure 4.6: Average utility, Unobserved ratio of generated samples for a PFF-ASHG game with 20 agents and 200 samples.

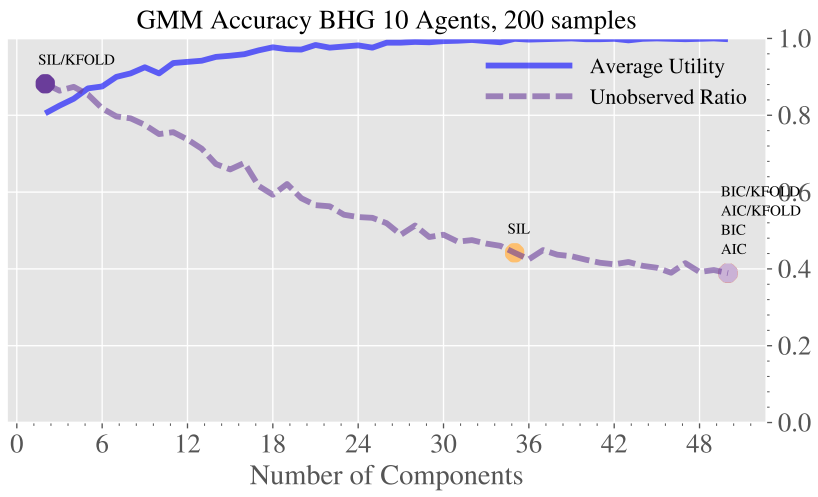

Figure 4.7: Average utility, Unobserved ratio of generated samples for a PFF-BHG game with 10 agents and 200 samples.

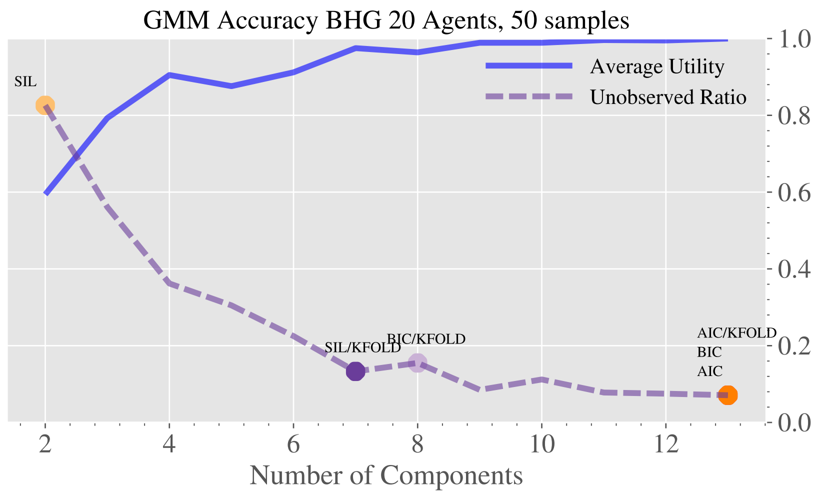

Figure 4.8: Average utility, Unobserved ratio of generated samples for a PFF-BHG game with 20 agents and 50 samples.

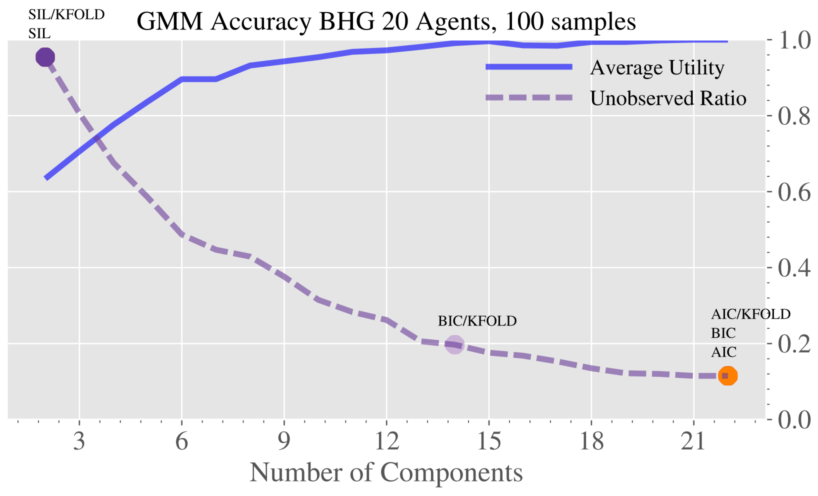

Figure 4.9: Average utility, Unobserved ratio of generated samples for a PFF-BHG game with 20 agents and 100 samples.

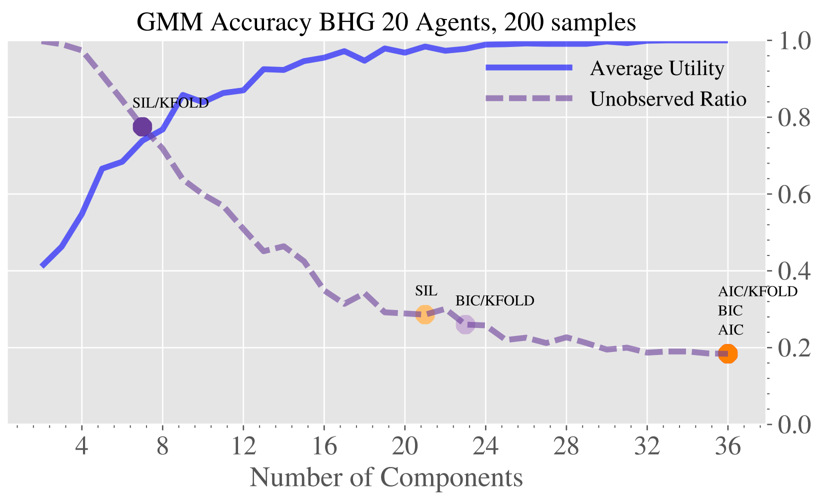

Figure 4.10: Average utility, Unobserved ratio of generated samples for a PFF-BHG game with 20 agents and 200 samples.

