DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Optimised Private Set Intersection for Vertical Federated Tree Models

Author:
Martin LI
Supervisors:
Dr. Rihan HAI
Danning ZHAN

Thesis Advisor:
Dr. Christoph LOFI
Ext. Committee Member:
Jérémie Decouchant

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

January 15, 2024

Declaration of Authorship

I, Martin Li, declare that this thesis titled, "Optimised Private Set Intersection for Vertical Federated Tree Models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Martin Chun Ho Li

Date: 15-01-2024

“This thesis is dedicated to my father who just beat cancer.”

Martin Li

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science

Software Technology

Master of Science

Optimised Private Set Intersection for Vertical Federated Tree Models

by Martin LI

In recent years, the rapid advancements in big data, machine learning, and artificial intelligence have led to a corresponding rise in privacy concerns. One of the solutions to address these concerns is federated learning. In this thesis, we will look at the setting of vertical federated learning based on tree models. We have built a system that can do both entity resolution through private set intersection (PSI) and vertical federated learning (VFL). In this system, we have implemented an optimisation to pre-sort the data per feature before the start of VFL. We have also created a privacy framework, where we define four levels of privacy. This optimisation did not affect the privacy level of the system. In our results, we have seen that pre-sorting the data lowers the overall training time. How much depends on the number of entities and features of the passive party. We observe from our results that we estimate the speed-up to be 0.3654 seconds per feature and 0.2093 seconds per 1000 entities.

Acknowledgements

Completing this master's thesis has been a long journey, but I am very grateful to the people who have supported me during this time.

First and foremost, I would like express my gratitude to my thesis supervisor, Rihan Hai, for her insights, feedback, and patience. Also, I extend my gratitude to Danning Zhan, my other supervisor who joined me halfway, for guiding me through the technical aspects of the thesis. I am also thankful to the rest of the members of my thesis committee, Christoph Lofi and Jérémie Decouchant, for their time and feedback.

I would also like to thank my friends who were also always on campus working on their thesis, or other university works. Thank you for the helpful discussions and distractions. Thanks to you, I didn't have to work on this all alone.

At last, I want to thank my family for their support. The last couple of months weren't easy, but I'm thankful that everyone is healthy again and that they can see me hand in the thesis.

Contents

Chapter 1

Introduction

With the fast developments in the field of big data and machine learning in the last decade, there is a rising concern regarding privacy, since a major portion of the data that is used to train the machine learning models is from real people. To combat these concerns many regulations have been implemented on worldwide levels. Europe introduced the *General Data Protection Regulation* [EU, 2018].

The GDPR is one of the most comprehensive privacy laws in the world, and it has had a significant impact on the way that organizations collect, use, and store personal data. The GDPR gives individuals more control over their data and requires organizations to be more transparent about how they use data.

In 2017, Google was one of the first large corporations that introduced privacy-preserving machine learning, and they called it *Federated Learning* [McMahan and Ramage, 2017]. The main goal during the development of this technique was to address privacy and data security concerns. The idea behind federated learning is that a model is trained via multiple independent parties, where each party uses their local data. The local data will never have to be sent, as plain text, to a different party or a central server.

Instead, each party trains a local model on their data and then sends the updated model parameters to a central server. The central server then aggregates the updated model parameters and creates a new, global model. This new model is then sent back to the parties, who can then use it to update their local models.

In Figure 1.1 you can see Google's idea for federated learning. In step A, your phone trains a model on your local data. Many users' updates are aggregated in step B. The aggregation is used to form a consensus change, in step C, to the shared model. This process could be repeated indefinitely.

In traditional machine learning, data is gathered and stored on a central server, which can raise privacy issues if sensitive information is exposed or if the server becomes compromised. Federated learning, however, operates differently by keeping the data on local devices, like smartphones or IoT devices, and performing model training directly on these devices. Only the model updates, rather than the raw data, are transmitted back to the central server, ensuring user data remains private and secure.

