Shapley Values as an Approximator for an Optimal Bias Bounty Permutation

Declan Harrison, Francesca Marini
University of Pennsylvania

Abstract

The Bias Bounty 'Kaggle' style competition completed in CIS 523 generated a unique, raw data set of finite subgroup-hypothesis pairs all designed for the same binary classification task. Choosing which pairs to introduce to a new pointer decision list (PDL) framework as well as the order in which to introduce them to produce a PDL with minimal error is left to be an open question. The intent of this paper is to examine a greedy method to identifying approximately optimal orderings of pairs from a finite universe using a notion of fair payoffs via Shapley values from coalition game theory. We attempt to provide a rudimentary investigation of this novel application of Shapley Values to Bias Bounties in order to spur future research inquiry on this topic.

1 Motivation

In the CIS 523 Bias Bounty Project Globus-Harris, Kearns, and Roth (2022), teams independently created pointer decision list (PDL) models by locally submitting subgroup-hypothesis pairs for the same binary classification task. Like a Kaggle-style competition, each team submitted their PDL and it is simple to pick a 'winner': the group whose PDL has the lowest error over the holdout test set X. However, this framework of a bias bounty model is designed for crowd-sourcing a model; to create a PDL indistinguishable from the Bayes Optimal classifier, no person should be able to introduce a new subgroup-hypothesis pair that can improve such a globally defined PDL model. To simulate this global structure from each team's submitted PDL models, we could naively take the 'winner's' PDL and attempt updates on this model using all other teams subgroup-hypothesis pairs. More technically written, suppose f_w is the PDL with the lowest overall error on the test data. Take all other teams' PDLs and create a universe U of the remaining, arbitrarily labeled, subgroup-hypothesis pairs $U = \{(g_1, h_1), \dots, (g_m, h_m)\}$. Then, we run

```
For pair in U:
    update(f_w, pair)
```

where the function *update* works in the same manner as *cscUpdater.iterative_update*.

This more or less models the crowd-sourced format of having a universal PDL where update queries are predefined and finite. There are arguments to be made that this does not follow the theoretical model as teams cannot query the current condition of the universal PDL and attempt updates over certain groups, but this is not the focus of this paper. Rather, we want to find the best PDL given a set of predefined subgroup-hypothesis pairs. We trivially note that since we are working in a finite universe of pairs, we can enumerate all permutations and $\exists\sigma$ order permutation such that $f_\sigma = f_{opt}$ where f_σ is defined by running n iterations of update$(f, (g_i, h_i))$ and the order in which (g_i, h_i) pairs are introduced is defined by the ordering σ. Then, after running all updates $(g_i, h_i) \in U$ on f_w, we know $\exists f_{opt}$ such that $\varepsilon_{f_{opt}}(X) \leq \varepsilon_{f_w}(X)$ where $\varepsilon_f(X)$ is the error of a model f with respect to the test data X. To exemplify why were care about this, we follow a toy example where $\varepsilon_{f_{opt}} < \varepsilon_{f_w}$ even after all updates are made:

Let our data set $X = [x_1, x_2, x_3, x_4]$ be four instances with true labels y_1, y_2, y_3, y_4. We define three group functions g_1, g_2, g_3 where

$$g_1(X) = [1, 1, 1, 1],\ g_2(X) = [1, 1, 1, 1],\ g_3(X) = [1, 1, 0, 0].$$

Suppose the respective hypotheses h_1, h_2, h_3 predict over X such that

$$h_1(X) = [y_1, y_2, y_3, \neg y_4],\ h_2(X) = [\neg y_1, \neg y_2, y_3, y_4],\ h_3(X) = [y_1, y_2, \neg y_3, \neg y_4]$$

where $\neg y_i$ implies the prediction is not equal to the label.

