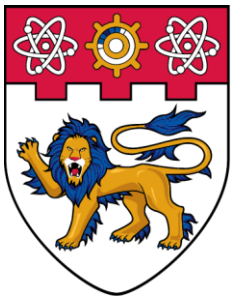

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Time-Efficient Algorithms to Approximate the Shapley Values for Horizontal Enterprise Federated Learning

Le Quang Anh

School of Computer Science and Engineering

2020

Abstract

As Federated Learning becomes the new paradigm for private and secure machine learning, a fundamental challenge is how to distribute the profit earned by the federated model among data contributors. A fair incentive mechanism utilising the Shapley Values (SV) has been proposed but calculating the SV remains a tricky task. Determining the exact SV involves training and evaluating a large number of federated models that rises exponentially with the number of data contributors, and hence is impractical in real scenarios. There is a need for an efficient approximation algorithm for SV. This project aims to refine and combine the existing algorithms for approximation of the SV of data contributors in the Federated Learning setting, namely One Round Model Reconstruction (OR) and Truncated Monte Carlo Shapley (TMC). The project manages to streamline OR into Adjusted OR and further improve it by incorporating the TMC idea. The new proposed algorithm, Adjusted OR-TMC, has shorter runtime while not sacrificing on the accuracy of OR, which makes it ideal as a replacement for OR. The project discusses the performance results and the reasons behind them.

Acknowledgements

First of all, I would like to thank Dr Liu Zelei for being a thoughtful mentor who guided me throughout this research journey. His guidance was integral to the implementation of the algorithms that are key to the project. I would like to thank Prof Yu Han and CNYSP for providing me with an opportunity to study such a fascinating and cutting-edge area of computer science. I am also grateful to my friend Lin Lejia for her precious collaboration for the early parts of the project, especially in setting up the datasets and the environment. Lastly, I would like to thank my friend Pye Sone Kyaw for his tremendously helpful advice on machine learning as well as moral support.

Table of Contents

List of Figures

List of Tables

Nomenclature

n	Number of data contributors
N	The set of indices of data contributors
i	Index of a data contributor
D_i	Data set of data contributor with index i
T	Number of global iterations for a federated learning setting. Also symbolise the standard test set in certain context
β	Minibatch size
E	Number of local epochs for a federated learning setting
M	Machine learning model
$U(M, T)$	Performance score of model M on standard test set T
M_S^T	Federated model trained on a subset S of clients for T iterations
$\widetilde{M_S^T}$	Approximation of M_S^T by the OR algorithm
ϕ_i	Contribution index of data contributor i. It is chosen to be equal to the Shapley Value of that data contributor.

Chapter 1 Introduction

1.1 Motivation for Research

Federated learning (FL) is a machine learning setting that helps to address the current privacy concerns over contributing data to a central database. In the FL setting, data contributors (clients) train their own local models and only send local updates, instead of raw data, to the main server. The global model at the server is then trained based on these local updates [1].

In order to fully apply FL among multiple organisations, an incentive mechanism should be built, such that the benefits can be shared fairly among the contributors, which will encourage more organisations to participate [2]. A game-theoretic viewpoint of this problem identifies the Shapley Values (SV) as a fair profit distribution scheme for this form of cooperation [3]. However, exact SV calculation is exponentially complex, and current SV approximation algorithms still incur huge time costs. To make SV the standard measure of values for data, we need to find more efficient SV approximation algorithms. This work aims to explore the current approximation algorithms and develop a more efficient algorithm.

1.2 Objectives

Using the idea from the Truncated Monte Carlo Shapley algorithm (TMC-Shapley) [4], this project aims to improve upon One-Round Model Reconstruct (OR), a cutting-edge SV approximation algorithm for Federated Learning [5]. The performance of the new proposed algorithms will be assessed by comparing against the original OR algorithm in terms of time and accuracy.

The implementation of the new combined algorithm is expected to have a

significant reduction in runtime with slightly larger deviation from the exact calculated Shapley Values as compared to the original OR algorithm.

1.3 Scope of Work

The project investigates the concept of Federated Learning, with a focus on horizontal enterprise supervised FL, where a small number of data contributors have big datasets with the same attributes. It also studies the problem of profit distribution and reviews works on the Shapley Values in profit distribution as well as current algorithms for approximating the SV.

The next part of the project focuses on refining and combining the current SV approximation algorithms available, namely TMC-Shapley and OR, to design new algorithms. The proposed algorithms are implemented and assessed on two important criteria, namely time and accuracy, and compared with other algorithms. The results of the experiments are analysed and thoroughly discussed.

An FL testbed will be constructed to apply the concept of the SV in the context of horizontal enterprise supervised FL. The PyTorch package and the Python programming language will be used as these are common tools in FL research. A GPU-enabled environment will also be utilised for faster calculations. The dataset used will be MNIST dataset, a standard publicly available dataset for machine learning that allows researchers to test image classification methods while spending minimal efforts in formatting [6].

1.4 Organisation of Report

The report is organised as follows:

Chapter 1, which is this chapter, presents an introduction to the project. It includes the research motivation, objectives and scope, as well as the organisation of the report.

Chapter 2 summarises findings from the literature review of federated machine learning, Shapley Value and OR.

Chapter 3 explains the design of the new combined algorithm.

Chapter 4 shows the experimental setup with the data pre-processing and the environment on which the experiments are conducted.

Chapter 5 presents results from the experiments.

Chapter 6 concludes and discusses future work on the topic.

Chapter 2 Literature Review

2.1 Related Work

2.1.1 Federated Learning – Context and Categorisation

Training a machine learning model requires huge amount of data, which may originate from different sources. However, in the real world, these sources of data are often isolated [2]. Due to growing concerns about data security and personal privacy, stricter regulations on data sharing and storage have been implemented, such as European Union's General Data Protection Regulation [7] and China's Cybersecurity Law [8]. These regulations further increase the barrier to data integration and hence hinder the applications of machine learning to solve problems in several fields.

Federated learning (FL) is a possible approach to deal with these challenges. In the traditional machine learning setting, the server will collect all data from clients, and train a model on the accumulated data [2]. There are great risks and responsibilities involved in this setting, which may not be in accordance with the current data regulations. On the other hand, in the FL setting, each data contributor has a local training dataset which is never collected by the server. Instead, each client computes a local update to the current global model in the server and sends this local update to the server. The server then aggregates all the local updates to make a new global model, which is then sent back to the contributors as "updated global model". Since these local updates are specific to enhancing the current global model, the updates can be discarded once they have been applied [9]. FL approach hence eliminates the need for collection and storage of raw data. It allows enterprises to collaborate on building a machine learning model without having to share their raw data which may be sensitive.

Yang *et al.* categorises FL into three main types based on the characteristics of the datasets [2]. Horizontal FL refers to the scenario that datasets of different data contributors refer to different samples but have the same features. For example, two banks may have data of different bankers, but these data have the same features such as amount of money, date of deposit, and so on. In vertical FL scenario, datasets of different contributors refer to the same samples (objects, people or phenomena) but have different features. An example is a bank and a telecommunication company working in the same area. They may have data about the same user Joe, but on different aspects of his life: the bank may have Joe's amount of bank deposit, while the telecommunication company may have data about the durations of his calls. The third type of FL is Federated Transfer Learning, whereby datasets differ in both features and samples.

This project focuses on horizontal enterprise FL. "Enterprise" means there is a small number of data contributors, but they are organisations like businesses and enterprises that each has a lot of data [5].

2.1.2 Federated Learning Framework

The FL framework presented below is taken from Song et al. [5], following the framework first proposed by McMahan et al. [9].

A simple illustration of the FL setting is shown in Figure 2-1.

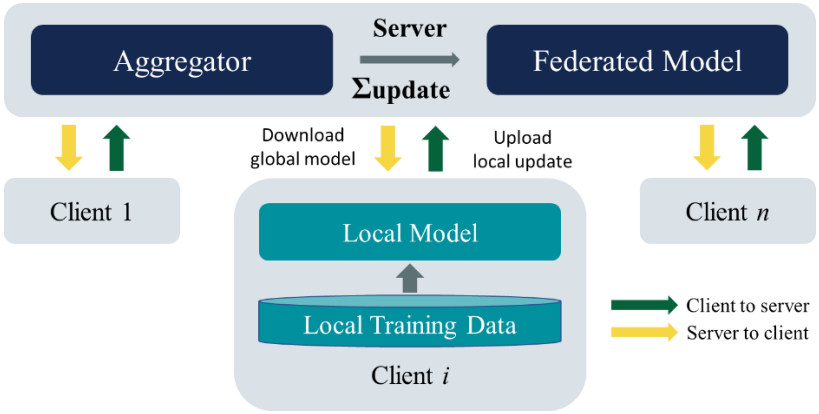

Figure 2-1: Federated Learning workflow.
Source: Adapted from [5]**.**

Suppose there are n clients (data contributors), each possessing a dataset D_i where $i \in N = 1,2,\dots,n$. For each global iteration $t \in 0,1,\dots,T-1$, three steps are involved:

- Step 1: The server sends a global model M^t to all the clients, as shown by the yellow arrows in Figure 2-1.
- Step 2: Each client, take client i as an example, trains M^t based on their local training data D_i and return the updated sub-model (local update) M_i^t to the server. This is represented by the green arrows.
- Step 3: The server aggregates all the sub-models $\{M_i^t \mid i \in N\}$ to form a new global model M^{t+1} for the next global iteration.

The Federated Average algorithm [9] explains the process of aggregation (step 3 above) as calculating the weighted average of all the local updates.

$$\boldsymbol{M^{t+1} = \sum_{i=1}^{n} \frac{|D_i|}{\sum_{i=1}^{n}|D_i|} \cdot M_i^t}$$

Equation 1

McMahan et al. [9] explores a scenario where there are many clients (100 in total) who are mobile phone users and may not connect to the server on every

iteration, and hence only a fraction of the clients will receive a global model in step 1 and perform training on their local datasets step 2, and the global model is updated from the local updates from that fraction in step 3. My project assumes all clients participate for all global iterations since they are enterprises which are actively collaborating on building a joint model. Hence the three steps exactly as presented above will be used.

2.1.3 Distribution of Profit and the Shapley Value

As with traditional machine learning, achieving a high performance for an FL model requires large amount of high-quality data. This may mean encouraging more organisations with high-quality data to join the data federation. Data contributors incur a cost for obtaining and processing of data, training and sending of local updates to the server. Getting access to the final global model may be considered as a form of compensation, and an incentive to join the federation. However, in the case that the trained global model is paid and used by parties outside the federation, data contributors would expect to be compensated parts of the profit. Moreover, from a game-theoretic viewpoint, giving incentives has been identified to be the most viable action to collect quality data. Assuming that all clients are rational and cooperative, a reward system will help to increase both the quantity and quality of data [10].

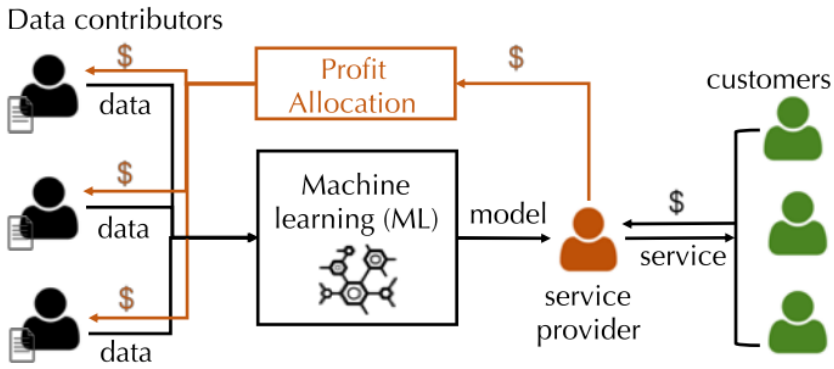

Figure 2-2: The data valuation problem.
Source: [3].

This brings to the fore the challenge of distributing profits fairly among the contributors. How much each data contributor receives will be based on the value of the contribution (Figure 2-2). For the same setting presented in 2.1.2 and with reference to Song et al. [5] and Ghorbani and Zou [4] we can define the Contribution Index (CI) as shown below.

Definition: Suppose there are *n* data contributors, each possessing a dataset D_i where $i \in N = 1,2,\dots,n$. N is the set of indices of all data contributors. Suppose there is a learning algorithm *A* and a standard test set *T*. For any subset of contributors, *S* of all the contributors, *N*, D_S is the union of data from this subset of contributors. Let $M_S(A)$ be the model built by training D_S on algorithm *A*, abbreviated as M_S. The performance score *U* of a model *M* on the standard test set *T* is denoted by *U(M, T)* or just *U(M)* for simplicity. *U* is also a black box that takes in a model and outputs a score. Let $\phi(A, D_N, T, D_i)$, or ϕ_i for short, be the CI of D_i in the context of D_N, *A* and *T*.

To ensure a fair distribution, the contribution index is expected to have the following desirable properties [3]–[5].

- Group rationality: The value of the entire federation's datasets is completely distributed among the data contributors, i.e. $U(M_N) = U(M_\emptyset) + \sum_i^n U(M_i)$. $U(M_\emptyset)$ is the value of a randomly initialized model.
- Symmetry: if two datasets D_i and D_j are identical with respect to what they contribute to the performance score under algorithm *A* on a test set *T*, their contribution indices should be equal. That is, if $U(M_{S\cup\{i\}}) = U(M_{S\cup\{j\}}), \forall S \subseteq N\backslash\{i,j\}$, then $\phi_i = \phi_j$.
- Dummy: if a dataset D_j does not affect the performance of a model trained under algorithm *A* on a test set *T*, its contribution index should be zero. That is, $\phi_i = 0$ if $U(M_{S\cup\{i\}}) = U(M_S),\ \forall S \subseteq N\backslash\{i\}$.

- Additivity: if D_i contributes values $\phi_i(T_1)$ and $\phi_i(T_2)$ to the predictions of test points 1 and 2 in the test set *T*, then the value of D_i in predicting both test points is $\phi_i(T_1) + \phi_i(T_1)$.

Any function satisfying the above properties has to be of the form of the Shapley Value (SV) [11]. SV is a concept in cooperative game theory to distribute the total benefits from a coalition of players. The Contribution Index shall hence be called the SV, with the formula as follows [3]–[5], [11]. Note that $|S|$ represents the cardinality of the subset S.