All aforementioned metrics (Subsection 3.3.2) that select the number of components with the maximum score are examined. We also test these metrics along with k-fold cross validation of our data, as the use of this technique further enhances the choices of these metrics, resulting in models that combine both properties. For a given unlabeled dataset $D = \{\mathbf{x}_1, \cdots, \mathbf{x}_d\}$, and a number of components n, k-fold splits D to two datasets. The model's training can be performed on the training split of k-fold, while each metric can be examined with the testing dataset for each iteration of k-fold. On average, the SIL\KFOLD metric provides us with a model that best combines both properties in most settings. All results were obtained with exhaustive search for the number c of components. The TPE algorithm is utilized alongside GMMs in the formation protocols.

4.3 Evaluating the different Coalition Formation Protocols

Experimental evaluation of both formation protocols is showcased below. As discussed in Section 3.4.3, agents act randomly in order to collect data, and only after a certain threshold of QP is exceeded, then agents exploit our methods for predicting preferences and offering partitions. Note that QP from observed data will not be the same as average QP from a large test dataset, as observed data will be far less, thus making it less likely to detect deviations in predicted labels. Figures 4.13,4.16,4.19,4.22 provide QP results, specifically QP $- Test$, which is the average QP from a dataset of 5000 samples (used only for evaluation, agents do not exploit this data for training) and $\mathrm{QP} - \mathit{Observed}$ which is the average QP of the last $2 \cdot \mathrm{Batch}$ partitions observed. Since $\mathrm{QP} - \mathit{Observed}$ is averaged across all agents, this value is closer to a more accurate $\mathrm{QP} - \mathit{Test}$, but each individual agent may have higher deviation. Table 4.6 contains our parameter choices.

In all settings, we compare two types of agents:

- **RationalAgent**: the agent resulting from our approach, who is unaware of its preferences, and initially acts as a random agent.
- **InformedAgent**: the agent who has access to its true preference relation, and only exploits a GMM model in order to produce new offers in a collaboration protocol.

We implicitly also compare both types of agents with a **RandomAgent** who perform random actions, via the behaviour of rational agents in the beginning, when they are uncertain. Additionally, the impact of GMMs can be observed on agents with the true preference relation (InformedAgent).

NNs were used as the regression model due to their adaptability, and GMMs with SIL\KFOLD metric, alongside the TPE algorithm, as a partition generator for new offers.

Evaluation on both formation protocols is showcased in Figures 4.11,4.12,4.14,4.15, 4.17, 4.18,4.20, 4.21, alongside the corresponding QP values for each game setting (Figures 4.13,4.16,4.19,4.22). Agents initially act randomly, as discussed in Subsection 3.4.3, and naturally, games with 20 agents require more samples for agents to exceed the desired threshold $\mathrm{QP}_{threshold}$, as the preferences are more complex. PFF-BHGs exhibit more fluctuation in the resulting social welfare, especially for 20 agents. This behaviour can be attributed to the fact that formulas contain many pairs (see Table 4.1). In settings with 20 agents, agents in PFF-ASHG settings become ”confident” about their approximated utility function earlier than in PFF-BHG. This behaviour is expected, as showcased in Figures 4.1, 4.2.

In settings with 10 agents, little to no performance gain can be obtained from GMMs, by observing the behaviour of informed agents. This can be attributed either to the fact that Informed Agents do not have as distributed data as Rational Agents, who act with random behaviour initially, or perhaps that the social welfare cannot improve further with this method. In settings with 20 agents, performance gain is quite distinct.

Parameter	10 agents	20 agents
Regression Model	Neural Network	Neural Network
$QP_{threshold}$	0.85	0.75
Offers per agent q (Copeland)	5	2
Batch (Training after every #Batch samples)	50 samples	100 samples

Table 4.6: Parameters choices for Coalition Formation

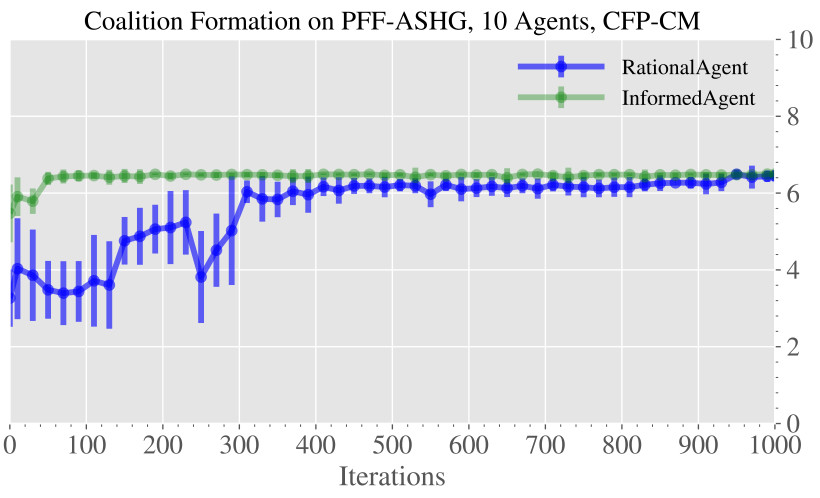

Figure 4.11: Average Social Welfare for 1000 iterations, for a PFF-ASHG game with 10 agents, using CFP-CM protocol for a RationalAgent and an InformedAgent.