Now, suppose three teams created pointer decision lists with a single of these subgroup-hypothesis pairs, i.e. three pointer decision lists f_1, f_2, f_3 are defined by $f_1 = (g_1, h_1), f_2 = (g_2, h_2), f_3 = (g_3, h_3)$ where the base model for each f_i is a predictor which misclassifies the whole data set. Then, the error of each model f_i is determined by the number of correctly labeled items in each (g_i, h_i) divided by the size of the data set. We can then compute

$$\varepsilon_{f_1}(X) = .25,\ \varepsilon_{f_2}(X) = .5,\ \varepsilon_{f_3}(X) = .5$$

We note that f_1 represents the winning team's PDL, and we attempt to run *update*$(f_1, (g_2, h_2))$ and *update*$(f_1, (g_3, h_3))$. Both of these updates will be rejected since $\varepsilon_{f_1}(g_2(X)) \leq \varepsilon_{h_2}(g_2(X))$ and $\varepsilon_{f_1}(g_3(X)) \leq \varepsilon_{h_3}(g_3(X))$. However, it is easy to see if we run *update*$(f_2, (g_3, h_3))$, this will result in a PDL f_{opt} with $\varepsilon_{f_{opt}}(X) = 0$.

In our toy example, there are six possible ways to introduce the group-hypothesis pairs to a base PDL:

$$\sigma_1 = (1,2,3),\ \sigma_2 = (1,3,2),\ \sigma_3 = (2,1,3),\ \sigma_4 = (2,3,1),\ \sigma_5 = (3,1,2),\ \sigma_6 = (3,2,1).$$

Let $f_{\sigma_1}, \ldots, f_{\sigma_6}$ be the respective pointer decision lists if the group-hypothesis pairs were introduced in the σ orderings described above. Let $\varepsilon_{f_{\sigma_i}}(X)$ denote the error of the predictions from $f_i(X)$ as opposed to the true labels. It is easy to compute each error and we list them below:

$$\varepsilon_{f_{\sigma_1}}(X) = \frac{1}{4},\ \varepsilon_{f_{\sigma_2}}(X) = \frac{1}{4},\ \varepsilon_{f_{\sigma_3}}(X) = \frac{1}{4},\ \varepsilon_{f_{\sigma_4}}(X) = 0,\ \varepsilon_{f_{\sigma_5}}(X) = \frac{1}{4},\ \varepsilon_{f_{\sigma_6}}(X) = 0.$$

We see that the order in which group-hypothesis pairs are introduced impacts the overall error of the pointer decision lists which raises an important question: how can we determine the best order to introduce pairs?

Clearly computing the error of every order permutation is infeasible; the number of permutations grows exponentially with n groups and every permutation requires up to n updates. Even computing a few of these permutations becomes expensive, thus a greedy method is required to produce an approximation for the best group ordering, which leads to our discussion on Shapley values.

2 Related Work

2.1 Bias Bounties

Since this is a term paper and given the audience reviewing the material, we assume readers have a deep understanding of the Bias Bounty project and refrain from going further in depth in this section (Globus-Harris et al., 2022).

2.2 Coalition Games and Shapley Values

The following discussion is a mixture of background definitions and axioms surrounding a game theory concept known as a coalition game. Examples in this section are novel and while definitions and axioms are standard across sources, we followed the structure of a lecture note series from (Leyton-Brown, 2022). In addition, we referenced (Bonanno, 2022) and (Chandrasekaran, 2021).

A **coalition game** (N, v) consists of a set of players $N = \{1, \ldots, n\}$ and a **characteristic function** $v : 2^n \to \mathbb{R}$ which maps subsets (or coalitions) of players S to real valued payoffs written as $v(S)$ (Leyton-Brown, 2022). A simple example of this can be seen with

$$v(S) = \begin{cases} 1 & S = N \\ 0 & S \neq N. \end{cases}$$

In this example, the only coalition that earns a payout in the game would be the entire player set. Thus, if any one player decides not to participate, then no one in the coalition gets any reward. However, if every player participates, then the coalition earns a reward of 1. For our basic example, it is intuitive that each player should earn an equal part of the reward (being $1/n$) in order to fairly distribute the payout considering each player contributed the same amount.

However, there are situations where this is not the case; suppose each player $p_1, \ldots, p_n$ has integer value $1, \ldots, n$ respectively and the characteristic function pays the sum of the player's integer values. Then, giving each player an equal portion of the reward seems unfair considering some players contributed more to the payout.