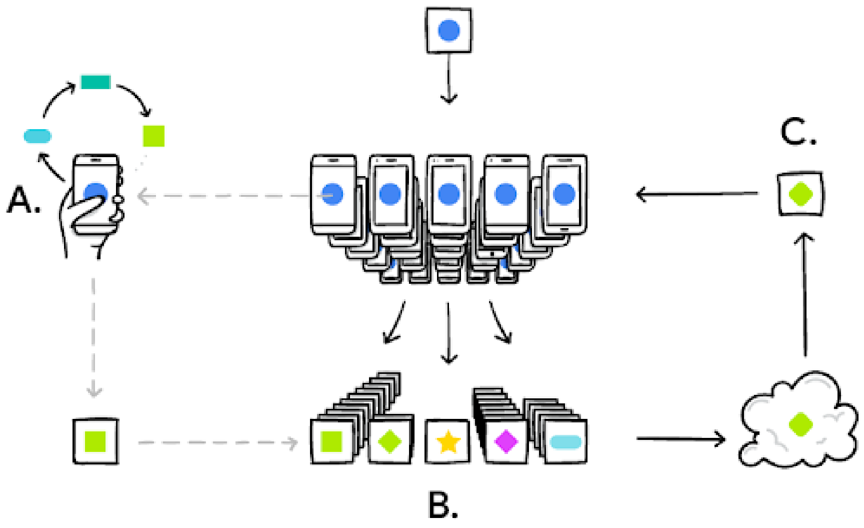
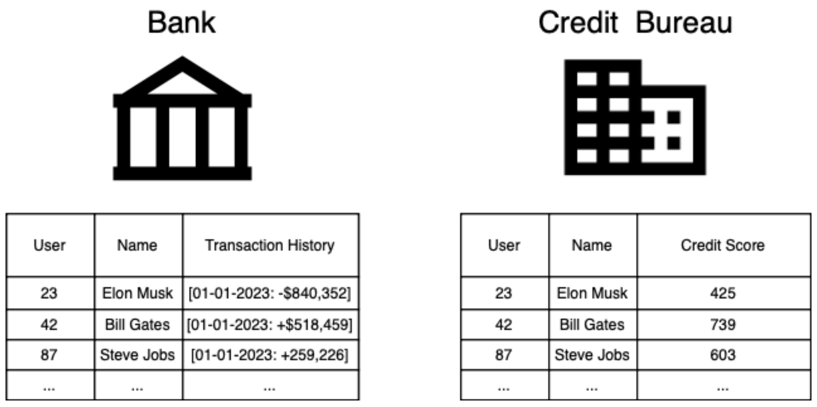


FIGURE 1.1: Google's original figure for Federated Learning (https://blog.research.google/2017/04/federated-learning-collaborative.html)

There are three different variants of federated learning: *Horizontal Federated Learning*, *Vertical Federated Learning* and *Federated Transfer Learning*.

In this thesis report, we will focus on the process of*Vertical* Federated Learning (VFL). In this variant, you can imagine tabular data that is split vertically. This means that when we consider two datasets, from different data owners, they have an overlap between the entities, but (almost) no overlap between the features. In this case, only one party has the target feature, and when training the model this party can leverage the extra features for each entity from the opposing party.


Bank
Credit Bureau

User	Name	Transaction History
23	Elon Musk	[01-01-2023: -$840,352]
42	Bill Gates	[01-01-2023: +$518,459]
87	Steve Jobs	[01-01-2023: +259,226]
...	...	...

User	Name	Credit Score
23	Elon Musk	425
42	Bill Gates	739
87	Steve Jobs	603
...	...	...

FIGURE 1.2: VFL setting between a bank and credit bureau

A real-life example of the application of vertical federated learning could be in the finance industry. VFL can be used to train a model for fraud detection, credit scoring, and risk assessment. A bank could collaborate with a credit bureau to train a model to detect fraudulent transactions. In figure 1.2, you can see how the data is partitioned. The bank would provide data on its customer's transaction history, while the credit bureau would provide data on consumer credit scores. Under most privacy and security laws these user data is not allowed to be shared with other organisations. With federated learning, this problem is largely solved, since no data is leaving the organisation's data silo unencrypted. The two parties can now train the model without sharing any customer-specific data in plain text.

For vertical federated learning, however, there is a problem that needs to be solved first: entity resolution. The bank and credit bureau need to make sure that their entities are correctly aligned, such that the first row in the data of the bank corresponds to the same first row in the data of the credit bureau. The first row of both data must refer to the same entity.

The need for private entity resolution is necessary for the vertical setting of federated learning. It ensures that data from different parties is correctly aligned, enabling the model to learn from the combined information without compromising privacy. Without entity resolution, the model would be trained on misaligned data, leading to inaccurate predictions.

However, since we are dealing with preserving privacy, the two parties can't simply send each other their data. So we need to find a method that can solve the entity alignment problem without compromising the data's privacy. One of the solutions for this problem is *Private Set Intersection (PSI)*.

1.1 Problem Statement

Definition 1. Private Set Intersection is a secure multiparty computation cryptographic technique that allows two parties holding sets to compare encrypted versions of these sets to compute the intersection. In this scenario, neither party reveals anything to the counterparty except for the elements in the intersection.

Definition 2. Vertical Federated Learning is a federated learning setting where multiple parties with different features about the same set of entities jointly train machine learning models without exposing their raw data or model parameters.

Given are two parties A and B, where both parties have their local data matrix $\{\mathbf{X} \in \mathbb{R}^{n \times d}\}$. Each row i in the matrix is a data instance, denoted as $\mathbf{X}_{i*} \in \mathbb{R}^{1 \times d}$. The two parties A and B have a different set of features and each party can hold a different set of data instances as well. Only one of the two parties can hold the label $\mathbf{y}$. The party with the label is often called the *active party*, whereas the party without the label is called the *passive party*.