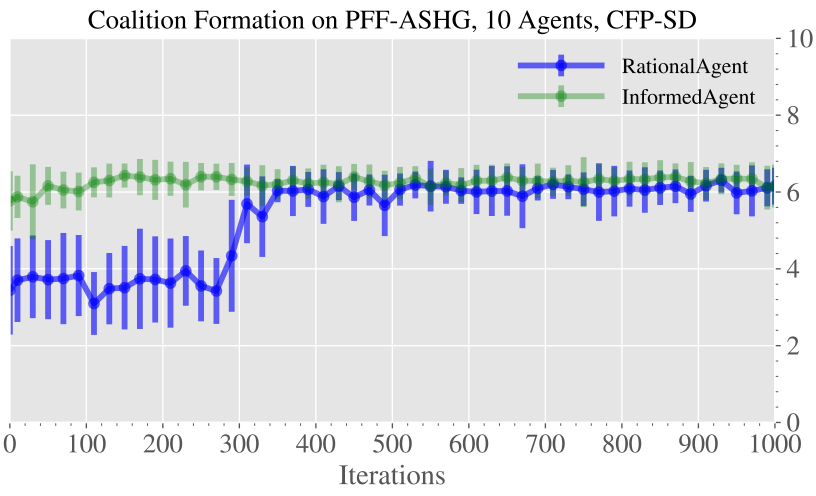

Figure 4.12: Average Social Welfare for 1000 iterations, for a PFF-ASHG game with 10 agents, using CFP-SD protocol for a RationalAgent and an InformedAgent.

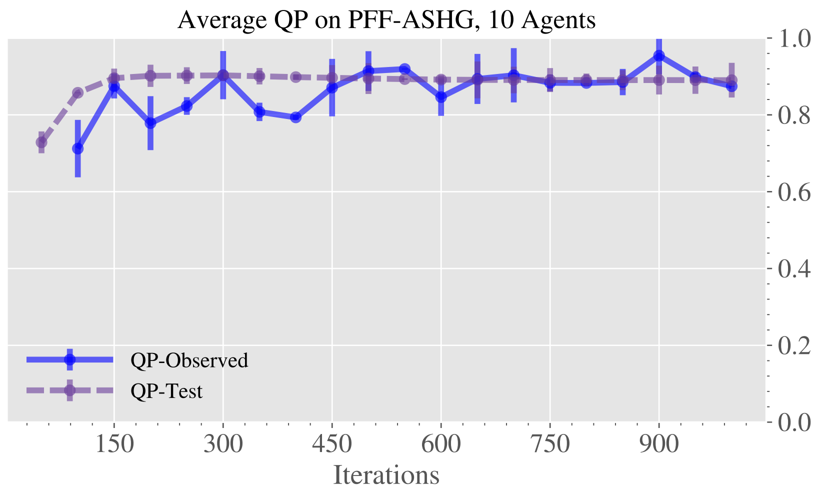

Figure 4.13: Average QP for a PFF-ASHG game with 10 agents, for a RationalAgent.

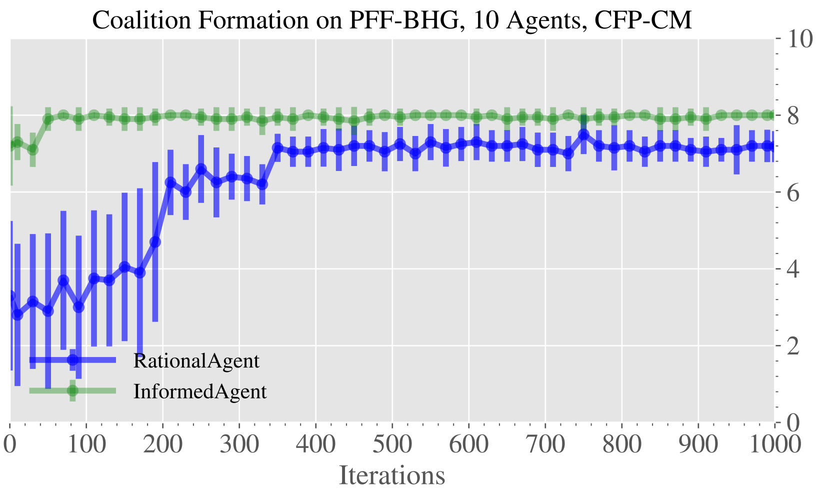

Figure 4.14: Average Social Welfare for 1000 iterations, for a PFF-BHG game with 10 agents, using CFP-CM protocol for a RationalAgent and an InformedAgent.

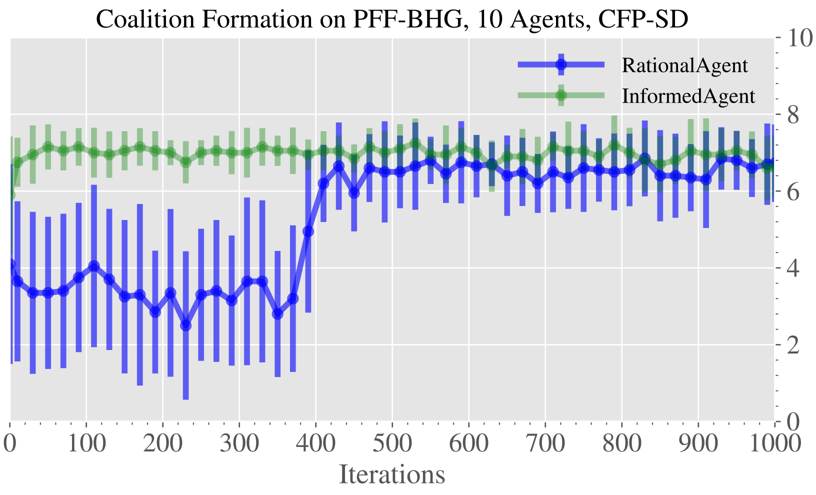

Figure 4.15: Average Social Welfare for 1000 iterations, for a PFF-BHG game with 10 agents, using CFP-SD protocol for a RationalAgent and an InformedAgent.