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{U\left(M_{S \cup \{i\}}\right) - U(M_S)}{n \cdot \binom{n-1}{|S|}}$$

Equation 2

The right-hand side of Equation 2 can be understood as the expected marginal contribution of the dataset D_i [12]. It is computed by considering all the possible orders of "arrival" of the datasets into the federation and averaging the marginal contributions across all these cases. The numerator is the marginal contribution of D_i when it arrives after S and denominator accounts for the probability of that happening.

2.1.4 Algorithms to Calculate the Shapley Value

Computational cost is a challenge in adopting SV to distribute the profits [3]. Calculating the exact SV using Equation 2 requires computing the performance of all 2^n models M_S where $S \subseteq N$. That involves training (2^n-1) additional federated models. This is computationally and communicationally expensive – the data contributors must compute and send the local updates for the training of the "extra" models – all models other than M_N. Adopting an algorithm that can approximate the SV efficiently is a more practical option.

Ghorbani and Zou suggested that the SV value calculations does not have to be 100% accurate. As the test set *T* is finite, *U(M)* only approximates the true performance of the trained model on the test distribution. Therefore, it is

enough to evaluate SV value up to the intrinsic noise in *U(M)*, which can be quantified by measuring variation in the performance of the same model across bootstrap samples of the test set. They propose two algorithms to approximate SV value of individual data points for the traditional machine learning (TML) setting. Truncated Monte Carlo Shapley (TMC-Shapley) is the first algorithm, which involves repeatedly generating permutations of "arrivals" the data points, calculating the marginal contributions of each data point for each permutation, and then taking the average across all permutations as the estimated SV when they converge. For each permutation, since the marginal contribution of each data point decreases as more data arrives, truncation – assigning zero marginal contribution to the rest of the data when a certain threshold is reached – can be applied to reduce computations. They noted that $3n$ Monte Carlo samples is sufficient for convergence, which means less than $3n^2$ models need to be trained to calculate the SV. The second algorithm suggested does not extend well to the FL setting.

Jia et al. [3] also explore the SV in TML setting, and develop a Group Testing-Based (GTB) approach to estimate SV. The idea behind this approach is to repeatedly draw a random set of "users" – the data contributors in the TML setting – and evaluate the performance score of a model trained on data from the selected set of users.

Unlike the two papers above that focuses on reducing the number of extra models to train and evaluate, Song et al. [5] works on the FL setting and develops two new algorithms that aim to reduce the time and number of communication rounds taken to build each of the extra models $M_S, \forall S \subseteq N$ (note the strict subset symbol, $\subset$) by approximating them instead of training them separately. One-Round Model Reconstruction (OR) and Multi-Round Model Reconstruction (MR) approximate the extra models, using the local updates while training M_N, hence reducing the time and communication for

additional training. The main difference between these two approaches is that, while OR approximates models through all the global iterations and only evaluates them to find SV afterwards, MR approximates and evaluates models for every global iteration and calculate the marginal contribution for each global iteration. This makes MR more computationally expensive than OR. In fact, Song et al. compares OR and MR with federated TMC-Shapley and federated GTB and concludes that OR is the most time-efficient. OR is slightly more accurate than MR for non-noised data, whereas MR is the most accurate for data with noised labels or noised features.

2.2 Research Gaps

While OR can calculate SV in shorter time than MR and is even significantly faster than federated TMC-Shapley and federated GTB, the model reconstruction and evaluation steps, which involve an additional $(2^n - 1)$ models, are still exponentially complex. This may make OR slow when the number of data contributors n in the federation grows, when the model is more complex, or when the test set is much larger. A combination of federated TMC-Shapley and OR may potentially reduce time taken while not sacrificing too much on the accuracy. My work aims to assess whether this is a possible direction to improve on OR.

Chapter 3 Research Methodology

3.1 Algorithms in Detail

The independent parameters for this research are the different algorithms for calculating the SV. These algorithms utilise Federated Average (FedAvg) and Minibatch Stochastic Gradient Descent for a supervised learning task. Since the ClientUpdate (steps 12-17 in Figure 3-1 below) section is the same for all algorithms, it will only be presented once in Exact Federated Shapley.

3.1.1 Exact Federated Shapley

This approach calculates the SV of data contributors using Equation 2. It trains federated models based on the different subsets S of contributors and these models are evaluated on the standard test set. Figure 3-1 shows the details.

Algorithm 1 Exact Federated Shapley (FedAvg).
B is the local minibatch size, n is the number of participants, T is the number of global iterations, E is the number of local epochs and η is the learning rate.

Server executes:

initialise $\{M_S^0\}$, where $S \subseteq N = \{1, 2, ..., n\}$;
for each subset $S \subseteq N$ **do**
  for each round $t = 0, 1, 2, ..., T-1$ **do**
    for each client $i \in S$ **in parallel do**
      Send M^t to all n clients;
      $M_{S,i}^{t+1} \leftarrow \text{ClientUpdate}(i, M^t)$
      $\Delta_{S,i}^{t+1} \leftarrow M_{S,i}^t - M_S^t$
    end for
    $M_S^{t+1} \leftarrow M_S^t + \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S} |D_i|} \cdot \Delta_{S,i}^{t+1}$
  end for
end for
for $i = 1, 2, ..., n$ **do**
  $\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{n \cdot \binom{n-1}{|S|}} [U(M_{S \cup \{i\}}^t) - U(M_S^t)]$;
end for
return M^T and $\phi_1, \phi_2, ..., \phi_n$;

ClientUpdate(i, M)**:**

for each local epoch $e = 1, 2, ..., E$ **do**
  for batch $b \in B$ **do**
    $M \leftarrow M - \eta \nabla L(M; b)$
  end for
end for
return M to server

Figure 3-1: Exact Federated Shapley algorithm.

3.1.2 One-Round Model Reconstruction (OR)

The idea behind OR (Figure 3-2) is to approximate the models M_S, using the local updates from training the main federated model M_N [5]. This means that the data contributors do not have to perform more computations nor communicate more with the server just for the training of extra models. Note that the ClientUpdate section of the algorithm is omitted since it is the same as Exact Federated Shapley.

Algorithm 2 OR

B is the local minibatch size, n is the number of participants, T is the number of global iterations, E is the number of local epochs and η is the learning rate.

Server executes:

1: initialise M^0, $\{\widetilde{M}_S^0\}$, where $S \subseteq N = \{1, 2, ..., n\}$;
2: **for** each round $t = 0, 1, 2, ..., T-1$ **do**
3: Send M^t to all n clients;
4: $M_i^t \leftarrow \text{ClientUpdate}(i, M^t)$;
5: $\Delta_i^{t+1} \leftarrow M_i^t - M^t$;
6: $M^{t+1} \leftarrow M^t + \sum_{i=1}^{n} \frac{|D_i|}{\sum_{i=1}^{n}|D_i|} \cdot \Delta_i^{t+1}$;
7: **for** each subset $S \subseteq N$ **do**
8: $\widetilde{M}_S^{t+1} \leftarrow \widetilde{M}_S^t + \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S}|D_i|} \cdot \Delta_i^{t+1}$;
9: **end for**
10: **end for**
11: **for** $i = 1, 2, ..., n$ **do**
12: $\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{n \cdot \binom{n-1}{|S|}} [U(\widetilde{M}_{S \cup \{i\}}^t) - U(\widetilde{M}_S^t)]$;
13: **end for**
14: **return** M^T and $\phi_1, \phi_2, ..., \phi_n$;

Figure 3-2: OR algorithm.

Note that $\widetilde{M}_S^t$ (Figure 3-2) is an approximation of the federated model M_S^t (Figure 3-1) trained with a subset of users *S*. Step 8 above is the approximation step, where the $\widetilde{M}_S^t$ is updated with gradients Δ_i^{t+1} that are calculated with respect to M^t. Mathematically, $\widetilde{M}_N^t$ is the same as M^t, and $M_\emptyset^t$ is never updated and is the randomly initialized model, hence the loop in steps 7-8 only needs (2^n-2) iterations.

Even so, this means performing (2^n-2) extra updates per global iteration. If the number of global iterations is large, the computational cost will be high. However, upon further inspection of the mathematics behind this algorithm, it is clear that:

$$\widetilde{M}_S^t = \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S}|D_i|} \cdot \widetilde{M}_{\{i\}}^t$$

With this idea, the algorithm is adjusted as shown in Figure 3-3. Instead of updating all the approximated model $\widetilde{M}_S^t$ every global iteration, we can choose

to only update *n* models $\widetilde{M}_{\{i\}}^{t}$ and calculate $\widetilde{M_S^T}$ directly as a weighted average at the end (steps 11-12).

Algorithm 3 Adjusted OR

B is the local minibatch size, n is the number of participants, T is the number of global iterations, E is the number of local epochs and η is the learning rate.

Server executes:

1: initialise M^0, $\{\widetilde{M}_S^0\}$, where $S \subseteq N = \{1, 2, ..., n\}$;
2: **for** each round $t = 0, 1, 2, ..., T-1$ **do**
3: Send M^t to all n clients;
4: $M_i^t \leftarrow \text{ClientUpdate}(i, M^t)$;
5: $\Delta_i^{t+1} \leftarrow M_i^t - M^t$;
6: $M^{t+1} \leftarrow M^t + \sum_{i=1}^{n} \frac{|D_i|}{\sum_{i=1}^{n} |D_i|} \cdot \Delta_i^{t+1}$;
7: **for** each $i \in N = \{1, 2, ..., n\}$ **do**
8: $\widetilde{M}_{\{i\}}^{t+1} \leftarrow \widetilde{M}_{\{i\}}^{t} + \Delta_i^{t+1}$;
9: **end for**
10: **end for**
11: **for** each subset $S \subseteq N$ **do**
12: $\widetilde{M}_S^T \leftarrow \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S} |D_i|} \cdot \widetilde{M}_{\{i\}}^T$;
13: **end for**
14: **for** $i = 1, 2, ..., n$ **do**
15: $\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{n \cdot \binom{n-1}{|S|}} [U(\widetilde{M}_{S \cup \{i\}}^t) - U(\widetilde{M}_S^t)]$;
16: **end for**
17: **return** M^T and $\phi_1, \phi_2, ..., \phi_n$;

Figure 3-3: Adjusted OR algorithm.

Both OR and Adjusted OR are expected to have low accuracy for noised data. With reference to [5], OR has the lowest accuracy (compared to TMC-Shapley, GTB, and MR) when there are different levels of noise in features or in labels of data. A plausible explanation is that the model reconstruction of $\widetilde{M_S^t}$ (steps 7-8 in Figure 3-2 and steps 7-8 and 11-12 in Figure 3-3) is approximated using local updates from training the main federated model M_N. These approximations for noised cases are worse than non-noised cases because M_N may have much more different gradients from M_S due to the different levels of noise across the datasets D_i.

3.1.3 OR-TMC Combinations

Evaluation of a model incurs a huge cost in terms of time, especially if the test set is large. With the current OR algorithm, we must reconstruct and evaluate 2^n models. The idea of Truncated Monte Carlo can be used to reduce the computation cost involved in steps 11-15 of the Adjusted OR algorithm (Figure 3-3). Figure 3-4 shows the OR-TMC algorithm.

Steps 1-10 are taken from the Adjusted OR algorithm. The TMC idea is incorporated in steps 11-26. First, we sample a random permutation of datasets (step 14). After that, we scan from the first dataset to the last dataset and calculate the marginal contribution of every new dataset. By repeating this process over multiple permutations, the approximation of SV is the average of all the calculated marginal contributions [4]. The while loop is run until certain convergence criteria are met. In this project we stop the loop when the average percentage change in ϕ_i after a TMC iteration m is less than a certain amount (termed "TMC error").

The truncation process is presented in steps 17-18. As the size of subset S increases, the marginal contribution of the next coming dataset becomes smaller. Hence, when the performance score v_{j-1}^{m} is within a certain distance (termed "performance tolerance") from $U(M_N)$, we can assign 0 to the rest of the coming datasets, for that particular permutation. The performance tolerance chosen is 1% of $U(M_N)$. Note that all the $U\left(\widetilde{M_S^T}\right)$ calculated will be saved in between the permutations so that a value can be quickly retrieved in step 22 if the next permutations happen to have the same *S* in step 20.

Algorithm 4 OR-TMC

B is the local minibatch size, n is the number of participants, T is the number of global iterations, E is the number of local epochs and η is the learning rate.

Server executes:

initialise M^0, $\{\widetilde{M}_S^0\}$, where $S \subseteq N = \{1, 2, ..., n\}$;
for each round $t = 0, 1, 2, ..., T-1$ **do**
  Send M^t to all n clients;
  $M_i^t \leftarrow \text{ClientUpdate}(i, M^t)$;
  $\Delta_i^{t+1} \leftarrow M_i^t - M^t$;
  $M^{t+1} \leftarrow M^t + \sum_{i=1}^{n} \frac{|D_i|}{\sum_{i=1}^{n} |D_i|} \cdot \Delta_i^{t+1}$;
  for each $i \in N = \{1, 2, ..., n\}$ **do**
    $\widetilde{M}_{\{i\}}^{t+1} \leftarrow \widetilde{M}_{\{i\}}^{t} + \Delta_i^{t+1}$;
  end for
end for
initialise $m = 0$
while Convergence criteria not met **do**
  $m = m + 1$
  π^m: Random permutation of datasets D_i
  $v_0^m \leftarrow U(\widetilde{M}_{\emptyset}^0)$
  for $j \in \{1, 2, ..., n\}$ **do**
    if $|U(M_N) - v_{j-1}^m|$ <Performance tolerance **then**
      $v_j^m = v_{j-1}^m$
    else
      $S \leftarrow \{\pi^m[1], \pi^m[2], ..., \pi^m[j]\}$
      $\widetilde{M}_S^T \leftarrow \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S} |D_i|} \cdot \widetilde{M}_{\{i\}}^T$;
      $v_j^m \leftarrow U(\widetilde{M}_S^T)$
    end if
    $\phi_{\pi^m[j]} \leftarrow \frac{m-1}{m} \phi_{\pi_m[j]} + \frac{1}{m}(v_j^m - v_{j-1}^m)$
  end for
end while
return M^T and $\phi_1, \phi_2, ..., \phi_n$;

Figure 3-4: OR-TMC algorithm.