Given a coalition game (N, v), we define a payoff function $\psi : \mathbb{N} \times \mathbb{R}^{2^n} \to \mathbb{R}^n$ which returns each player's marginal payoff (Leyton-Brown, 2022). We denote player i's payoff by $\psi_i(N, v)$, and require $\sum\limits_{i \in S} \psi_i(N, v) = v(S)$. This forces the sum of each marginal payoff to be the entire subset's payoff in the coalition. In order to make this payoff function, there are agreed upon axioms for fair division of payoffs in coalition games (Leyton-Brown, 2022):

1. **Axiom 1** (Symmetry) For any subset S not containing players i and j, if $v(S \cup \{i\}) = v(S \cup \{j\})$, then i, j are said to be equivalent. For any v, if i and j are equivalent players, then $\psi_i(N, v) = \psi_j(N, v)$.
2. **Axiom 2** (Null Player) A player i is said to be null if for any subset S not containing i, $v(S \cup \{i\}) - v(S) = v(\{i\})$. For any v, if i is a null player, then $\psi_i(N, v) = v(\{i\})$.
3. **Axiom 3** (Additivity) Given two characteristic functions v_1, v_2 over the same players in different coalition games, then for any player i, we have $\psi_i(N, v_1 + v_2) = \psi_i(N, v_1) + \psi_i(N, v_2)$ where the game $(N, v_1 + v_2)$ is defined by $(v_1 + v_2)(S) = v_1(S) + v_2(S)$ for any subset S.

The following theorem is a result of Lloyd Shapley (Shapley, Roth, & Press, 1988).

Theorem 2.1 (Shapley Value). *There exists a unique function ψ which satisfies Axioms 1-3, for games with finite coalitions; it is given by the formula*

$$\psi_i(N, v) = \frac{1}{|N|!} \sum_{S \subseteq N} |S|!(|N| - |S| - 1)! \left(v(S) - v(S - \{i\})\right).$$

Intuitively, a Shapley Value is the expected marginal contribution of a player in a coalition game. Given a coalition S which contains player i, you can ask "what is the payoff value to this coalition with and without player i?", which is shown by the term $(v(S) - v(S - \{i\}))$ in the summation. The remaining terms in the summation take the average payout for player i across all coalitions S that contain player i. Then in a coalition game (N, v), the Shapley value of each player i is the average contribution and level of importance in the game. We will translate this idea to subgroup-hypothesis pairs for an optimal PDL in the bias bounty project.

2.3 Shapley Values in Bias Bounties

Suppose now that m teams from the bias bounty project submit their PDL $f_1, \ldots, f_m$, along with each subgroup-hypothesis pair that was accepted to their model. Let U be the universe of subgroup-hypothesis pairs over all submitted PDLs such that $U = \{(g_1, h_1), \ldots, (g_n, h_n)\}$. In order to simulate a coalition game, suppose that the set of n players are the subgroup-hypothesis pairs in U such that $N = U$. We slightly modify the characteristic function by forcing the size of coalitions to be n or $n-1$ such that the number of possible coalitions is $\mathcal{O}(n!)$. We force this restriction since the end goal of using Shapley Values for this problem is to find an optimal ordering in which we introduce subgroup-hypothesis pairs to a base model. Due to the theoretical guarantee of monotonic decreasing error on the PDL given updates, we only need to consider permutations which contain all pairs or permutations where one pair is missing to evaluate its overall contribution.

For the sake of clarity, we introduce notation to represent ordered sets on n and $n-1$ pairs. Let Σ be the set of all permutations existing on exactly n numbers. For $\sigma \in \Sigma$, let $v(\sigma)$ be the error on the PDL where subgroup-hypothesis pairs are introduced in σ ordering. Let $v(\sigma^{-i})$ denote an ordering σ where subgroup-hypothesis pair i (or player i) is missing from the permutation σ. Since we force the coalitions in the game to be of size n and $n-1$, we are able to simplify the Shapley Value significantly such that the Shapley Value of player i is semantically equivalent to

$$\psi_i(N, v) = \sum_{\sigma \in \Sigma} \left(v(\sigma) - v(\sigma^{-i})\right)$$

since each player i is a part of the same number of coalitions.

While our metric is not precisely equivalent to Shapley Values, it still provides a reasonable sense of a pair's importance given a permutation and, hopefully, provides insight to an optimal ordering.

3 Implementation[1]

3.1 Dataset

For our experimentation, we used data from the American Community Survey similar to the CIS 523 Bias Bounty project. The general machine learning problem was a binary classification task of deciding whether an individual makes more than $50,000 annually. Instances were individuals from California over the course of the year 2018 and used the feature set {Age, Working Class, Education Level, Marital Status, Occupation, Place of Birth, Relationship, Work Hours per Week, Binary Sex, Race}. The entire data set contained 30,000 instances and a random 80-20 split created a training and test set.