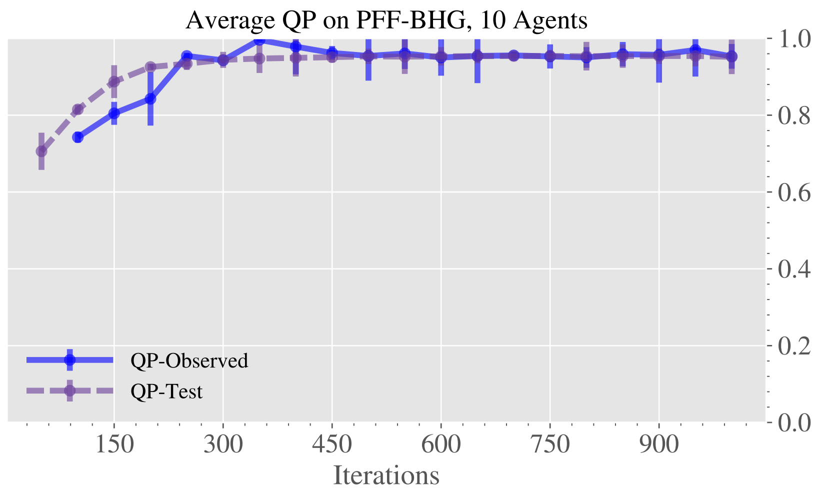

Figure 4.16: Average QP for a PFF-BHG game with 10 agents, for a RationalAgent.

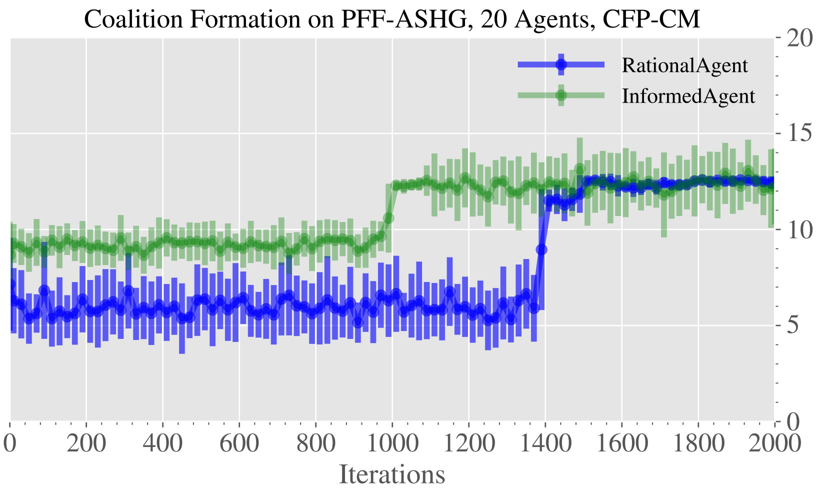

Figure 4.17: Average Social Welfare for 2000 iterations, for a PFF-ASHG game with 20 agents, using CFP-CM protocol for a RationalAgent and an InformedAgent.

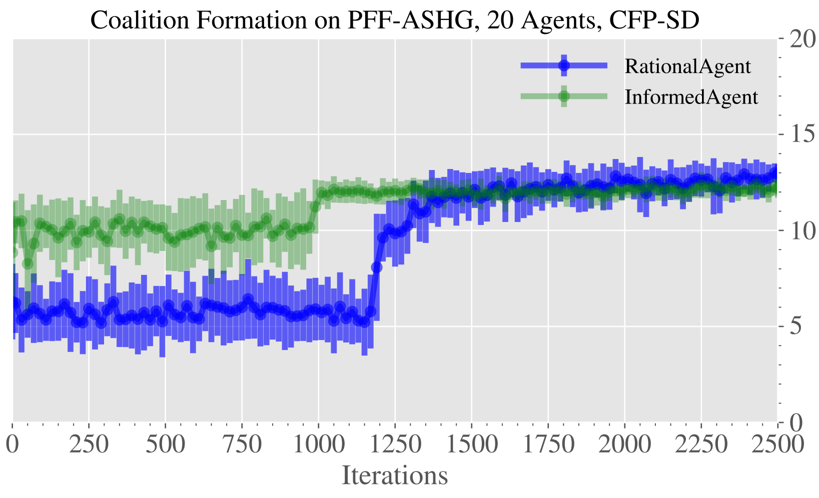

Figure 4.18: Average Social Welfare for 2000 iterations, for a PFF-ASHG game with 20 agents, using CFP-SD protocol for a RationalAgent and an InformedAgent.