From [5], TMC-Shapley takes a very long time to converge and has low accuracy. This can be because the marginal contribution of the first dataset for each permutation, v_1^t, is large. Ghorbani and Zou [4] achieve convergence after only $3n$ permutations, perhaps because they are calculating SV for individual data points, which are much smaller than the SV for datasets as in our project. A slight modification can be made to ensure that OR-TMC converges faster, by ensuring that each dataset has equal chance to be the first element of the permutation. Figure 3-5 shows how this is done.

Algorithm 5 Adjusted OR-TMC

B is the local minibatch size, n is the number of participants, T is the number of global iterations, E is the number of local epochs and η is the learning rate.

Server executes:

initialise M^0, $\{\widetilde{M}_S^0\}$, where $S \subseteq N = \{1, 2, ..., n\}$;
for each round $t = 0, 1, 2, ..., T-1$ **do**
  Send M^t to all n clients;
  $M_i^t \leftarrow \text{ClientUpdate}(i, M^t)$;
  $\Delta_i^{t+1} \leftarrow M_i^t - M^t$;
  $M^{t+1} \leftarrow M^t + \sum_{i=1}^{n} \frac{|D_i|}{\sum_{i=1}^{n} |D_i|} \cdot \Delta_i^{t+1}$;
  for each $i \in N = \{1, 2, ..., n\}$ **do**
    $\widetilde{M}_{\{i\}}^{t+1} \leftarrow \widetilde{M}_{\{i\}}^t + \Delta_i^{t+1}$;
  end for
end for
initialise $m = 0$
while Convergence criteria not met **do**
  $m = m + 1$
  for $k \in N = \{1, 2, ..., n\}$ **do**
    $\pi^{m,k}$: Random permutation of datasets D_i, the first element must be k
    $v_0^{m,k} \leftarrow U(\widetilde{M}_\emptyset^0)$
    for $j \in \{1, 2, ..., n\}$ **do**
      if $|U(M_N) - v_{j-1}^{m,k}|$ <Performance tolerance **then**
        $v_j^{m,k} = v_{j-1}^{m,k}$
      else
        $S \leftarrow \{\pi^{m,k}[1], \pi^{m,k}[2], ..., \pi^{m,k}[j]\}$
        $\widetilde{M}_S^T \leftarrow \sum_{i \in S} \frac{|D_i|}{\sum_{i \in S} |D_i|} \cdot \widetilde{M}_{\{i\}}^T$;
        $v_j^{m,k} \leftarrow U(\widetilde{M}_S^T)$
      end if
      $\phi_{\pi^{m,k}[j]} \leftarrow \frac{n \cdot (m-1) + k - 1}{n \cdot (m-1) + k} \phi_{\pi_m[j]} + \frac{1}{n \cdot (m-1) + k} (v_j^{m,k} - v_{j-1}^{m,k})$
    end for
  end for
end while
return M^T and $\phi_1, \phi_2, ..., \phi_n$;

Figure 3-5: Adjusted OR-TMC algorithm.

Each TMC iteration *m* now contains *n* permutations instead of 1 permutation as in Figure 3-4. The convergence criteria and the performance tolerance are kept identical to those of OR-TMC. The number of TMC permutations needed for Adjusted OR-TMC's convergence can be calculated by multiplying the number of TMC iterations with n.

3.2 Evaluation Metrics

The following two parameters are recorded for comparison:

- **Time:** Only the time spent for calculation of the SV are compared. This is the total running time excluding the training of the main model M_N and the writing of data into files.
- **Shapley Values:** the performance score function *U* is chosen to be the accuracy function – the percentage of test cases the model *M* manages to predict correctly. The SV are then calculated according to the different algorithms.

For comparison of the accuracy of the SV, all SV calculated are first "standardised" by scaling them by a common factor such that $\sum_{i=1}^{n} \phi_i = 1$. This is appropriate because profit distribution will likely be based on the percentage contribution. Let the standardised SV vectors generated by the Exact Federated Shapley be denoted by $\phi^* =< \phi_1^*, \phi_2^*, \dots, \phi_n^* >$ and by the approximation algorithms $\phi =< \phi_1, \phi_2, \dots, \phi_n >$. The **Euclidean Distance** is defined as below [5].

$$D_E = \sqrt{\sum_{i=1}^{n} (\phi_i^* - \phi_i)^2}$$

The Average Euclidean Distance and the Maximum Euclidean Distance across different repeated runs of the same algorithm will be used as comparison with other algorithms. The lower the Average Euclidean Distance and the Maximum Euclidean Distance, the more accurate that approximation algorithm is.

3.3 Controlled Parameters

Other parameters for federated learning are kept constant across all the algorithms. The federated model used is a 2-layer fully connected Multilayer Perceptron, with ReLU as the activation function. Some examples of controlled parameters are shown in Table 3-1.

Table 3-1: Some controlled parameters.

Parameter	Symbol	Value
Number of local epochs	E	10
Minibatch size	B	64
Learning rate	η	0.01
Communication cost		0

Aside from Exact Federated Shapley, communication cost is the same for all the approximation algorithms shown, since there is no extra communication other than that already incurred in training the main model M_N. Therefore, the communication cost is assumed to be negligible. All computations are done on the same machine instead of separate server and client machines as in real use cases.

Chapter 4 Experimental Setup

4.1 Dataset

The experiments are done on the classic MNIST dataset for hand-written digits in black and white [6]. The pre-processing of data follows the framework by Song et al. [5]. The MNIST dataset has 60000 training images and 10000 test images. For both training set and test set, the number of images of different labels may vary, i.e. the number of images of digit "1" may differ from the number of images of digit "2". We randomly exclude some images from the training set so that each digit has 5420 images. We do not edit the test set.

To simulate real-case scenarios where different data contributors may have different size, distribution and quality of data, with reference to Tong et al. [5], the training set is split into 5 datasets (for $n = 5$) in 5 different ways as follows. Note that when the number of data contributors n is 10, each of the 5 datasets are further randomly split into 2 smaller datasets.

- Case 1: same distribution with same dataset size. Each dataset D_i has the same size and the same number of images corresponding to each digit. For this case, each dataset has 1084 training images for each digit.
- Case 2: different distribution with same dataset size. Each dataset D_i has the same size, but the training images are not equally divided for each digit. 40% of D_i is of digit "(2i-2)", another 40% is of digit "(2i-1)" and the remaining 8 digits equally share the remaining 20%. For example, D_1 has 4336 images of "0", 4336 images of "1", and 271 is for each of the other digits.
- Case 3: same distribution with different dataset size. The training set is split into 5 parts randomly whose ratio of data size is 2:3:4:5:6. Within each dataset D_i, the digits have equal percentage. For example, D_1 has

5420 images and each digit accounts for 542 images, and D_2 has 8130 images and each digit has 813 images.

- Case 4: noised data on label with same dataset size. First, we split the training set in similar manner as case 1. We then change the labels of the different datasets randomly by 0%, 5%, 10%, 15%, 20%. This means that 5% of the labels for the training images in D_2 are wrong, while 20% of the labels for the training images in D_5 are wrong.
- Case 5: noised data on feature with same data size. First, we split the training set in similar manner as case 1. Afterwards, we generate 0 – 4x Gaussian noise, where 0x is for the feature without noise. This is done by adjusting the standard deviation of the normal distribution. The different levels of noise are then added to the all the images of the different datasets accordingly, so that D_1 will have no noise and D_5 will have the greatest level of noise. Figure 4-1 below shows the effect of the noise on images.

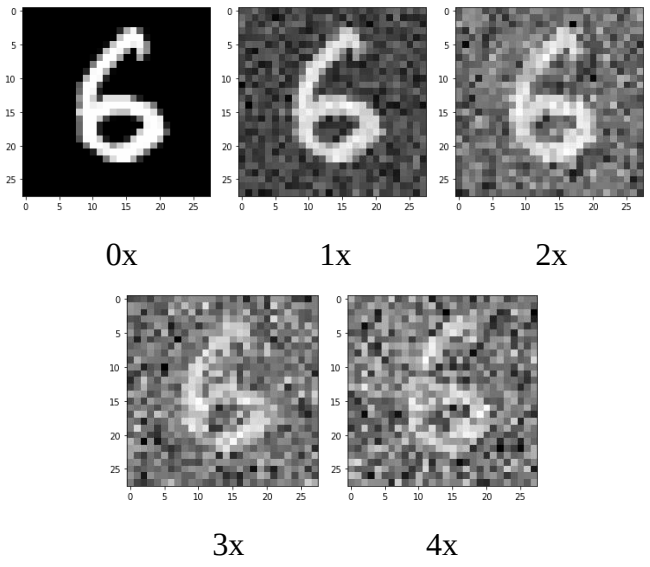

Figure 4-1: Images with different levels of Gaussian noise.

4.2 Environment

The experiments are conducted on Google Colab with Tesla T4 GPU and Intel(R) Xeon(R) CPU @ 2.20GHz with 13GB of RAM. The codes are implemented on python with PyTorch 1.4.0 and NumPy as the main packages.

Chapter 5 Experiments and Results

5.1 OR vs Adjusted OR

These two algorithms are only compared in terms of time taken to calculate Shapley Values. The accuracy is not compared since mathematically they produce the same results.

Each of the algorithms is run 5 times for each case of forming the datasets as detailed in Section 4.1. The average time taken is computed from these repetitions. The average across the 5 cases is also calculated.

5.1.1 Run for 5 Clients and 5 Global Iterations

Figure 5-1 shows the time taken to run OR and Adjusted OR algorithms, where number of data contributors n is 5 and number of global iterations T is 5. The vertical axis represents the time taken in seconds, while the horizonal axis represents the case of splitting the dataset as detailed in Section 4.1.

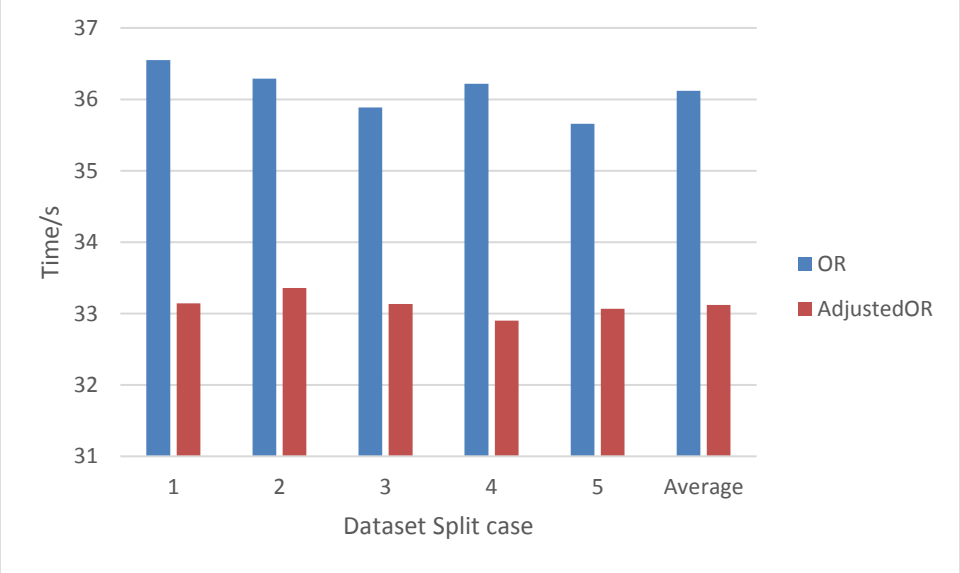

Figure 5-1: Time taken for OR vs Adjusted OR for n=5 and T=5.

As Figure 5-1 suggests, Adjusted OR algorithm consistently outperforms OR in terms of time. On average it is faster than OR by about 3.0s, or 8.3%.

5.1.2 Run for 5 Clients and 10 Global Iterations

Figure 5-2 shows the time taken to run OR and Adjusted OR algorithms, where number of data contributors n is 5 and number of global iterations T is 10.

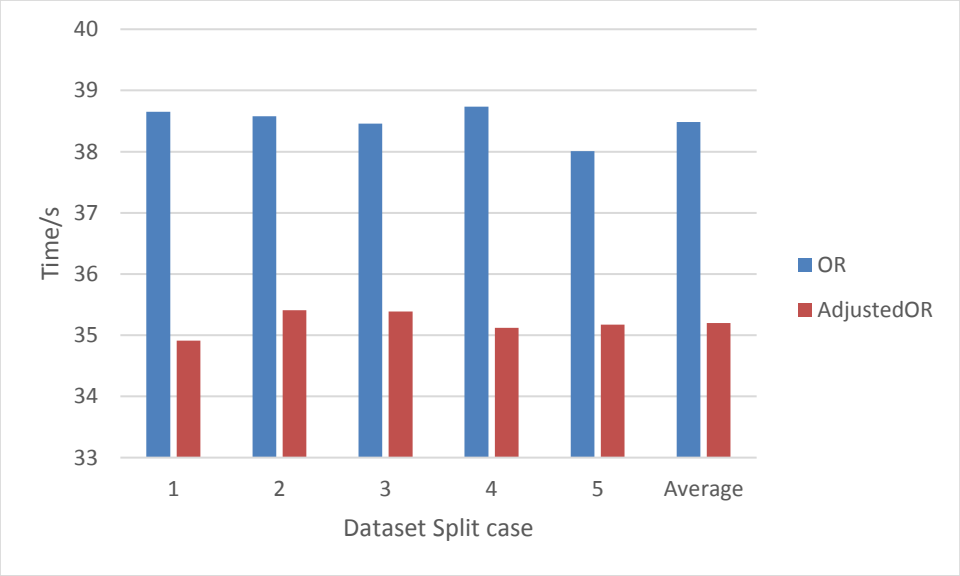

Figure 5-2: Time taken for OR vs Adjusted OR for n=5 and T=10.

Similar to 5.1.1, Adjusted OR takes less time to run than OR. On average it is faster than OR by 8.5%. It is interesting to note that when the number of global iterations is doubled from 5.1.1 to 5.1.2, the average time taken is only increased by 6.3% for Adjusted OR and 6.5% for OR. The smaller increase in time for Adjusted OR compared to OR can be attributed to the fact that OR updates $\mathbf{2^n - 2}$ (which is 30 for n=5) approximate models per global iteration while Adjusted OR only updates $\boldsymbol{n}$ approximate models. As the number of global iterations increases, Adjusted OR will save even more time compared to OR.