3.2 Subgroup-Hypothesis Pairs

We randomly created 10 subgroup-hypothesis pairs $(g_1, h_1), \ldots, (g_{10}, h_{10})$ that when introduced in ascending order, each were successively accepted via the updater function to a PDL using a decision stump over the entire data set as a foundation. The exact pairs can be found in the *Shapley Values.ipynb* file, but we list the subgroup functions below to give a general sense of overlap and size of subgroups:

- g_1: (X[‘WKHP’] == 40): 40 hour work week (full time).
- g_2: (X[‘WKHP’] $\leq$ 2): Less than 20 hour work week (part time).
- g_3: (X[‘RAC1P’] $\geq$ 3): Other than White, Black, or African American.
- g_4: (X[‘RAC1P’] == 1): White.
- g_5: (X[‘SCHL’] $\leq$ 12): Education level is at most 9th grade.
- g_6: (X[‘SCHL’] $\geq$ 16): Education level is at least high school graduate.
- g_7: (X[‘AGEP’] $\leq$ 30): Individuals under the age of 30.
- g_8: (X[‘COW’] == 1): Employee of a private for-profit company or business, or of an individual, for wages, salary, or commissions.
- g_9: (X[‘POBP’] $\leq$ 20) & (X[‘SEX’] == 2): Women born in states Alabama - Kansas listed alphabetically.
- g_{10}: (X[‘OCCP’] $\leq$ 100): Managers, general.

Each hypothesis has a defined random state in order to be reproducible. This is vital to finding a best permutation since classifiers that use randomness will be more or less significant depending upon their random initialization state (e.g. random forests and gradient boosting). For each of the experiments we run below, only groups $1-8$ are used in order to be computationally feasible, though $9-10$ are included for future experimentation.

3.3 Best/Worst Permutations

The first part of our experiment was centered around finding the best and worst possible errors on a PDL and was done within the script best_permutation.py. To implement this, we created a permutations list Σ of the numbers $1-8$. Then, for each $\sigma \in \Sigma$, we began on a base PDL and iteratively proposed updates (g_i, h_i) to the PDL using the order from the permutation σ. After we proposed the final update on the PDL, we computed the error of the PDL over the entire test data set. Over the course of all permutations, we maintained the best and worst error seen, initializing worst error at 0 and best error at 100. The final output of this script was two orderings $\sigma_{opt}, \sigma_{bad}$, the best and worst errors over the entire data set (related to the errors of $f_{\sigma_{opt}}, f_{\sigma_{bad}}$), and the time taken to terminate.

[1]Link to project repository.

3.4 Shapley Value Contributions

The second part of our experiment was focused on computing the expected Shapley Value of a given group in our coalition game. In *Shapley Values.ipynb*, we constructed a function *fractional_ shapley_ value* which takes a PDL, a permutation ordering σ to introduce subgroup-hypothesis pairs, and outputs an array which for each position i contains value $\varepsilon_{f_{\sigma-i}}(X) - \varepsilon_{f_\sigma}(X)$. Thus, position i in the array is the Shapley Contribution for player i given an ordering (or game) σ, computing the error of the ordering on the test data without i in σ and subtracting the error of the ordering on the data, where i is present in σ. We note that a positive Shapley contribution for player i implies the error on the ordering σ is decreased when player i is present.

We then used this function to compute the Shapley contributions of randomly select permutations drawn without replacement from the permutation list. We ran this function for a duration of time (24 hours), computing Shapley Contributions for each group over random permutations in order to compute the expected Shapley Value of every player. Below we provide a snippet of code for both functions:

Fractional Shapley Value Function

```python
def fractional_shapley_value(mod,permutation, group_list, predicate_list):

    #initialize empty array
    contribution_array = np.zeros(len(permutation))

    #save current PDL
    file = open('pdl.pkl','wb')
    pickle.dump(mod,file)

    #iterate over permuation
    for value in permutation:
        #check for improvement
        improvement_check = verifier.is_proposed_group_good_csc(mod, test_x, test_y,
                                                        predicate_list[value],group_list[value])
        if improvement_check:
            # run the update
            cscUpdater.iterative_update(mod, predicate_list[value], group_list[value], train_x,
                                        train_y, test_x, test_y, 'g'+str(value))

    #compute error over test set given permuation with all players
    permutation_error = sk.metrics.zero_one_loss(test_y,np.array(mod.predict(test_x),dtype=bool))

    #iterate over permutation
    for value in permutation:
        #load previous PDL version
        file = open('pdl.pkl','rb')
        mod = pickle.load(file)

        #for each index in the permutation, compute the PDL without the group being introduced
        for sub_value in permutation[permutation.index(value)+1:]:
            #check improvement
            improvement_check = verifier.is_proposed_group_good_csc(mod, test_x, test_y,
                                                            predicate_list[sub_value],
                                                            group_list[sub_value])

            if improvement_check:
            # run the update
                cscUpdater.iterative_update(mod, predicate_list[sub_value], group_list[sub_value],
                                            train_x, train_y, test_x, test_y, 'g'+str(sub_value))

        #compute error over test set without player i in permutation
        error_without_group = sk.metrics.zero_one_loss(test_y,np.array(mod.predict(test_x),dtype=bool))

        #add player's fractional shapley contribution to array
        contribution_array[value] += (error_without_group - permutation_error)

        #reload PDL prior to removing player i
        file = open('pdl.pkl','rb')
        mod = pickle.load(file)

        #check improvement on previous PDL given player i
        improvement_check = verifier.is_proposed_group_good_csc(mod, test_x, test_y,
                                                        predicate_list[value],group_list[value])
        if improvement_check:
            # run the update
            cscUpdater.iterative_update(mod, predicate_list[value], group_list[value],
                                        train_x, train_y, test_x, test_y, 'g'+str(value))
        #save PDL with player i include
        file = open('pdl.pkl','wb')
        pickle.dump(mod,file)

    #return fractional shapley contribution over permutation for all players
    return contribution_array

```

Expected Shapley Value Function

```python
#define groups used for permutations
groups=[0,1,2,3,4,5,6,7]

#running shapley value contribution totals
shapley_array = np.zeros(len(groups))

#all possible permutations
permutations_list = list(itertools.permutations(groups))

#define which permutations we will take
selection = np.random.choice(len(permutations_list), size=10000, replace=False, p=None)

#initialize PDL
mod = model.PointerDecisionList(initial_model.predict, [])
mod.test_errors.append(cscUpdater.measure_group_errors(mod, test_x, test_y))
mod.train_errors.append(cscUpdater.measure_group_errors(mod, train_x, train_y))

#save base PDL
file = open('basepdl.pkl','wb')
pickle.dump(mod,file)

#intialize df to record shapley values
shapley_df = pd.DataFrame(columns= groups)

#initialize counter
counter = 1

#iterate over random permutations
for index in selection:

    #reload base PDL
    file = open('basepdl.pkl','rb')
    mod = pickle.load(file)

    #define ordering
    permutation = permutations_list[index]

    #get fractional contributions
    fractional_contribution = fractional_shapley_value(mod,permutation, group_list, predicate_list)

    #add fractional contribution
    shapley_array += fractional_contribution

    #add shapley contribution to dataframe
    shapley_df.loc[counter] = fractional_contribution

    #update count
    counter += 1
    print(counter)
```

3.5 Computing Resources and Scalability

During our experiment, we used two separate computers to run the two parts of our experiment. For the first part of our experiment, we used a computer with the following specifications to run the script *best_permutation.py*:

- Operating System: Ubuntu 20.04
- Processor: AMD Threadripper 3960X: 24 cores, 3.80 GHz, 128 MB cache, PCIe 4.0
- GPU: RTX 3090 (24 GB) | 1300W PSU
- Memory: 128 GB
- Operating System Drive:m 1 TB SSD (NVMe)

This script output the following results:

- Best error: 0.15383333333333338
- Best Order: (1, 2, 3, 4, 5, 6, 7, 8)
- Worst error: 0.15733333333333333
- Worst Order: (1, 2, 3, 5, 8, 4, 6, 7)
- Total time (seconds): 78542.59732747078

We note that the total runtime of this script was roughly 22 hours on only 8 groups using a top of the line graphics card. Even increasing the total group size by one would extend this script's run time to almost a week, and using many groups makes this computationally infeasible even with modern systems such as GPU farms and compute clusters. This further promotes the need for finding a greedy metric that grants information about optimal orderings for pairs.