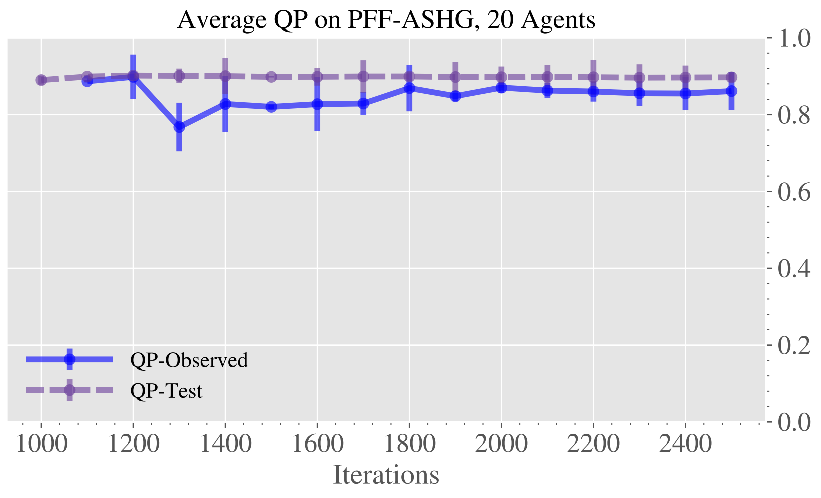

Figure 4.19: Average QP for a PFF-ASHG game with 20 agents, for a RationalAgent.

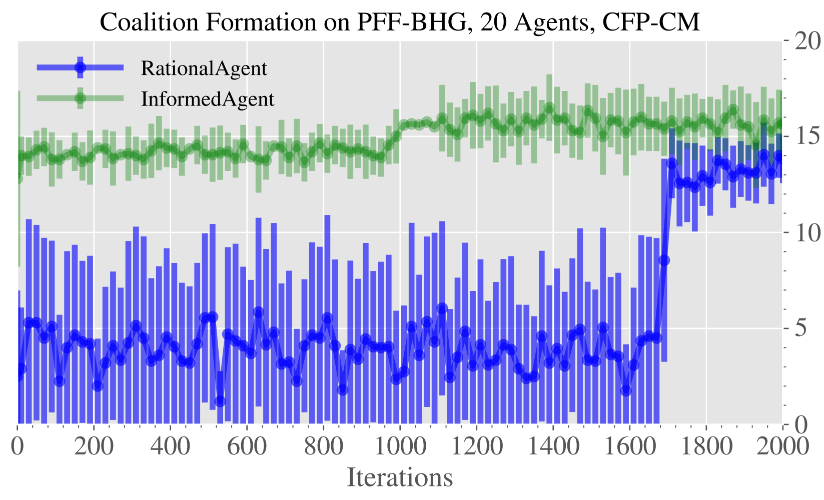

Figure 4.20: Average Social Welfare for 1000 iterations, for a PFF-BHG game with 20 agents, using CFP-CM protocol for a RationalAgent and an InformedAgent.

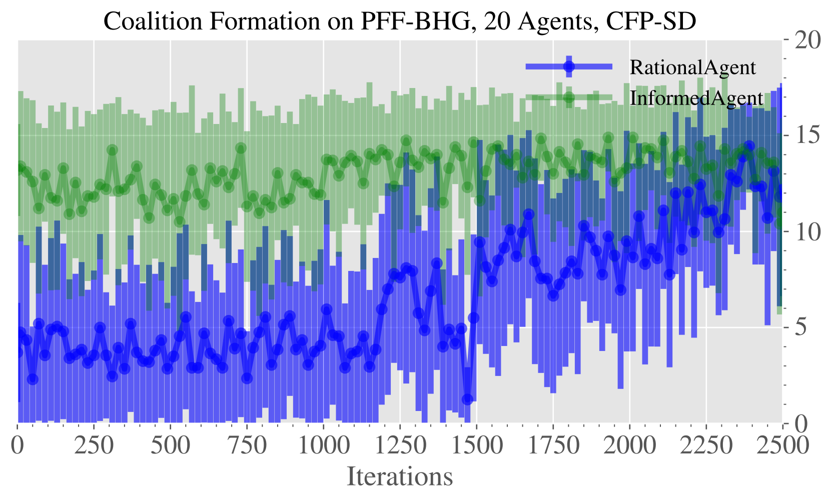

Figure 4.21: Average Social Welfare for 2000 iterations, for a PFF-BHG game with 20 agents, using CFP-SD protocol for a RationalAgent and an InformedAgent.

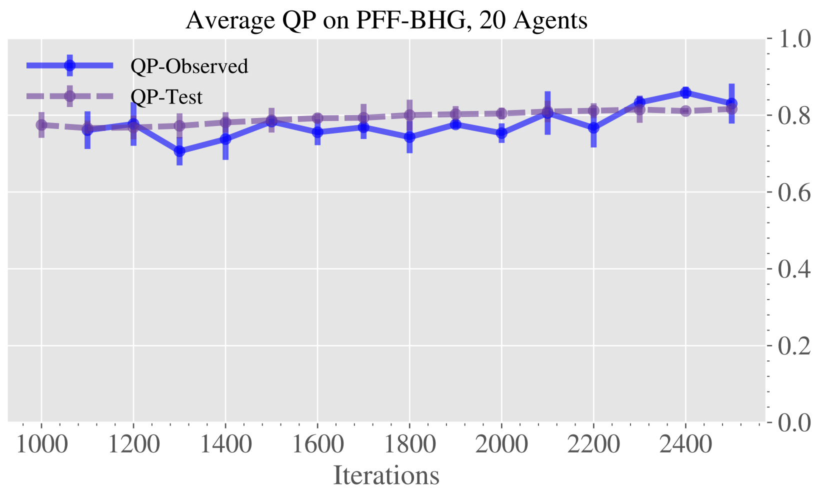

Figure 4.22: Average QP for a PFF-BHG game with 20 agents, for a RationalAgent.