5.1.3 Run for 10 Clients and 5 Global Iterations

Figure 5-3 shows the time taken to run OR and Adjusted OR algorithms, where number of data contributors n is 10 and number of global iterations T is 5. Note that the datasets for the 10 clients are created by randomly splitting each of the five datasets D_i created from Section 4.1 for the n=5 setting into 2 smaller datasets.

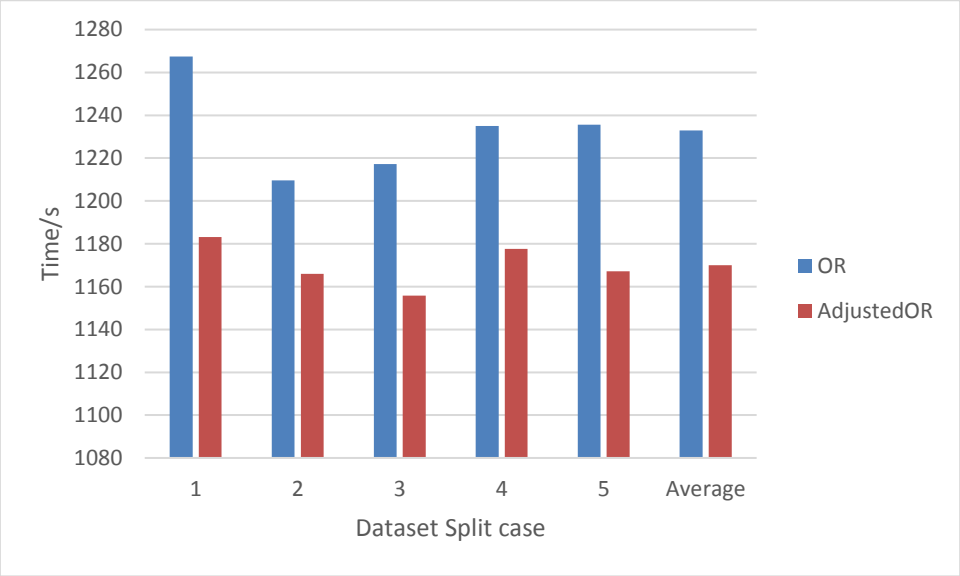

Figure 5-3: Time taken for OR vs Adjusted OR for n=10 and T=5.

On average, Adjusted OR saves 63.01716s, or 5.1% of runtime compared to OR. This is a huge gap as compared to 5.1.1. When the number of clients is doubled, the time difference between Adjusted OR and OR is increased by more than 20 times. This is expected since the time saving to update n approximate models instead of $2^n - 2$ models per global iteration will increase with n.

5.1.4 Overall Comments

Figure 5-4 shows the average time taken for OR and Adjusted OR when different parameters are used. When the number of global iterations T is doubled, keeping n constant, the increase in runtime is not significant. However, when the number of data contributors is doubled, keeping T at 5, the runtime increases by more than 30x for both algorithms. Even though Adjusted OR

streamlines the model reconstruction process, it does not solve the fundamental problem of evaluating 2^n models to calculate SV.

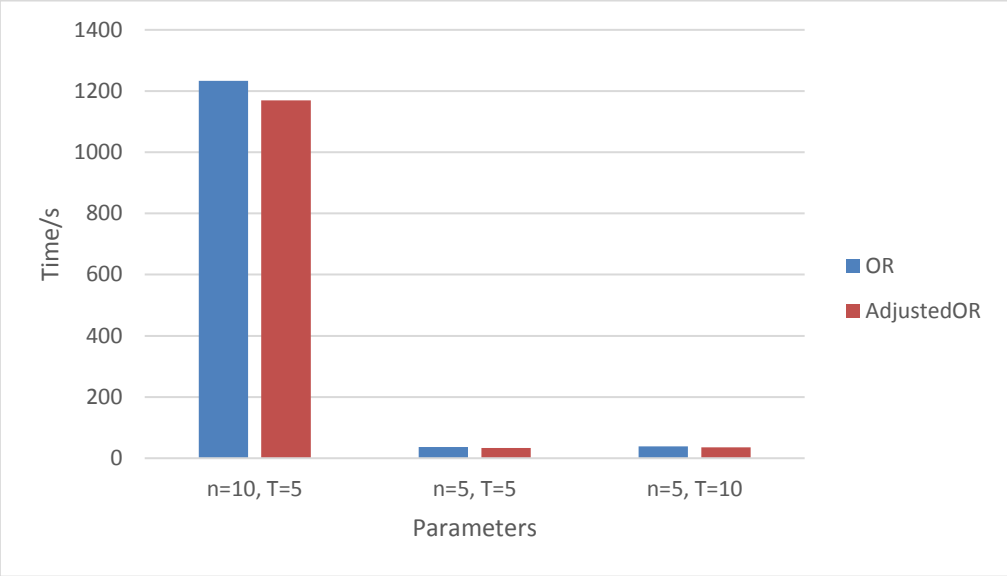

Figure 5-4: Average time taken for OR vs Adjusted OR with different parameters.

For subsequent comparisons, Adjusted OR will be used instead of OR because it is clearly the superior algorithm in terms of runtime and it also has the same accuracy as OR.

5.2 Adjusted OR vs OR-TMC vs Adjusted OR-TMC

5.2.1 Run for 10 Clients and 10 Global Iterations

Due to the astronomical amount of time required to run Exact Federated Shapley for *n*=10, and the time limit allowed on Google Colab, the SV values for benchmarking the accuracy of approximation algorithms are not obtained experimentally. Instead, the values used for accuracy benchmarking for *n*=10 are obtained by dividing the average standardised SV generated by Exact Federated Shapley for *n*=5 by 2. This is because we assume that when each of the 5 big datasets are each randomly split in 2 equal, smaller datasets, these two smaller datasets will equally share the SV of the big dataset.

For each case of dataset setting, each of the approximation algorithms, namely Adjusted OR, OR-TMC and Adjusted OR-TMC, is run for 5 times and the average run time, Maximum Euclidean Distance and Average Euclidean Distance are calculated. Both OR-TMC and Adjusted OR-TMC are set with the same TMC error of 1% and performance tolerance of 1%.

If all standardised SV generated by an approximation algorithm differ by 0.05 from the values generated by Exact Federated Shapley, the Euclidean Distance will be 0.16. Hence, a Euclidean Distance of more than 0.16 signifies at least one standardised ϕ_i differs more than 0.05 from the value calculated by Exact Federated Shapley. This is a significant error for n=10 because the SV for each data contributors is smaller.

5.2.1.1 Case 1: Same Distribution with Same Dataset Size

Figure 5-15 (Left) shows the average time taken for the approximation algorithms to calculate SV of data contributors. Adjusted OR takes more than 1100s to run, which is multiple times longer than training the main federated model M_N, and hence is impractical. OR-TMC is 36.7% faster runtime and Adjusted OR-TMC is 45.5% faster than Adjusted OR, which are huge improvements (Figure 5-5). This is due to the smaller number of models being evaluated. Adjusted OR calculates the performance score $U\left(\widetilde{M}_S^T\right)$ for 1024 models corresponding to 1024 subsets S, whereas OR-TMC and Adjusted OR-TMC evaluates much fewer models.

It is worth looking at the number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge (Figure 5-5 (Right)). Note that for Adjusted OR-TMC, this number is obtained by multiplying the number of TMC iterations with n, as explained in Section 3.1.3. OR-TMC takes more than 2x the number of permutations to converge compared to Adjusted OR-TMC. The reason why runtime of OR-TMC is not 2x that of Adjusted OR-TMC is that the time spent for each permutation is not constant. Truncation is used which stop

the calculations for a permutation if the performance tolerance is reached. Moreover, the $U\left(\widetilde{M_S^T}\right)$ are stored between permutations (see Section 3.1.3) and the permutation is randomised. Since the bulk of the time is spent evaluating the accuracy of the approximated models, two permutations having more similar portions, for example, (1,2,3,4,5) and (1,2,3,5,4), can have shorter time for calculations. Even though OR-TMC takes 126 permutations, which means theoretically the maximum number of models it evaluates is about 1260, many of these evaluations are not done thanks to truncation within a permutation and information saving in between permutations. Therefore OR-TMC, while being worse than Adjusted OR-TMC, is still much faster than Adjusted OR.

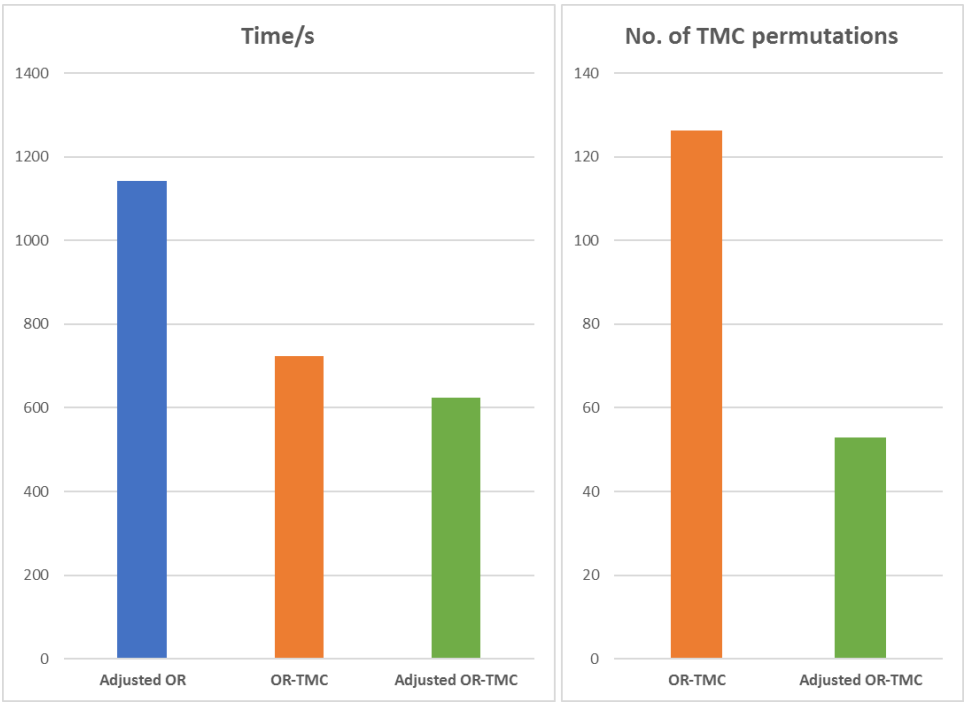

Figure 5-5: (Left) Average time taken to run different approximation algorithms for n=10 and T=10 for same distribution with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

Figure 5-6 shows that Adjusted OR and Adjusted OR-TMC have similar accuracy, while OR-TMC performs the worst on both metrics, with more than 2x the Average Euclidean Distance and Maximum Euclidean Distance as the

other two algorithms. This suggests that OR-TMC performs badly both on average and in a worst-case scenario.

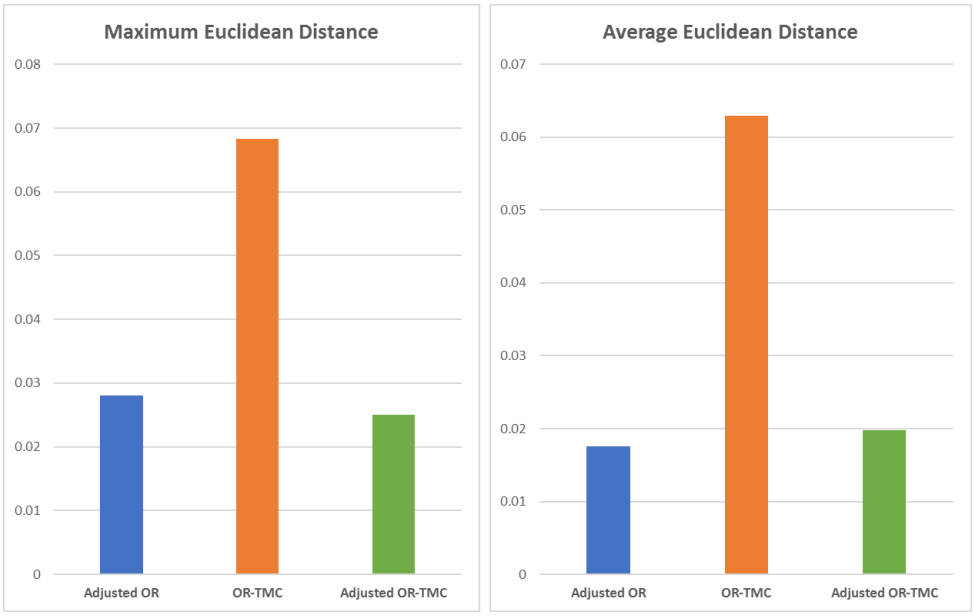

Figure 5-6: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=10 and *T*=10 for same distribution with same dataset size.

5.2.1.2 Case 2: Different Distribution with Same Dataset Size

OR-TMC runs the fastest for this case, saving 56.0% compared to Adjusted OR (Figure 5-7). Adjusted OR-TMC is 14.6% faster than Adjusted OR. It is interesting to note how OR-TMC converges very fast for this case of dataset setting, similar to the same case with *n*=5 in Section 5.2.2.2. OR-TMC takes nearly 50 permutations fewer than Adjusted OR-TMC to converge.

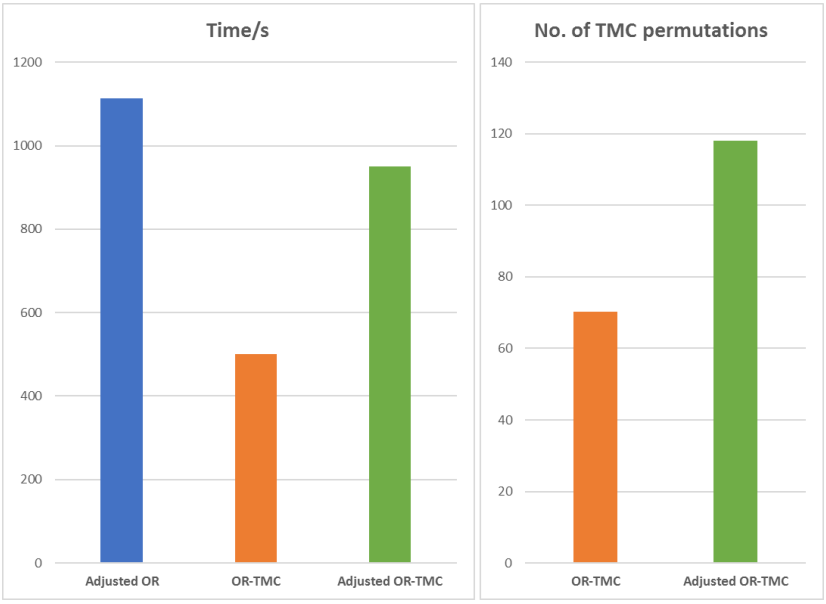

Figure 5-7: (Left) Average time taken to run different approximation algorithms for n=10 and T=10 for different distribution with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, OR-TMC continues to be the least accurate on both metrics (Figure 5-8). Adjusted OR-TMC's Maximum Euclidean Distance is 0.017 more than that of Adjusted OR, but it performs well on average, with Average Euclidean Distance only 0.0031 more than that of Adjusted OR.