In the second portion of our experiment where we computed Shapley contributions, we used the following equipment:

- Operating System: Ubuntu 20.04
- Processor: Intel Core i7-10700K: 8 cores, 3.80 GHz, 16 MB cache, PCIe 4.0
- GPU: RTX 3070 Ti (8 GB) | 750W PSU
- Memory: 32 GB
- Operating System Drive:m 500 TB SSD (SATA)

We ran the *Shapley Values.ipynb* notebook cell which computes Shapley contributions for random permutations for 24 hours. During this time, we were able to complete 1,800 iterations. While we would have wished to complete all 40,000 iterations, the time constraints would not allow this as well as our goal is to find a greedy implementation, thus 1,800 entries are sufficient for approximating an optimal ordering.

4 Evaluation

While the initial idea of our project was to use the expected Shapley value of groups to create an optimal ordering, it did not appear that expected Shapley value had any correlation to an optimal ordering. No intuitive ordering of pairs expected Shapley Value produced results for permutations outside the mean error, but in the process of collecting this data we discovered that individual Shapley contributions from random permutations grant more predictive power in finding optimal/worst permutations.

4.1 Main Results

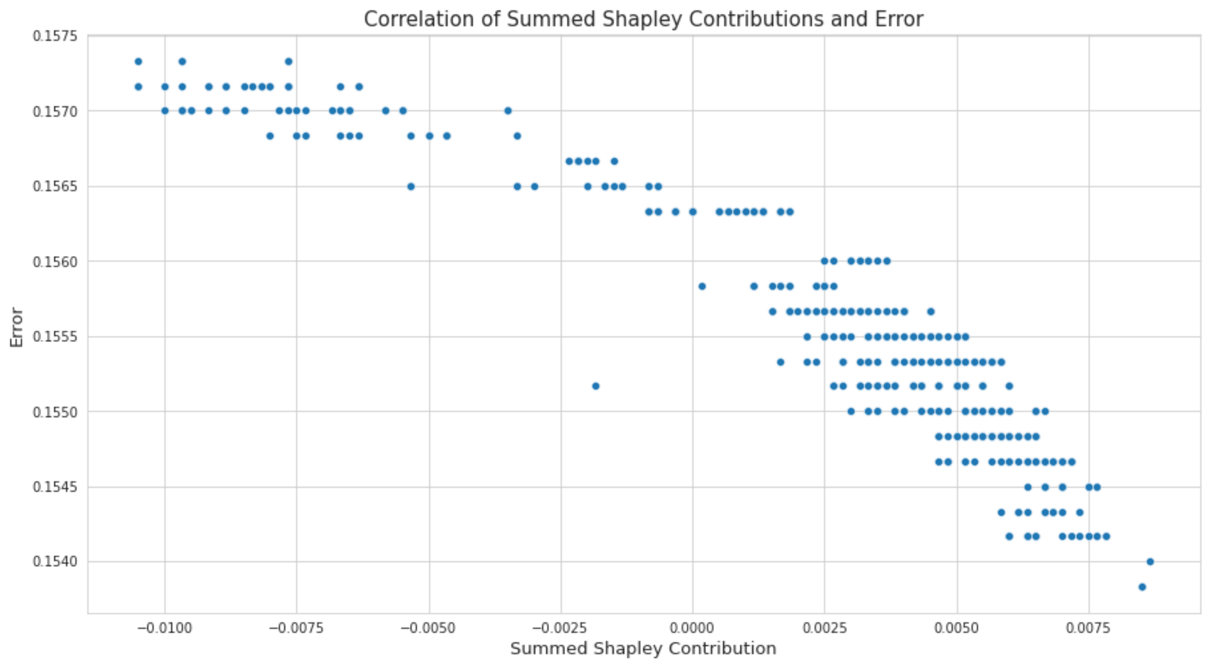
The first notion we discovered is rather trivial yet useful to understanding the Shapley contributions of each group in an ordering; given some permutation σ, it is not necessarily true that $\forall i, \varepsilon_{f_{\sigma^{-i}}} > \varepsilon_{f_\sigma}$. Less formally, this means there may be subgroup-hypothesis pairs who hurt the overall error rather than helping given when they are introduced in an ordering. We provide an example of this in the notebook where introducing the pair (g_4, h_4) in the ordering $\sigma = [2, 1, 3, 6, 5, 4, 7, 8]$ actually increases the error compared to the ordering $\sigma^{-4} = [2, 1, 3, 6, 5, 7, 8]$. While not necessarily consequential on its own, this caused us to examine the Shapley contributions of the optimal and worst permutations found by *best_permutation.py* to search for correlations. When we computed the *fractional_ shapley_ contribution* function over the best and worst permutations, we received the following:
We note that in the best permutation, nearly all contributions are positive, and in the worst permutation,