Both protocols illustrate similar performance, but CFP-CM exhibits more convergence towards a social welfare value after agents stop acting randomly, while CFP-SD retains a level of randomness. For this reason, we examine CFP-SD for more iterations for 10 agents, in order to capture a potential increase in performance. Rational agents, after training on a large enough dataset, perform very similarly to informed agents, even though they are not aware of their preference relations, further illustrating the effectiveness of our approach on preference extraction (Section 3.2). CFP-CM provides us with slightly better results in PFF-BHGs, while in PFF-ASHGs, results are quite similar. Overall, **CFP-CM** is the dominant protocol in our settings.

We also examine our approach on a PFF-BHG game that was modified in order to contain partitions that satisfy all agents, by adjusting an existing generated game's formulas with 10 agents. Results of rational agents performance is showcased in Figures 4.23, 4.24. Both protocols perform better in this environment, but the dominance of CFP-CM protocol is unambiguous in this setting. However, a partition that satisfies all agents was observed only in CFP-SD, due to its randomness. CFP-CM protocol was examined again, but now, since offers depend on GMMs and a small amount of data was collected, we attempt to delay the learning process to collect more data. Delaying the learning process did not have an effect on other games, possibly due to the already large collection of data in other games. Figure 4.25 demonstrates the obtained results with this method.

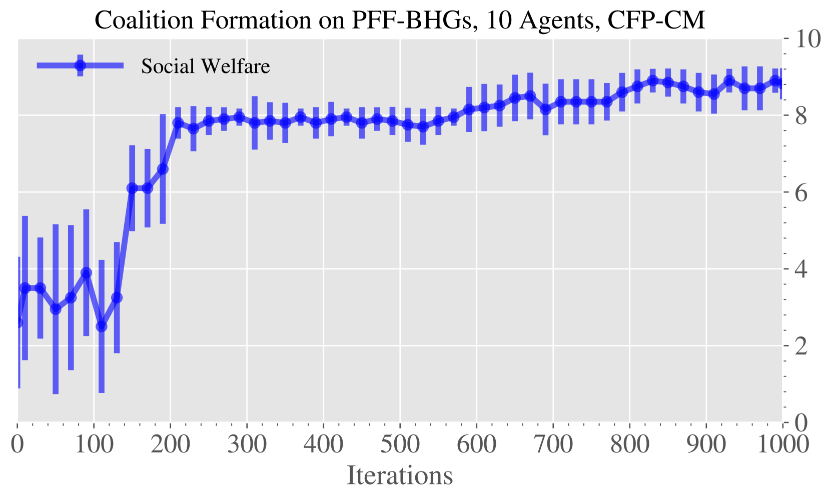

Figure 4.23: Average Social Welfare for 1000 iterations, for a PFF-BHG modified game with 10 agents, using CFP-CM protocol for a RationalAgent.

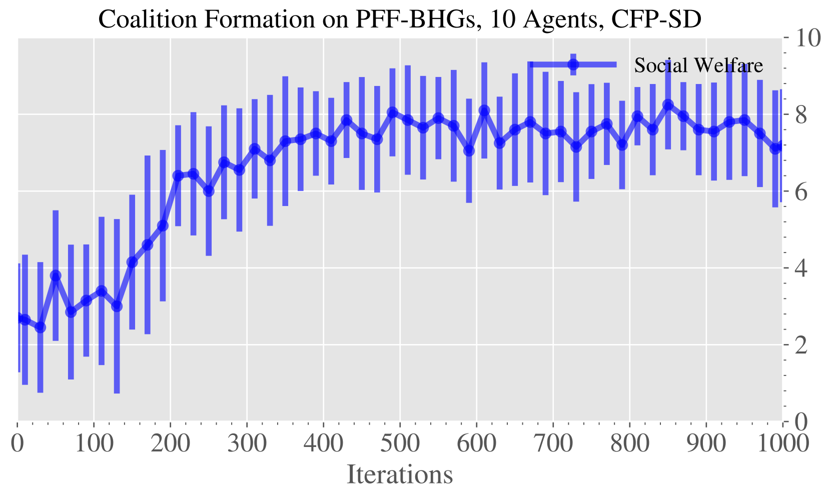

Figure 4.24: Average Social Welfare for 1000 iterations, for a PFF-BHG modified game with 10 agents, using CFP-SD protocol for a RationalAgent.

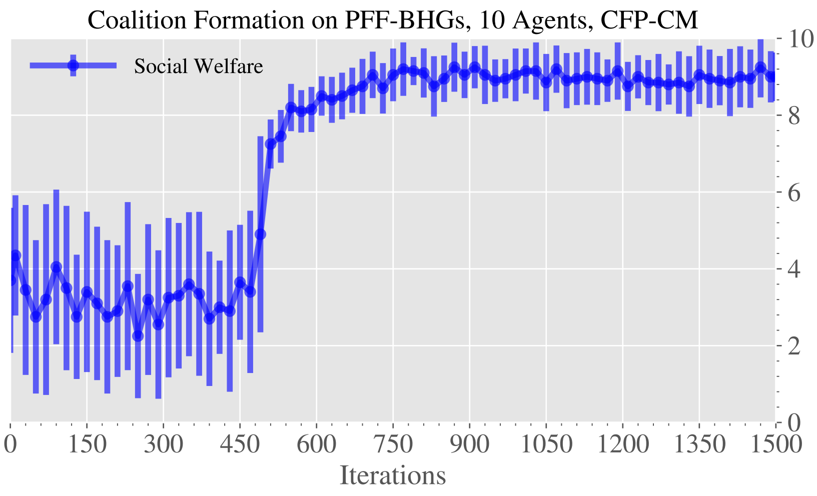

Figure 4.25: Average Social Welfare for 1500 iterations, for a PFF-BHG modified game with 10 agents, using CFP-CM protocol for a RationalAgent with delayed learning process.