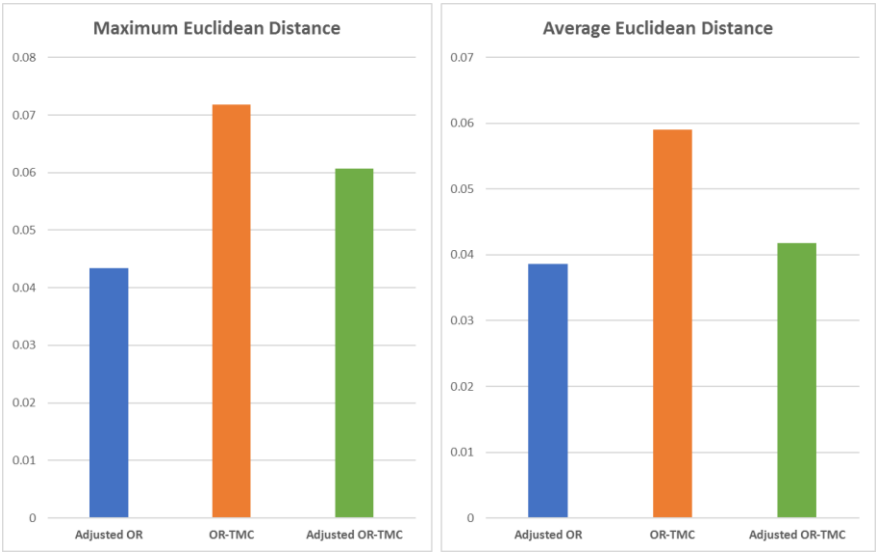

Figure 5-8: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=10 and *T*=10 for different distribution with same dataset size.

5.2.1.3 Case 3: Same Distribution with Different Dataset Sizes

OR-TMC and Adjusted OR-TMC have similar runtime and are about 40% faster than that of Adjusted OR (Figure 5-9). OR-TMC takes almost 2x as many permutations to converge as compared to Adjusted OR-TMC.

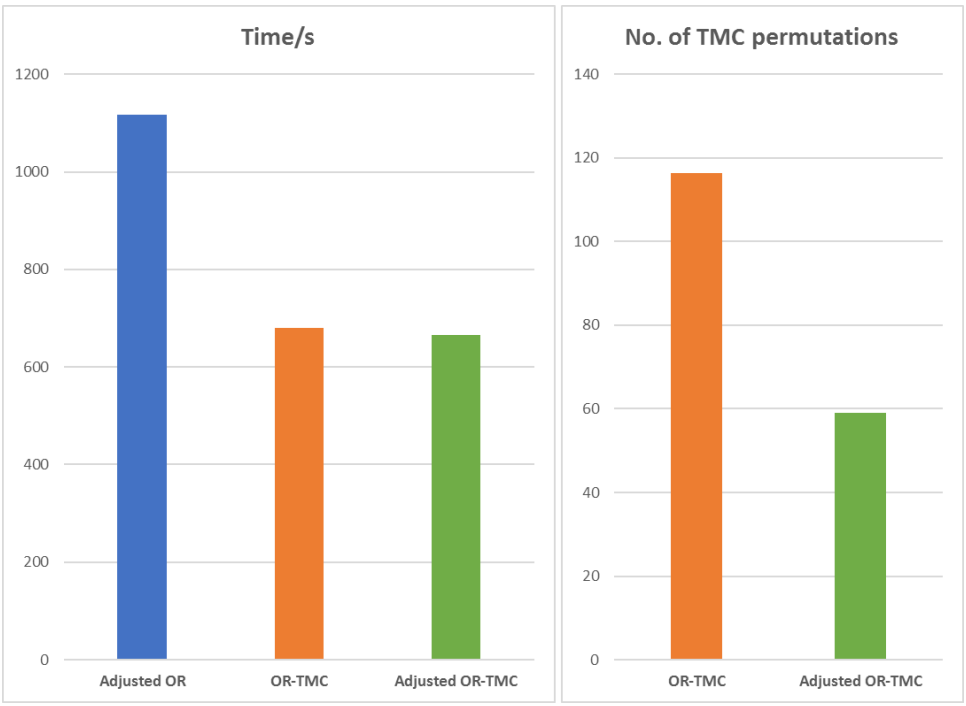

Figure 5-9: (Left) Average time taken to run different approximation algorithms for n=10 and T=10 for same distribution with different dataset sizes. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, all three algorithms perform worse than the case for same distribution and same dataset size in Section 5.2.1.1, with Average Euclidean Distance being from 1.6x to 4.3x as large as the corresponding algorithms in 5.2.1.1. The Adjusted OR and Adjusted OR-TMC perform similarly on both metrics. OR-TMC is the worst performing, with Average Euclidean Distance 35.5% larger than that of Adjusted OR, and Maximum Euclidean Distance 53.4% larger (Figure 5-10).

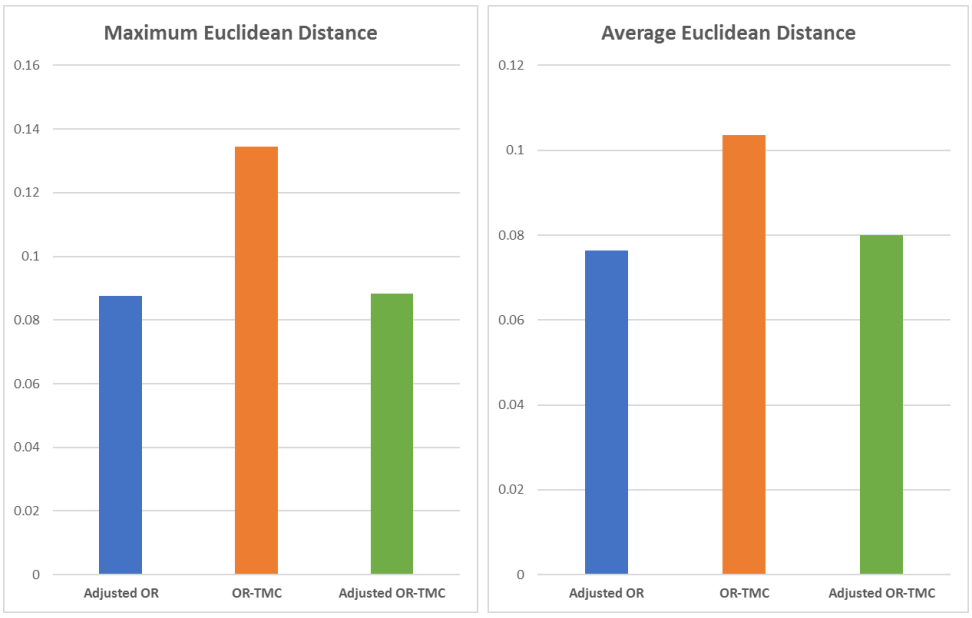

Figure 5-10: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=10 and *T*=10 for different distribution with same dataset sizes.

5.2.1.4 Case 4: Noised Data on Label with Same Dataset Size

OR-TMC has the shortest runtime, 36.4% less than Adjusted OR. Meanwhile, Adjusted OR is 6.9% faster than Adjusted OR, and about 20 more permutations to converge compared to OR-TMC (Figure 5-11).

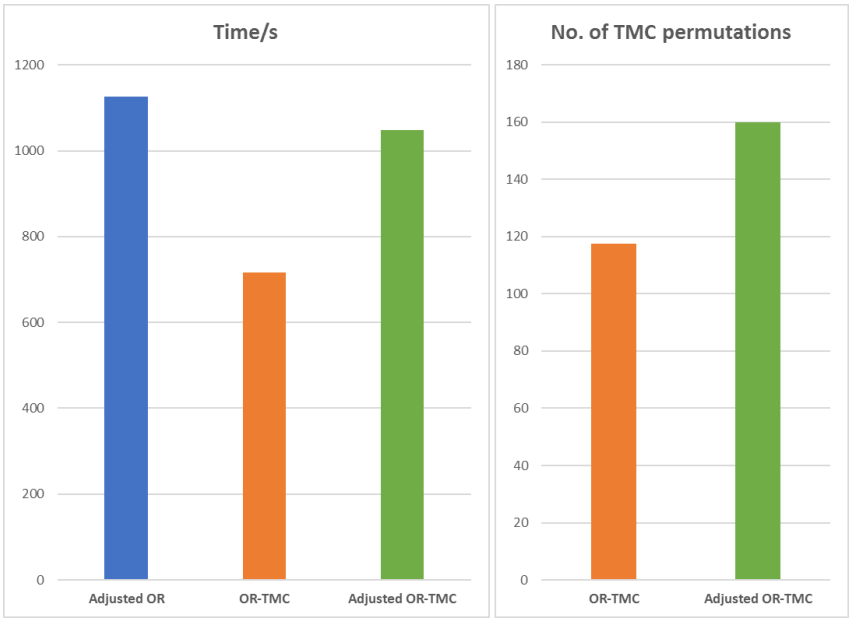

Figure 5-11: (Left) Average time taken to run different approximation algorithms for n=10 and T=10 for noised data on label with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, Adjusted OR-TMC and OR-TMC performs slightly better than Adjusted OR on both metrics. As a whole, all three approximation algorithms perform poorly for this case, with Average Euclidean Distance above 0.25 (Figure 5-12), which means that there exists a data contributor whose SV calculated differs from the actual value by more than 0.05.

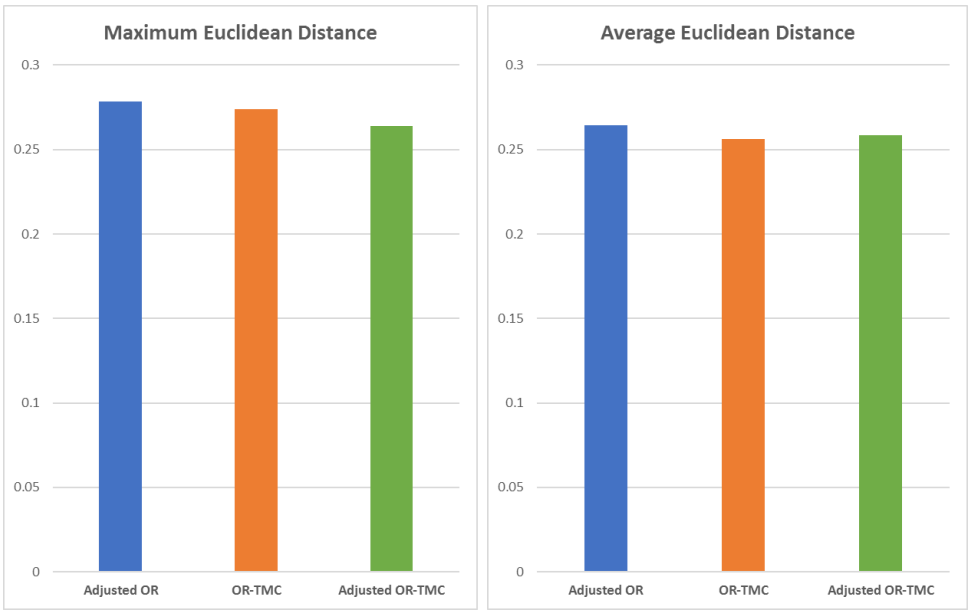

Figure 5-12: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=10 and *T*=10 for noised data on label with same dataset size.

Table 5-1 shows a comparison of the average standardised SV produced by different algorithms. Note that the values by Exact Federated Shapley are not experimentally determined but derived from the values for *n*=5 as explained earlier on.

Table 5-1: Average Standardised SV (rounded to 4 dp) produced by different algorithms for n=10 and T=10 for noised data on label with same dataset size.

Algorithm	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}
Exact Federated	0.0999	0.0999	0.1009	0.1009	0.1001	0.1001	0.0997	0.0997	0.0994	0.0994
Adjusted OR	0.2167	0.2184	0.1685	0.1669	0.0911	0.0950	0.0337	0.0245	-0.0063	-0.0085
OR-TMC	0.2104	0.2132	0.1563	0.1648	0.0883	0.1055	0.0394	0.0229	0.0030	-0.0039
Adjusted OR-TMC	0.2170	0.2050	0.1682	0.1707	0.0926	0.0963	0.0357	0.0325	-0.0118	-0.0062

Since the level of noise increases for D_i every time i increases by 2, the SV of data contributors should show a decreasing trend as seen on the row for the Exact Federated Shapley. SV produced by the approximation algorithms also show the correct decreasing trend, but the decrease is greater than the exact one, with ϕ_9 and ϕ_{10} having negative values. This suggests that which suggests that using local updates from the 9th and 10th data contributors reduces the accuracy of the approximated models $\widetilde{M}_S^T$. However, these negative values do not reflect the true values for contributors with indices 9 and 10, since each of them still contributes 0.0994, or nearly 10% of total contribution. The discrepancy is likely caused by the OR reconstruction method. This is an important finding to complement Song et al.'s work, which shows that OR is sensitive to noised label but does not examine the values obtained.

5.2.1.5 Case 5: Noised Data on Feature with Same Dataset Size

OR-TMC and Adjusted OR-TMC have similar runtime and are about 37% faster compared to Adjusted OR (Figure 5-13). Adjusted OR-TMC takes 23s less to run on average compared to OR-TMC, and fewer permutations to converge.