Table 1: Shapley Contributions of Best/Worst Permutations

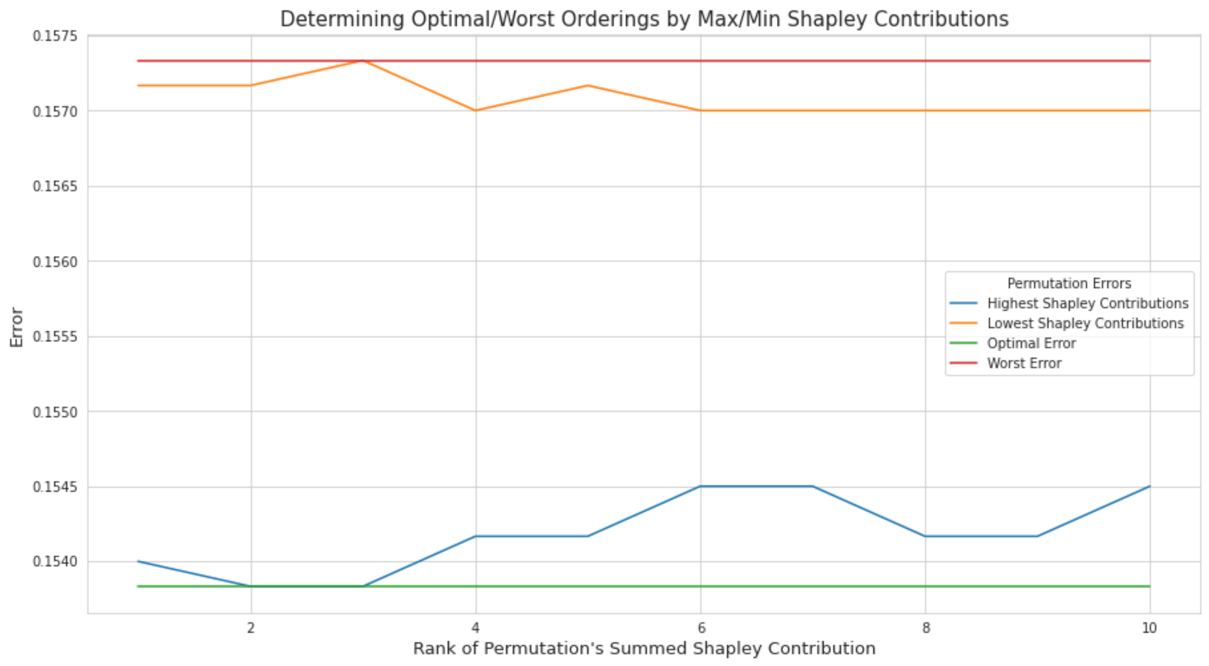
Group	Best permutation	Worst permutation
1	0.000167	-0.002167
2	0.001500	-0.002667
3	0.000500	-0.002000
4	0.000833	-0.001667
5	0.000500	0.000500
6	0.002500	0.000000
7	0.000667	0.000000
8	0.001833	-0.001667

nearly all contributions are negative or zero. Intuitively, it makes sense that a 'good' permutation is one where pairs are 'rewarded' rather than 'punished' considering the reward is related to the error of the overall model. Thus, we wondered how correlated the sum of each pair's Shapley contributions in a given permutation is with the overall error of the PDL created using the ordering from that permutation.

Unfortunately, the data collected from *Shapley Values.ipynb* only contained the Shapley contributions but not the permutation. We collected another 650 samples using the same methodology from *Shapley Values.ipynb* which also contained the permutation associated with each contribution, and computed the error of each PDL on these permutations. Each point on the graph below is connected to a permutation where the y-axis is the overall error of the PDL created from the permutation and the x-axis is the sum of each pairs Shapley contribution from the permutation.