Chapter 5

Conclusions

5.1 Summary

This thesis focused on agents in *Hedonic Games with Externalities* who are *unaware* of their underlying utility function and have *conflicting* preferences over partitions. We extended two hedonic game classes to partition function form, applied supervised learning techniques for these new settings, and evaluated approximations with a suitable metric. Moreover, GMMs allowed us to generate new satisfiable partitions from available data. Finaly, we proposed two coalition formation protocols for agents with preferences over partitions, and incorporated the aforementioned methods in these protocols in order to examine their effectiveness in a multi-agent setting, attempting to maximize the social welfare of such agents.

Throughout our experimental evaluation, we conclude that agents in these settings can approximately extract their own preferences, even for a large set (50 agents). Moreover, generated partitions in PFF-BHG (Boolean Hedonic Games in Partition Function Form) settings create a trade-off between partition that satisfy agents and partitions that are unobserved, while in PFF-ASHG (Additively Separable Hedonic Games in Partition Function Form) this is hardly noticeable. Both coalition formation protocols undoubtedly improve upon a random policy, and succeed in forming partitions that approximately satisfy at least 60% of agents, even in settings with 20 agents, environments with high complexity, conflicting preferences and without the presence of a centralized entity or exchange of information between agents.

5.2 Future Work

Since our work is focused on a novel class of games, many future work extensions can be considered. *Recurrent Neural Networks* can be employed as regression models for our problem at hand, due to their property for no predetermined input size. A different input encoding can be considered, possibly with a non-fixed input size, which might be befitting to encoding partitions, as they contain a dynamic size of coalitions. We could also adopt *Preference Learning* (Fürnkranz and Hüllermeier, 2010) methods that can be utilized for ranking partitions based on observed data, as they are seemingly suitable for extracting preferences over partitions. Additionally, it would be interesting to study hedonic games in partition function form with PAC learning, and extend the work in (Sliwinski and Zick, 2017), especially in the context of finding stable partitions for PFF-ASHGs and PFF-BHGs.

Furthermore, we could consider different metrics on GMMs. A preliminary notion for a metric could be to combine a model's *accuracy* with the ratio of *unobserved* partitions. Additionally, our method for generating partition (in Section 3.3) could be examined in different environments, i.e. in other games with externalities or any domain that requires effective partitioning for a given set of objects and labeled data are provided.

Moreover, instead of a single deviation on CFP-SD, each agent in the formation protocol could consider searching in a different partition space. A preliminary notion could be for each agent to select a new partition that performs one *merge* or *split* operation of different coalitions on the last proposed partition (inspired by Apt and Witzel, 2009). Another interesting consideration is to approach the problem of coalition formation of agents with preferences over partitions and incomplete information with *Multi-agent Reinforcement Learning* techniques, i.e. attempt to model the coalition formation process as a state-action function, with a reward function, combined with the various methodologies for this multi-agent domain.

Bibliography

Akaike, H. (Dec. 1974). “A new look at the statistical model identification”. In: *IEEE Transactions on Automatic Control* 19.6, pp. 716–723. ISSN: 0018-9286. DOI: 10.1109/TAC.1974.1100705 (cit. on p. 40).

Apt, Krzysztof R and Andreas Witzel (2009). “A generic approach to coalition formation”. In: *International Game Theory Review* 11.03, pp. 347–367 (cit. on p. 70).

Avrachenkov, Konstantin E. et al. (Oct. 2018). “Network partitioning algorithms as cooperative games”. In: *Computational Social Networks* 5.1, p. 11. ISSN: 2197-4314. DOI: 10.1186/s40649-018-0059-5. URL: https://doi.org/10.1186/s40649-018-0059-5 (cit. on p. 1).

Aziz, Haris, Paul Harrenstein, et al. (2016). “Boolean Hedonic Games”. In: *Proc. of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*. KR’16. AAAI Press, pp. 166–175 (cit. on pp. 7, 27, 42).

Aziz, Haris, Rahul Savani, and Hervé Moulin (2016). “Hedonic Games”. In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. Cambridge University Press, pp. 356–376 (cit. on pp. 1, 6, 7).

Banerjee, Suryapratim, Hideo Konishi, and Tayfun Sönmez (Jan. 2001). “Core in a simple coalition formation game”. In: *Social Choice and Welfare* 18.1, pp. 135–153. ISSN: 1432-217X. DOI: 10.1007/s003550000067. URL: https://doi.org/10.1007/s003550000067 (cit. on p. 6).

Bell, E. T. (1934). “Exponential Polynomials”. In: *Annals of Mathematics* 35.2, pp. 258–277. ISSN: 0003486X. URL: http://www.jstor.org/stable/1968431 (cit. on p. 23).

Bergstra, James S et al. (2011). “Algorithms for hyper-parameter optimization”. In: *Proc. of NIPS-2011*, pp. 2546–2554 (cit. on p. 16).

Bergstra, James, Daniel Yamins, and David Daniel Cox (2013). “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures”. In: (cit. on p. 49).

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag (cit. on pp. 8, 9, 17, 18, 21).

Brandt, Felix et al. (2016). *Handbook of computational social choice*. Cambridge University Press (cit. on p. 44).

Chalkiadakis, Georgios, Edith Elkind, and Michael Wooldridge (2011). *Computational Aspects of Cooperative Game Theory (Synthesis Lectures on Artificial Inetlligence and Machine Learning)*. 1st. Morgan & Claypool Publishers. ISBN: 1608456528, 9781608456529 (cit. on pp. 1, 5).