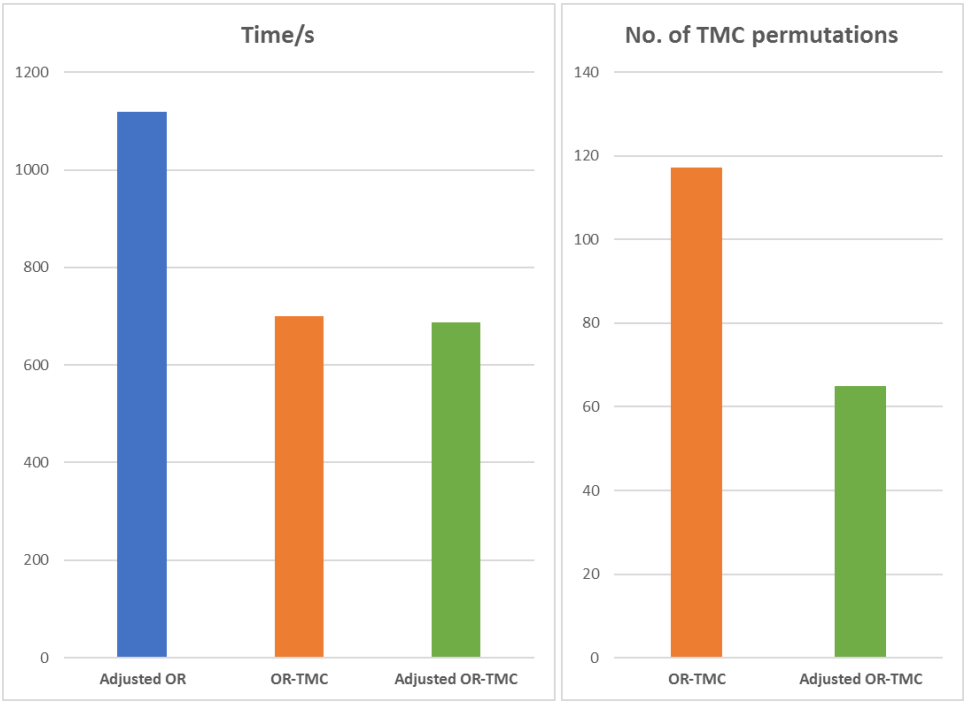

Figure 5-13: (Left) Average time taken to run different approximation algorithms for n=10 and T=10 for noised data on feature with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, the three approximation algorithms perform worse in this case than case 1 (Section 5.2.1.1) and case 2 (Section 5.2.1.2) but is better than case 3 (Section 5.2.1.3) and case 4 (Section 5.2.1.4). Adjusted OR-TMC has the best performance with Average Euclidean Distance 15.8% smaller than that of Adjusted OR, while OR-TMC has the worst performance on both metrics.

Figure 5-14: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=10 and *T*=10 for noised data on feature with same dataset size.

Even though the three algorithms are more accurate in this case as compared to case 4, the SV values generated do not follow the decreasing trend as expected (Table 5-2). The fact that they do not follow the expected trend is similar to findings from Table 5-5 for *n*=5, and may be the consequence of the OR reconstruction step. OR-based methods are hence not so suitable for noised feature scenario. The table complements Song et al.'s work, which shows that OR is sensitive to noised features but does not examine the values obtained.

Table 5-2: Average Standardised SV (rounded to 4 dp) produced by different algorithms for n=10 and T=10 for noised data on label with same dataset size.

Algorithm	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}
Exact Federated	0.1011	0.1011	0.1017	0.1017	0.1004	0.1004	0.0990	0.0990	0.0977	0.0977
Adjusted OR	0.0787	0.0690	0.0763	0.0859	0.1185	0.1160	0.1186	0.1185	0.1152	0.1033
OR-TMC	0.0721	0.0683	0.0832	0.0890	0.1150	0.1263	0.1066	0.1229	0.1127	0.1039
Adjusted OR-TMC	0.0816	0.0789	0.0766	0.0902	0.1094	0.1112	0.1184	0.1172	0.1108	0.1057

5.2.1.6 Overall Comments

OR-TMC and Adjusted OR-TMC manage to shorten the runtime for Adjusted OR significantly when n=10. By doing so, they make SV approximation more practical especially when the number of data contributors is large.

Because OR-TMC and Adjusted OR-TMC use the TMC approximation to further shorten the time taken by Adjusted OR, these two algorithms generally have worse accuracy than Adjusted OR. Nonetheless, Adjusted OR-TMC is on par with Adjusted OR and is consistently more accurate than OR-TMC. Adjusted OR-TMC is hence a superior alternative to Adjusted OR, which is itself superior to the current OR algorithm.

There are still cases where such as 4 and 5, where all three algorithms perform poorly, especially for case 5 where the SV calculated do not even follow the expected trend. This highlights a drawback in OR-based reconstruction method when noised data are involved.

For the next section, experiments with smaller n are conducted to test the effect of n on the time-saving ability of OR-TMC and Adjusted OR-TMC.

5.2.2 Run for 5 Clients and 10 Global Iterations

Exact Federated Shapley is run for 3 times for each case of dataset split as presented in 4.1 and the average of the standardised SV is calculated across the 3 runs. The average value is used as benchmark to calculate the Euclidean Distance as presented in Section 3.2. For each case of dataset setting, each of the approximation algorithms is run for 5 times and the average run time, Maximum Euclidean Distance and Average Euclidean Distance are calculated. Both OR-TMC and Adjusted OR-TMC are set with the same TMC error of 1% and performance tolerance of 1%.

For the same case, if all standardised SV generated by an approximation algorithm differ by 0.05 from the values generated by Exact Federated Shapley, the Euclidean Distance will be 0.11. Hence, a Euclidean Distance of more than 0.11 signifies at least one standardised ϕ_i differs more than 0.05 from the value calculated by Exact Federated Shapley.

5.2.2.1 Case 1: Same Distribution with Same Dataset Size

Figure 5-15 (Left) shows the average time taken for the approximation algorithms to calculate SV of data contributors. Adjusted OR-TMC has the shortest runtime, running 2.5% faster than Adjusted OR. OR-TMC has the longest runtime, 14.3% more than Adjusted OR.

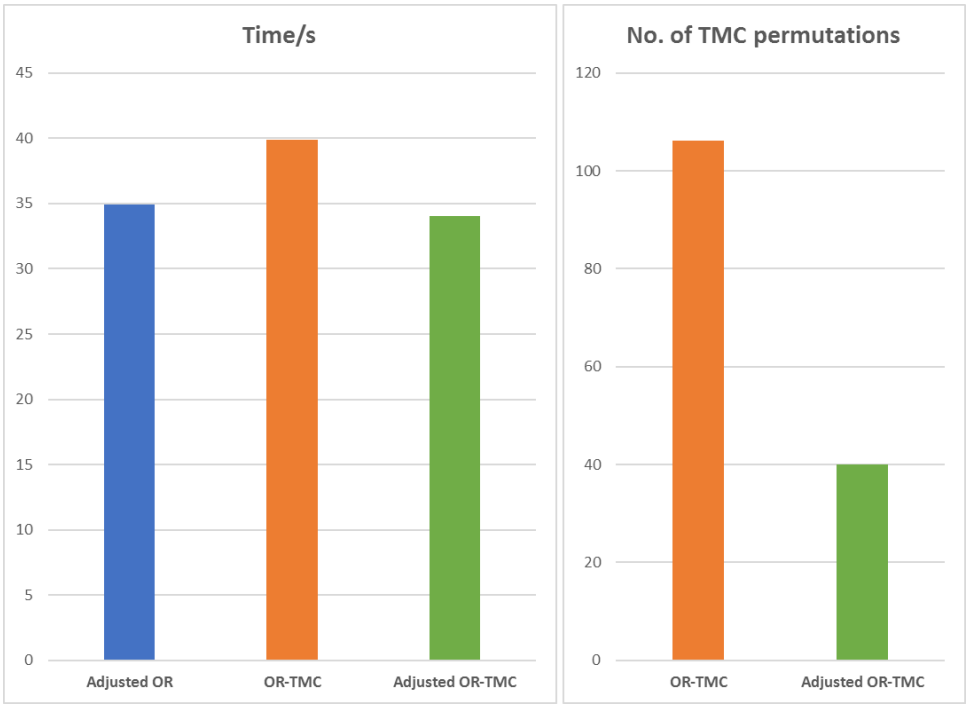

Figure 5-15: (Left) Average time to run different approximation algorithms for *n*=5 and *T*=10 for same distribution and same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

OR-TMC takes a lot of permutations to converge, probably due to reasons presented in 3.1.3. On average, OR-TMC takes more than 2.5x the number of permutations to converge compared to Adjusted OR-TMC. OR-TMC's behaviour is also erratic: one test run (not included here) takes only 2 permutations to converge. This is possibly because of the randomisation of the

permutations. When the first two permutations happen to be identical (with $\frac{1}{120}$ chance for n=5), the convergence criterion is immediately met.

OR-TMC and Adjusted OR-TMC are not significantly faster than Adjusted OR for n=5 as compared to Section 5.2.1.1, due to the smaller value of n. After running 40+ permutations, nearly all of $U\left(\widetilde{M_S^T}\right), S \subseteq N$ have been calculated (there are only 32 such subsets S for n=5), and therefore these two algorithms have quite similar runtime as Adjusted OR. Still, all the three approximation algorithms have shortened the time to calculate SV by more than 65x. Exact Federated Shapley, which is not included on the graph, takes 2772s to run on average.

Figure 5-16 shows the Maximum Euclidean Distance and Average Euclidean Distance for the three approximation algorithms. Adjusted OR performs best, while Adjusted OR-TMC has about 2x the Maximum Euclidean Distance and about 1.5x the Average Euclidean Distance compared to OR. OR-TMC is the worst performing: its Maximum Euclidean Distance is more than 4x that of OR and its Average Euclidean Distance is nearly 4x that of OR.

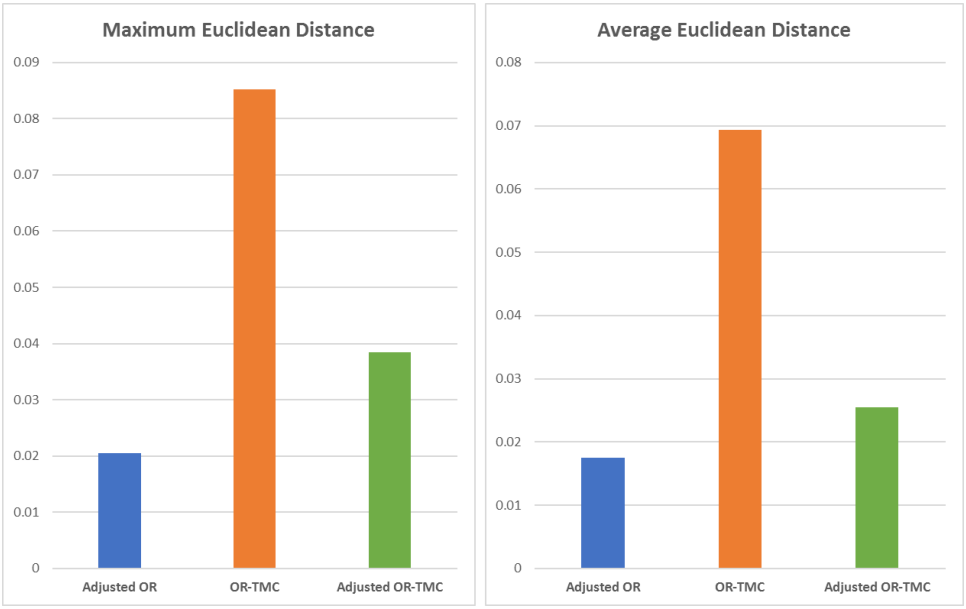

Figure 5-16: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=5 and *T*=10 for same distribution and same dataset size.

5.2.2.2 Case 2: Different Distribution with Same Dataset Size

For this case, all three algorithms have similar average runtime, although Adjusted OR-TMC runs slightly faster (0.4% less time) and OR-TMC runs slightly slower (2.6% more time) compared to Adjusted OR (Figure 5-17 (Left)). It is interesting that for this case, both OR-TMC and Adjusted OR-TMC require much fewer permutations to converge compared to 5.2.2.1. OR-TMC requires 5.6 fewer permutations to converge on average, but still runs slower than Adjusted OR-TMC (Figure 5-17 (Right)). This further highlights the fact that time spent on each permutation is not constant.

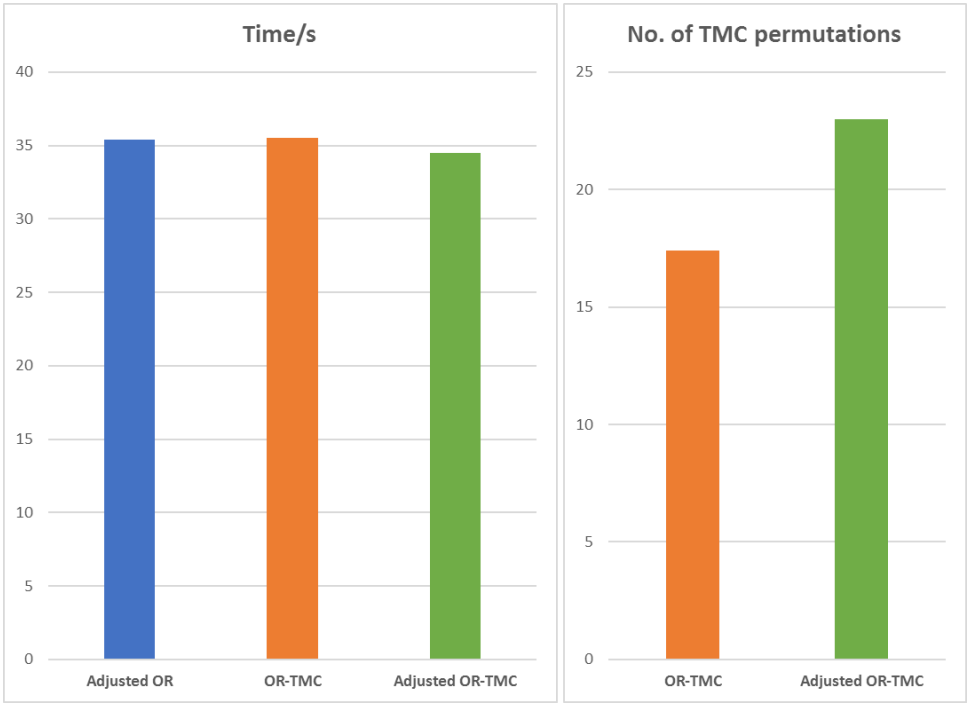

Figure 5-17: (Left) Average time to run different approximation algorithms for n=5 and T=10 for different distribution and same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy (Figure 5-18), the performance of the three algorithms are similar for Average Euclidean Distance. Even so, adjusted OR-TMC performs best on both metrics. OR-TMC continues to have the highest Maximum Euclidean Distance.