We note the decreasing error tendency as the summed Shapley contributions increase. This grants a method to determining which permutations are closest to the optimal and worst permutations. Using the top ten permutations for both largest and smallest summed Shapley contributions, we plot the errors of the PDLs created on these permutations with optimal and worst error reference lines below:



We note that while top ten best/worst Shapley contribution permutations did not lead perfectly to our optimal/worst error, we were able to obtain rough approximations using an extremely small sample of all Shapley contributions (∼1.5%). This result scales well for larger sets of groups as it is not required to use the expected Shapley value for individual pairs, rather it uses the sum of Shapley contributions on specific permutations.

This changes the overall question from searching for a random permutation which produces an optimal solution to finding the permutation which maximizes the summed Shapley contribution of an individual permutation from the permutation set Σ. The expected Shapley value of a pair may prove useful for solving this optimization problem, though this question remains open for future research.

4.2 Additional Minor Result

Another result stemmed from a question we wrote for potential future research and solved along the way indirectly. We set a scenario where a team submits a PDL f_σ which accepts all subgroup-hypothesis pairs $(g_1, h_1, \ldots, g_n, h_n)$ with ordering σ. We wanted to know does $\exists \sigma' \neq \sigma$ ordering where all pairs are accepted such that $\varepsilon_{f_{\sigma'}}(X) < \varepsilon_{f_\sigma}(X)$? Explicitly, if n pairs are all accepted as updates in one ordering, is this ordering optimal when the universe is restricted to these pairs.

We inadvertently ran into an instance where all groups were accepted during an iterative update, but the final error on the model was greater than that of our optimal error. Examining the Shapley contribution of this permutation, we noted that one of the pairs contribution was 0, implying the final PDL was equivalent to a PDL using the same ordering without the aforementioned pair. This intuitively makes sense that a pair could be overwritten, but it stems to a new question of finding an optimal ordering given the knowledge that all groups are accepted in some ordering.

We wrote a simple algorithm in the *Shapley Values.ipynb* notebook which takes in an initial permutation and computes the Shapley contributions on this permutation. Given the contributions, a new permutation is created by forcing all 'bad' updates (pairs whose contributions are less than zero) to the end positions of the permutation, and any updates who are generally ignored (pairs whose contributions equal zero) are moved to the front of the permutation. All other pairs remain in their relative position, and we recompute the Shapley contribution of the new permutation. We continue this process until there all contributions are positive. We do not include the algorithm for the code in this paper but encourage the reader to examine the example in the notebook.

4.3 Future Research Question

Suppose there is a universe $U = \{(g_1, h_1), \ldots, (g_n, h_n))\}$ of subgroup-hypothesis pairs. Let σ_w be a worst ordering such that $\forall \sigma \in \Sigma, \varepsilon_{f_\sigma}(X) \leq \varepsilon_{f_{\sigma_w}}(X)$. Let σ_{opt} be an optimal ordering such that $\forall \sigma \in \Sigma, \varepsilon_{f_{\sigma_{opt}}}(X) \leq \varepsilon_{f_\sigma}(X)$. Can we find a good upper bound L such that $\varepsilon_{f_{\sigma_w}}(X) - \varepsilon_{f_{\sigma_{opt}}}(X) \leq L$? More practically, can we use approximate Shapley Values to find a worst ordering, and thereby know how far from an optimal ordering we are?

5 Papers Referenced

References

Bonanno, G. (2022, Cooperative Games). *Economics 122: Game theory - lecture notes: The shapley value.* University of California Davis.

Chandrasekaran, R. (2021). *Computer science 6301: Game theory - lecture notes: Cooperative games.* The University of Texas at Dallas.

Globus-Harris, I., Kearns, M., & Roth, A. (2022). Beyond the frontier: Fairness without accuracy loss. *CoRR*, *abs/2201.10408*. Retrieved from https://arxiv.org/abs/2201.10408

Leyton-Brown, K. (2022, Lecture 23). *Computer science 532: Modeling human strategic behavior - lecture notes: The shapley value and the core.* University of British Columbia.

Shapley, L., Roth, A., & Press, C. U. (1988). *The shapley value: essays in honor of lloyd s. shapley.* Cambridge University Press. Retrieved from https://books.google.com/books?id=JK7MKu2A9cIC