Darmann, Andreas et al. (2012). “Group Activity Selection Problem”. In: *Proceedings of the 8th International Conference on Internet and Network Economics*. WINE’12. Liverpool, UK: Springer-Verlag, pp. 156–169. ISBN: 978-3-642-35310-9. DOI: 10.1007/978-3-642-35311-6_12. URL: http://dx.doi.org/10.1007/978-3-642-35311-6_12 (cit. on p. 1).

Dempster, Arthur P, Nan M Laird, and Donald B Rubin (1977). “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22 (cit. on p. 19).

Duchi, John, Elad Hazan, and Yoram Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159 (cit. on p. 14).

Elkind, Edith and Michael Wooldridge (2009). “Hedonic Coalition Nets”. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. AAMAS’09. Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems, pp. 417–424. ISBN: 978-0-9817381-6-1 (cit. on p. 7).

Fürnkranz, Johannes and Eyke Hüllermeier (2010). *Preference learning*. Springer (cit. on p. 70).

G. Saari, Donald and Vincent Merlin (Feb. 1996). “The Copeland method (*).” In: *Economic Theory* 8, pp. 51–76. DOI: 10.1007/BF01212012 (cit. on pp. 19, 44).

Georgara, Athina, Thalia Ntiniakou, and Georgios Chalkiadakis (2018). “Learning Hedonic Games via Probabilistic Topic Modeling”. In: *European Conference on Multi-Agent Systems*. Springer, pp. 62–76 (cit. on pp. 2, 8).

Georgara, Athina, Dimitrios Troullinos, and Georgios Chalkiadakis (2019). “Extracting Hidden Preferences over Partitions in Hedonic Cooperative Games”. In: *Knowledge Science, Engineering and Management*. Ed. by Christos Douligeris, Dimitris Karagiannis, and Dimitris Apostolou. Cham: Springer International Publishing, pp. 829–841. ISBN: 978-3-030-29551-6 (cit. on pp. 23, 32).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press (cit. on p. 11).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc. (cit. on p. 8).

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (n.d.). “Neural Networks for Machine Learning Lecture 6a Overview of mini–batch gradient descent”. In: () (cit. on p. 14).

Jang, I., H. Shin, and A. Tsourdos (Dec. 2018). “Anonymous Hedonic Game for Task Allocation in a Large-Scale Multiple Agent System”. In: *IEEE Transactions on Robotics* 34.6, pp. 1534–1548. ISSN: 1552-3098. DOI: 10.1109/TRO.2018.2858292 (cit. on p. 1).

Janovsky, P. and S. A. Deloach (Oct. 2016). “Multi-agent Simulation Framework for Large-Scale Coalition Formation”. In: *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pp. 343–350. DOI: 10.1109/WI.2016.0055 (cit. on p. 41).

Kendall, Maurice G. (1948). *Rank correlation methods*. London: Griffin. VII, 160 (cit. on p. 32).

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: (cit. on p. 15).

Lippman, David and Pierce College (2013). *Math in Society* (cit. on p. 45).

MacQueen, James et al. (1967). “Some methods for classification and analysis of multivariate observations”. In: *Proc. of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA, pp. 281–297 (cit. on p. 17).

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `http://tensorflow.org/` (cit. on p. 49).

McLachlan, Geoffrey J and Kaye E Basford (1988). “Mixture models: inference and applications to clustering”. In: (cit. on p. 18).

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press. ISBN: 0262018020, 9780262018029 (cit. on p. 21).

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on p. 49).

Procaccia, Ariel D. (June 2008). “Mathematical Foundations of AI”. `http://www.cs.cmu.edu/~arielpro/mfai_papers/lecture6.pdf`. Lecture 6 (cit. on p. 45).

Refaeilzadeh, Payam, Lei Tang, and Huan Liu (2009). “Cross-Validation”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: `10.1007/978-0-387-39940-9_565`. URL: `https://doi.org/10.1007/978-0-387-39940-9_565` (cit. on p. 36).

Reynolds, Douglas A. (2009). “Gaussian Mixture Models”. In: *Encyclopedia of Biometrics* (cit. on p. 18).

Rousseeuw, Peter J. (1987). “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20, pp. 53–65. ISSN: 0377-0427. DOI: `https://doi.org/10.1016/0377-0427(87)90125-7`. URL: `http://www.sciencedirect.com/science/article/pii/0377042787901257` (cit. on p. 40).

Ruder, Sebastian (2016). “An overview of gradient descent optimization algorithms”. In: (cit. on pp. 13–15).

Saad, W. et al. (2012). “Coalitional Games in Partition Form for Joint Spectrum Sensing and Access in Cognitive Radio Networks”. In: *IEEE Journal of Selected Topics in Signal Processing*, pp. 195–209 (cit. on p. 6).

Schwarz, Gideon et al. (1978). “Estimating the dimension of a model”. In: *The annals of statistics* 6.2, pp. 461–464 (cit. on p. 40).

Sliwinski, Jakub and Yair Zick (2017). “Learning Hedonic Games”. In: *Proc. of the 26th IJCAI-17*, pp. 2730–2736 (cit. on pp. 2, 70).

Taywade, Kshitija, Judy Goldsmith, and Brent Harrison (Jan. 2019). “Decentralized Multiagent Approach for Hedonic Games: 16th European Conference, EUMAS 2018, Bergen, Norway, December 6–7, 2018, Revised Selected Papers”. In: pp. 220–232. ISBN: 978-3-030-14173-8. DOI: `10.1007/978-3-030-14174-5_15` (cit. on p. 41).

Thrall, R. M. and W. F. Lucas (1963). “N-person games in partition function form”. In: *Naval Research Logistics Quarterly* 10.1, pp. 281–298 (cit. on p. 1).