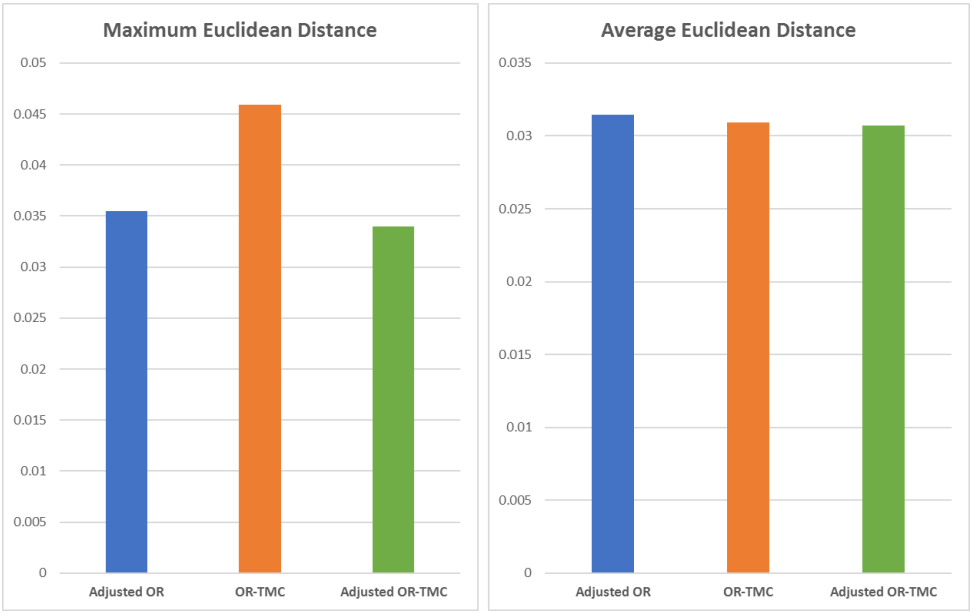

Figure 5-18: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=5 and *T*=10 for different distribution and same dataset size.

5.2.2.3 Case 3: Same Distribution with Different Dataset Sizes

Figure 5-19 is quite similar to Figure 5-15. Adjusted OR-TMC runs slightly faster (1.4% less time) than Adjusted OR, while OR-TMC runs slower (11.9% more time) than Adjusted OR. OR-TMC takes 2.5x as many permutations to converge compared to Adjusted OR-TMC.

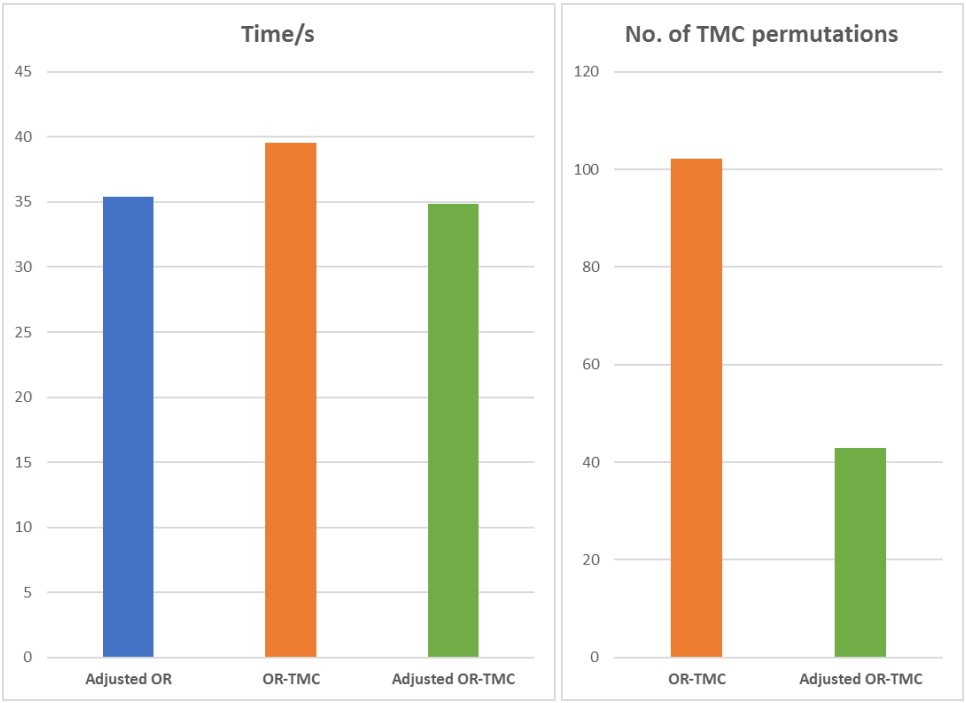

Figure 5-19: (Left) Average time to run different approximation algorithms for n=5 and T=10 for same distribution and different dataset sizes. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, Adjusted OR-TMC performs slightly better than Adjusted OR on average, and slightly worse in Maximum Euclidean Distance. OR-TMC is still the worst performing. Overall, all three algorithms have much worse accuracy in this case compared to 5.2.2.1. Adjusted OR has nearly 6x the Average Euclidean Distance compared to that of the same algorithm in 5.2.2.1, while Adjusted OR has 4x and OR-TMC has nearly 2x the Average Euclidean Distance compared to the same algorithms 5.2.2.1.

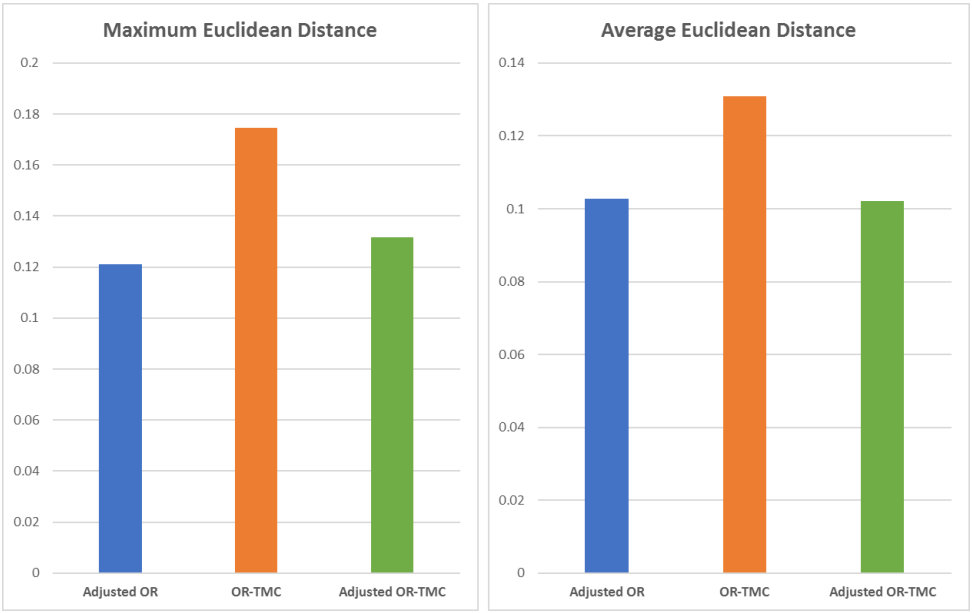

Figure 5-20: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=5 and *T*=10 for same distribution and different dataset sizes.

A possible explanation to this phenomenon is the way FedAvg and OR works. FedAvg only approximates the gradients to the global model by taking an average of the local updates sent from the clients. This may cause Exact Federated Shapley to be not so "exact": the SV calculated deviate from the trend generated by the Non-Federated algorithm. Table 5-3 shows a comparison of the average standardised SV produced by different algorithms. Note that the values generated by Exact Non-Federated Shapley are the average across 5 runs which have a different assumption to the rest, that $U(M_{\emptyset}) = 0$ instead of being equal to the value of a randomly initialized model. However, this does not affect the trend in Shapley values.

Table 5-3: Average Standardised SV (rounded to 4 dp) produced by different algorithms for n=5 and T=10 for same distribution and different dataset size.

Algorithm	$\boldsymbol{\phi_1}$	$\boldsymbol{\phi_2}$	$\boldsymbol{\phi_3}$	$\boldsymbol{\phi_4}$	$\boldsymbol{\phi_5}$
Exact Non-Federated Shapley	0.1919	0.1967	0.2016	0.2046	0.2051
Exact Federated Shapley	0.2000	0.1984	0.2005	0.2012	0.2000
Adjusted OR	0.1494	0.1439	0.2125	0.2350	0.2592
OR-TMC	0.1541	0.1443	0.2389	0.1835	0.2793
Adjusted OR-TMC	0.1462	0.1497	0.2076	0.2499	0.2465

In fact, the trend shown by Exact Non-Federated Shapley is consistent with our expectations, since the datasets have equal distribution but different size, in ratio of 2:3:4:5:6. Therefore, ϕ_i should be strictly increasing when i increases. However, this is not the case for Exact Federated Shapley. The SV calculated by Exact Federated Shapley are very close to each other, which suggests that perhaps this is because MNIST task is too simple for dataset size to be significant. A harder task that requires larger quantity of data to achieve the same accuracy would have been more ideal for the analysis. This is further discussed in Section 6.2.

Interestingly Adjusted OR is more consistent with the trend of Exact Non-Federated Shapley than Exact Federated Shapley, even though the increasing trend it shows is more exaggerated than the actual one. This may be attributed to the model approximation $\widetilde{M}_S^T$ as explained in 3.1.2. Adjusted OR-TMC also mirrors this trend by Adjusted OR, which is expected because OR-TMC and Adjusted OR-TMC rely on and are supposed to approximate the (Adjusted) OR algorithm. However, OR-TMC results seem to be rather random.

5.2.2.4 Case 4: Noised Data on Label with Same Dataset Size

Adjusted OR-TMC takes nearly the same time (only 0.8% more) to run as compared to Adjusted OR (Figure 5-21). OR-TMC continues to take the longest time to run, 11.5% more than Adjusted OR, even though it takes fewer permutations to converge compared to Adjusted OR-TMC. Adjusted OR-TMC takes much more permutations to converge in this case compared to 5.2.2.1, 5.2.2.2 and 5.2.2.3. This is perhaps due to the significant difference in data quality due to different levels of noise on labels.

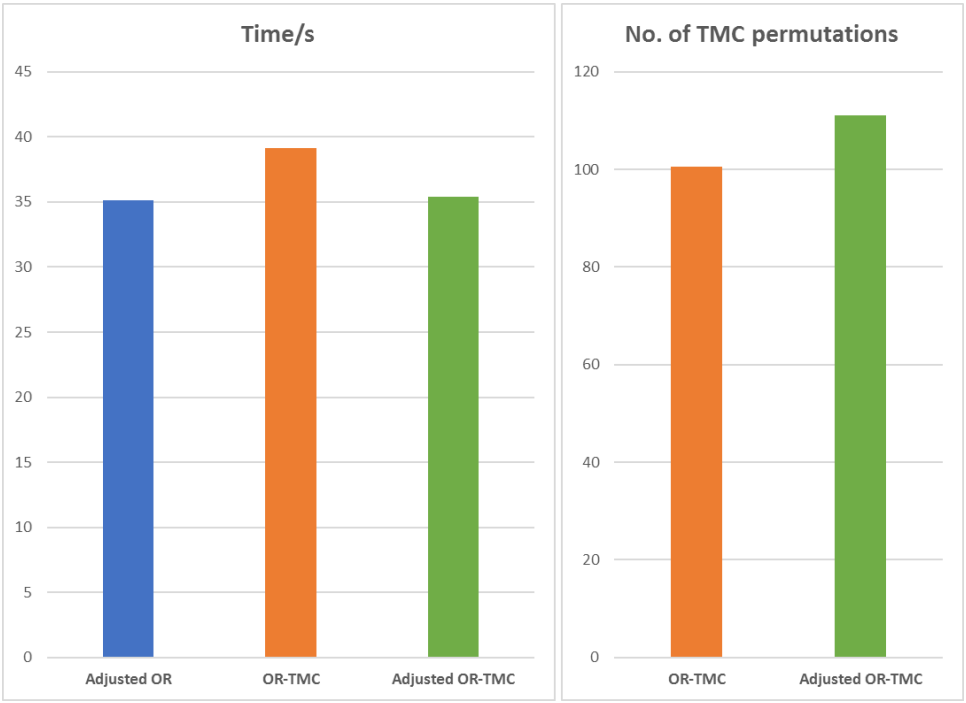

Figure 5-21: (Left) Average time to run different approximation algorithms for n=5 and T=10 for noised data on label with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

In terms of accuracy, all three approximation algorithms perform poorly for this case, with Average Euclidean Distance above 0.4 (Figure 5-22), which means that there exists a data contributor whose SV calculated differs from the actual value by at least 0.18: if any of the approximation algorithms is used for profit distribution, that data contributor will gain or lose 18% of the total profit distributed compared to what he actually contributes. Adjusted OR-TMC performs best on both metrics but is not significantly better than Adjusted OR. OR-TMC has the highest Maximum Euclidean Distance, while its Average Euclidean Distance is similar (1.5% less) to Adjusted OR.

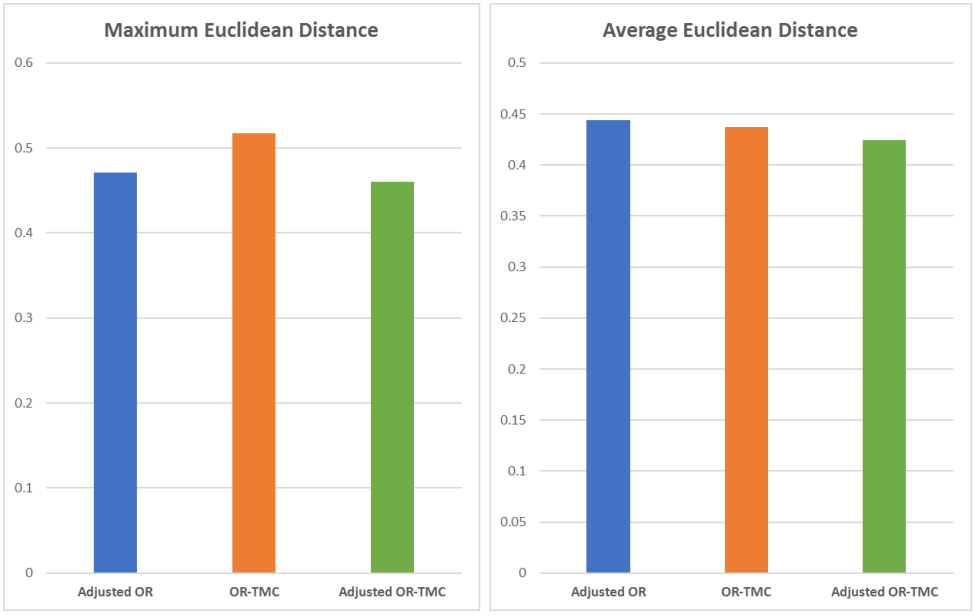

Figure 5-22: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=5 and *T*=10 for noised data on label with same dataset size.

Examining the average standardised SV generated by different algorithms in Table 5-4, we can observe that the trend by Exact Federated Shapley closely follows the decreasing trend shown in Exact Non-Federated Shapley, although there is an anomaly at ϕ_1. The other algorithms also follow this decreasing trend, but the values decrease much more significantly. ϕ_5 has negative values for all three approximation algorithms, which suggests that using local updates from the 5th data contributor reduces the accuracy of the approximated models $\widetilde{M}_S^T$. These algorithms show the trend but do not reflect the true value for this case, since the 5th data contributor still contributes significantly to the federated model, as shown by the value of 0.1989 by Exact Federated Shapley.

Table 5-4: Average Standardised SV (rounded to 4 dp) produced by different algorithms for n=5 and T=10 for noised data on label with same dataset size.

Algorithm	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
Exact Non-Federated Shapley	0.2013	0.2023	0.2008	0.1983	0.1973
Exact Federated Shapley	0.1998	0.2018	0.2001	0.1994	0.1989
Adjusted OR	0.4930	0.3597	0.1533	0.0257	-0.0316
OR-TMC	0.4704	0.3652	0.1642	0.0284	-0.0282
Adjusted OR-TMC	0.4682	0.3623	0.1704	0.0233	-0.0241

5.2.2.5 Case 5: Noised Data on Feature with Same Dataset Size

Similar to 5.2.2.4, Adjusted OR-TMC takes about the same time to run as compared to Adjusted OR (Figure 5-21). OR-TMC takes the longest time to run, 10.4% more than Adjusted OR, and more permutations to converge compared to Adjusted OR-TMC.

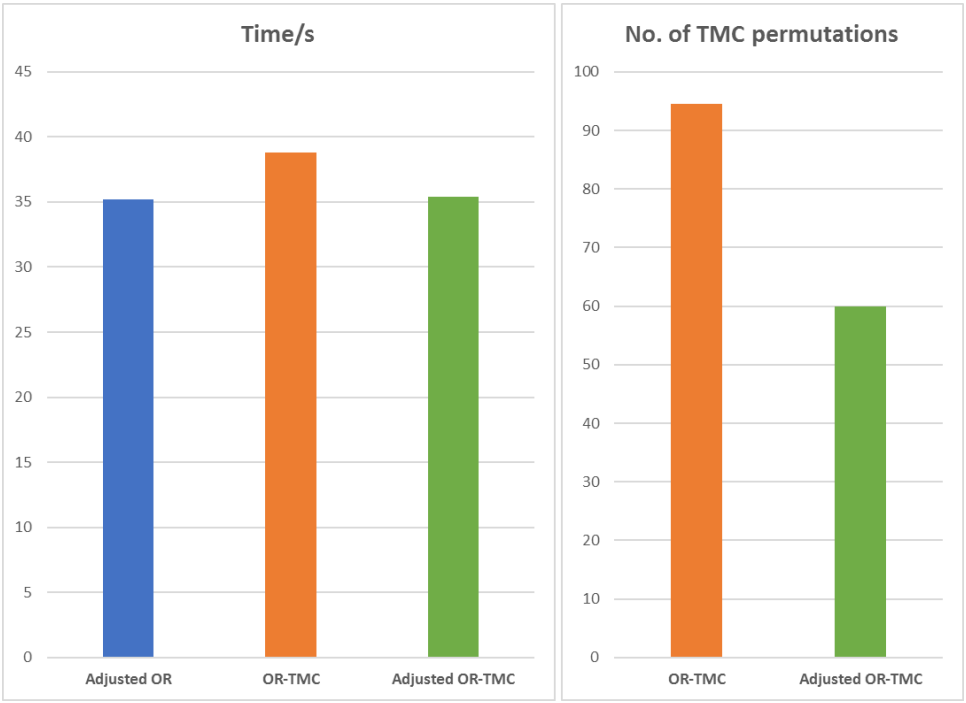

Figure 5-23: (Left) Average time to run different approximation algorithms for n=5 and T=10 for noised data on feature with same dataset size. (Right) Number of TMC permutations for OR-TMC and Adjusted OR-TMC to converge.

Adjusted OR-TMC performs best on accuracy, with the smallest Maximum Euclidean Distance and Average Euclidean Distance (Figure 5-24). Meanwhile, OR-TMC performs the worst.

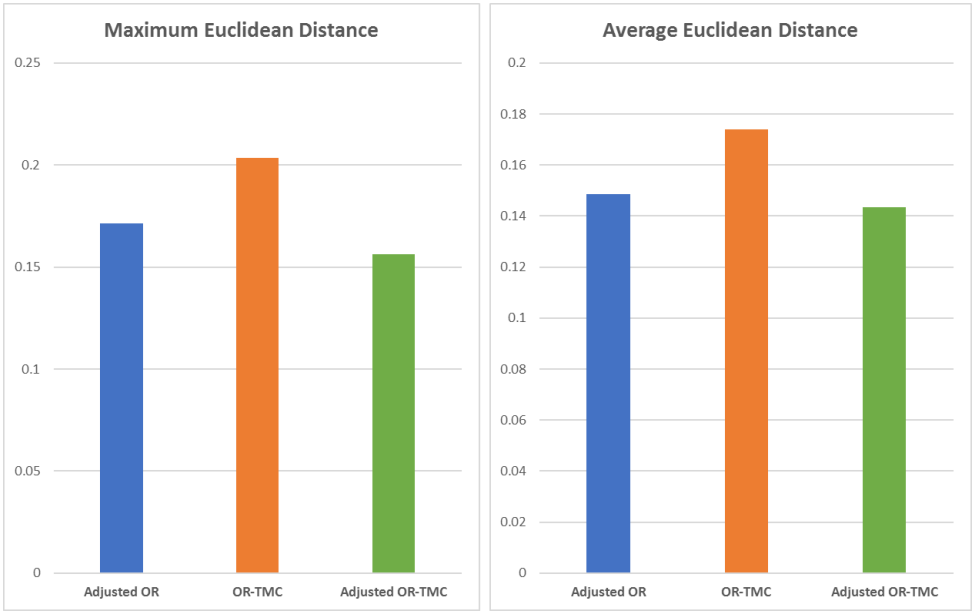

Figure 5-24: Maximum Euclidean Distance and Average Euclidean Distance for different approximation algorithms for *n*=5 and *T*=10 for noised data on feature with same dataset size.

Similar to 5.2.2.3, all three approximation algorithms have much worse accuracy than 5.2.2.1, but not as bad as 5.2.2.4. It may be easy to conclude that they perform better when there is noise on data features, than when there is noise on data labels. However, a closer look at the values generated (Table 5-5) suggests otherwise.

Table 5-5: Average Standardised SV produced by different algorithms for n=5 and T=10 for noised data on feature with same dataset size.

Algorithm	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
Exact Non-Federated Shapley	0.2027	0.2026	0.2016	0.1987	0.1944
Exact Federated Shapley	0.2023	0.2034	0.2008	0.1981	0.1954
Adjusted OR	0.1144	0.1310	0.2394	0.2665	0.2486
OR-TMC	0.0977	0.1342	0.2477	0.2787	0.2417
Adjusted OR-TMC	0.1257	0.1271	0.2355	0.2617	0.2499

There are different levels of noise in the datasets, with D_1 having no noise and D_5 having the highest level of noise (refer to Section 4.1 for more details). Both Exact Non-Federated Shapley and Exact Federated Shapley reflect this decreasing trend in the quality of the datasets D_i as i increases, although

ϕ_1 is again an anomaly for Exact Federated Shapley. However, the results generated by the three approximation algorithms do not follow this trend at all, with ϕ_i increasing when i increases from 1 to 4, before falling when $i = 5$. This may be because the OR model reconstruction step cannot approximate M_S accurately, as explained earlier in Section 3.1.2.

5.2.2.6 Overall Comments

All three approximation algorithms reduce the time taken to calculate SV significantly, from an average of 2772s for Exact Federated Shapley to below 40s for all cases.

Like the results in Section 5.2.1, Adjusted OR-TMC is almost as accurate as Adjusted OR while OR-TMC consistently has the worst accuracy. However, for this value of *n*, OR-TMC fails to reduce runtime by Adjusted OR, probably due to the large number of permutations required to converge. Adjusted OR-TMC generally reduces the runtime by Adjusted OR, but not as significantly as Section 5.2.1. This is because the number of data contributors is not large enough. For n=5, Adjusted OR evaluates 32 models, while Adjusted OR-TMC takes at least 20 permutations to converge, and in the process Adjusted OR-TMC may have to already evaluate a large proportion of these 32 models.

Like the Section 5.2.1, all three approximation algorithms have low accuracy on noised data. This further highlights the drawback in OR-based reconstruction method and is an area for future studies.

Chapter 6 Conclusions and Recommendations

6.1 Conclusions

This project seeks to improve upon the current algorithms in approximating Shapley Value in a horizonal enterprise Federated Learning setting. Adjusted OR manages to streamline OR and reduces time taken by about 5-8% while keeping the results the same.

OR-TMC incorporate the idea of TMC to further reduce runtime by Adjusted OR. Adjusted OR-TMC modifies OR-TMC to make it more suited to the enterprise FL setting and hence converges faster. When the number of contributors is at n=10, OR-TMC and Adjusted OR-TMC have been experimentally proven to cut down the runtime by as much as 40%. For a smaller number of contributors, such as n=5, their time-saving ability is less significant.

While reducing the time taken, Adjusted OR-TMC also manages to get similar accuracy as Adjusted OR, for both n=5 and n=10. On the other hand, OR-TMC usually has the worst accuracy. Adjusted OR-TMC hence is a more viable option that can replace OR as a time-efficient algorithm in approximating Shapley Values, especially for larger values of n and in cases that OR has been shown to work accurately such as case 1 (same distribution with same dataset size) or case 2 (different distribution with same dataset size). When the datasets of the data contributors have different levels of noise on labels or features, all the OR-based approximation algorithms are inaccurate in capturing the Shapley Values of data contributors.

6.2 Limitations and Recommendations for Future Work

The project only studies one model architecture: 2-layer fully connected Multilayer Perceptron. The results may not generalise well to other model architectures. In the future more model architectures can be studied to give more conclusive results.

The training task used is MNIST hand-writing classification, which may be too simple to assess the SV. As shown in Table 5-3, Table 5-4 and Table 5-5, the values calculated by Exact Non-Federated Shapley are very close even though the data quantity and quality vary significantly across the datasets. In fact, the models trained purely on datasets with the highest levels of noise (on label or on feature) still manage to get accuracy as high as 94%. This may explain why the Exact Federated Shapley algorithm does not strictly produce the expected trend in SV. Future work can use a harder learning task that requires larger quantity and quality of training data to get a high accuracy, allowing the trend to be more pronounced.

The number of local epochs, local minibatch size and learning rate are hyperparameters that are not studied. Changing them may lead to different results. Future work can investigate the impacts of these hyperparameters in the performance of the approximation algorithms.

The experiments are run on Google Colab. Although care has been taken to ensure a constant environment for all experimental runs, there may still be variability in resources allocated by the server. Future experiments can be done instead on a platform where we can specify the capabilities of the virtual machines.

Finally, other model approximations such as Multi-Round Model Reconstruction (MR) can be studied instead of OR. MR has been shown to have

higher accuracy than OR on noised dataset but also takes much longer time, since it calculates Shapley Values for every global iterations and averages them at the end. This process may be made faster by applying the same idea of TMC like how we manage to speed up OR.

List of References

[1] L. Tian, S. Anit Kumar, T. Ameet, and S. Virginia, “Federated Learning: Challenges, Methods, and Future Directions,” pp. 1–21, 2018.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: Concept and applications,” *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

[3] R. Jia *et al.*, “Towards Efficient Data Valuation Based on the Shapley Value,” vol. 89, 2019.

[4] A. Ghorbani and J. Zou, “Data Shapley: Equitable Valuation of Data for Machine Learning,” 2019.

[5] T. Song, Y. Tong, and S. Wei, “Profit Allocation for Federated Learning.”

[6] L. Yann, C. Corinna, and C. Burges, “MNIST handwritten digit database.” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 19-Dec-2019].

[7] European Union, “General Data Protection Regulation (GDPR).” [Online]. Available: https://gdpr-info.eu/issues/. [Accessed: 19-Apr-2020].

[8] L. Xhao and L. Xia, “China’s Cybersecurity Law: An Intro for Foreign Businesses,” 2018. [Online]. Available: https://www.china-briefing.com/news/chinas-cybersecurity-law-an-introduction-for-foreign-businesspeople/. [Accessed: 19-Apr-2020].

[9] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, “Communication-efficient learning of deep networks from decentralized data,” *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54, 2017.

[10] B. Faltings and G. Radanovic, “Game Theory for Data Science,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2017.

[11] L. S. Shapley, “A value for n-person games,” *Annals of Mathematical*

Studies, vol. 28, pp. 307–317, 1953.

[12] Giacomo Bonanno, “Cooperative Games: the Shapley Value.” [Online]. Available: http://faculty.econ.ucdavis.edu/faculty/bonanno/teaching/122/Shapley.pdf. [Accessed: 21-Apr-2020].