In this thesis, we will work with tabular data and vertical federated learning based on tree models. Aligning entities and training a model are two sequential steps, where training the model can only start when the alignment is found. The goal of this thesis is to make the process of vertical federated tree models computationally faster by efficiently pre-processing the data.

Making the process of VFL faster solves a few disadvantages and issues that current systems deal with. These disadvantages include:

- Reduced productivity: The development and deployment might be delayed, and slow training speed leads to a limited ability to adapt to changing requirements.
- Increased costs: On cloud servers, slower training speeds mean more resource hours on the servers. On local servers, it means more energy and electricity costs.

1.2 Research Questions

In this thesis, we will create a prototype system that can do entity resolution through private set intersection and perform vertical federated learning. The pre-processing step of the data before VFL should optimise the VFL step in terms of execution time. Through this prototype, we will try to answer the following research questions:

RQ1. How does pre-sorting the data per feature affect the computation time of building vertical federated learning trees?
After researching the protocols behind vertical federated learning that are built around tree models, we have analysed the tree-building properties. I found that one of the steps in the algorithm is sorting the data per feature. This is done to find the local optimal split for each feature. In this thesis, we will conduct experiments to gain insights into how pre-sorting the data and removing the step for sorting affects the overall training speed of the algorithm.

RQ2. Do we maintain the level of privacy during PSI and VFL while introducing modifications to the system?
This question is a bit more complicated. Privacy is a complex concept and it is challenging to measure it precisely. For privacy, there is no universal agreed-upon metric. However, many researchers and experts in the field have developed frameworks and methodologies to assess and evaluate privacy in many different contexts. In this thesis, I will do the same. I will create a small framework which will follow my own assumptions and interpretation of privacy. The system that I will develop will then be tested against this framework to see whether and how the changes and modifications affect privacy.

1.3 Thesis Goals

The following points encapsulate the goals of this thesis:

G1. Optimise the pre-processing of data to reduce the computation time to build VFL trees.
Efficiently pre-processing the data before building VFL trees is a pivotal goal for this thesis, as it would reduce the overall training time. To fulfil this goal we examine the tree-building properties. One of those methods that we will specifically look at for this thesis, is by feeding the VFL process pre-sorted the data. By feeding it pre-sorted data, the sorting step for tree-building will be eliminated, which would reduce the overall training time.

G2: The level of privacy and security should not be reduced by the introduction of any modifications in the system
Privacy and security are, as mentioned earlier, not easy to quantify. There is no standard that we can use to tell whether the level of privacy has been increased or decreased. So in this thesis, we will define our interpretation. Based on our framework, we should conclude whether the introduced changes will affect the level of privacy.

G3: Create a proof-of-concept system that performs ER and VFL.
This last goal is a necessity that needs to be done to fulfil goals 1 and 2. A system needs to be built that can perform ER and VFL. I will not create these two processes from scratch, but instead, I will use existing frameworks and extend them.

1.4 Outline

In the next chapter (2), we will have a look at the preliminaries of this thesis. The chapter will tell you everything you will need to know before we head into the technical side of the thesis. In chapter 3, we dive into the state-of-the-art regarding private set intersection and vertical federated learning. In chapter 4 we present the implementation and architecture of our system. This system includes the optimisation of pre-sorting the data during pre-processing. In chapter 5, we describe the experimental setup, show the results and discuss said results. And in the last chapter (6), we will conclude this thesis by answering the research questions, and propose the future directions this thesis can take.

Chapter 2

Preliminaries

2.1 Classic Tree-based Machine Learning

In classical machine learning, tree-based models constitute a category of supervised learning algorithms used for both classification and regression tasks. These models construct a tree-like structure to determine the class or value of the target variable based on the input features. Tree-based algorithms are widely employed in the prediction of tabular and spatial/GIS datasets, making them a popular choice in the field of Machine Learning.

There are multiple forms of tree-based machine learning models:

1. Single Estimator: e.g. Decision Tree
2. Ensemble Methods and Bagging: e.g. Random Forest
3. Boosting: Adaptive and Gradient Boosting Machine: e.g. AdaBoost
4. Extreme Gradient Boosting: e.g. XGBoost

2.1.1 Single Estimator

The simplest form of a tree-based algorithm is a *Decision Tree*. A decision tree has a flowchart-like structure in which each internal node represents a test on an attribute (e.g. true or false), each branch represents the outcome of the test and each leaf node represents a class label. A decision tree could look like this:

2.1.2 Ensamble Methods and Bagging

One of the most popular ensemble learning algorithms is *Random Forest*. Ensemble learning methods are made up of a set of classifiers, e.g. decision trees, and their predictions are aggregated to identify the most popular result. So, the idea behind random decision trees is to combine the output of multiple decision trees to reach a single result. Random forest is an extension to the random decision trees, by combining the original algorithm with Breiman's "bagging" [Breiman, 1996] and random selection of features

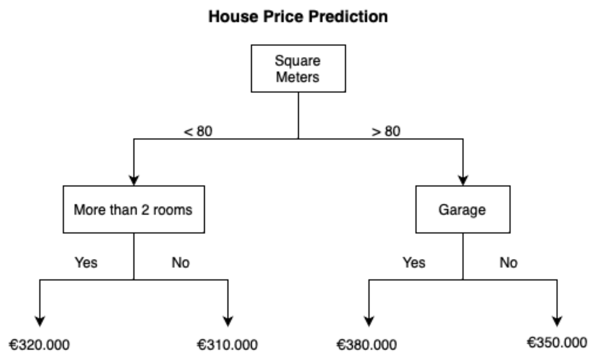


FIGURE 2.1: Decision Tree

to construct a collection of random decision trees with controlled variance. By majority vote of all trees, a final answer will be constructed.

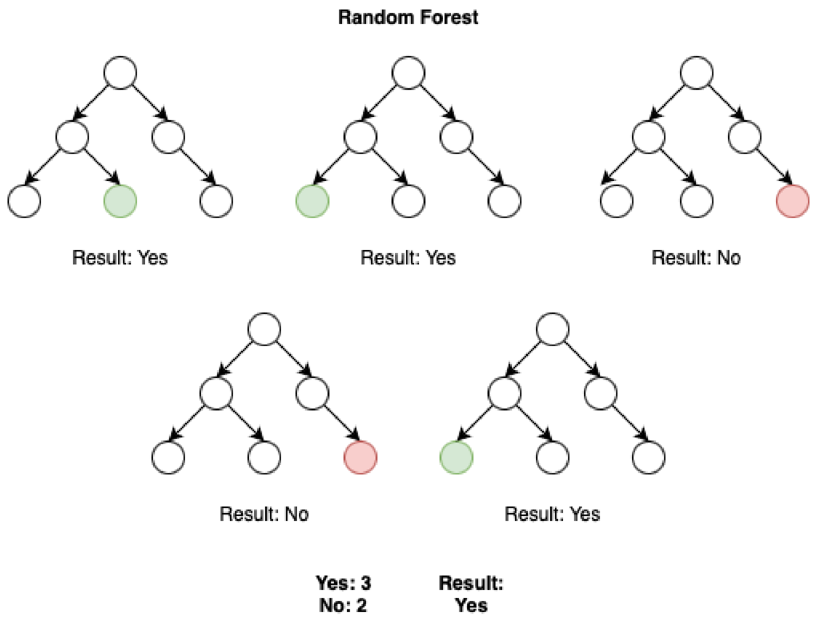


FIGURE 2.2: Random Forest

2.1.3 Boosting

The random forest algorithm creates several separate decision trees at the same time. Each of these models is independent of the others. In *Tree Boosting*, the trees are created sequentially. The first tree learns from the dataset. The second tree also learns from the same dataset and the inaccuracies of the first tree. Then the third tree learns from the same dataset and the inaccuracies of the previous tree and so forth, until it has reached the number of user-specified trees. Generally speaking, boosting has a higher accuracy than random forest, because each base model gets to learn from the original dataset and the inaccuracies from the previous trees.

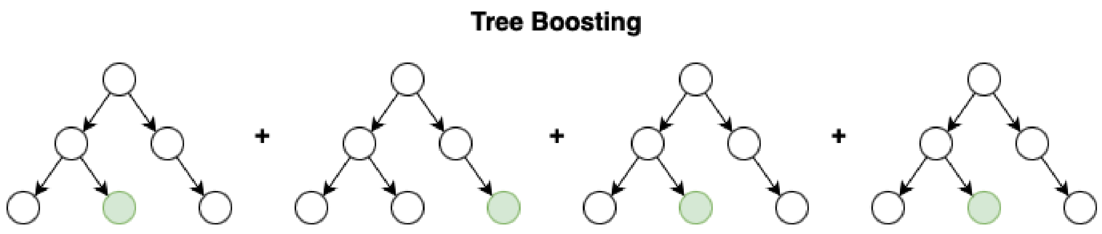


FIGURE 2.3: Tree Boosting

2.1.4 Extreme Gradient Boosting

Extreme gradient boosting, also known as *XGBoost*, is an optimised version of gradient boosting. XGBoost, like regular boosting, is built by creating multiple decision trees sequentially. The most significant difference is that XGBoost includes L1 (Lasso) and L2 (Ridge) regularization techniques. The two techniques help control the model complexity and it reduces overfitting. Other differences include parallel processing, handling missing data and tree pruning.

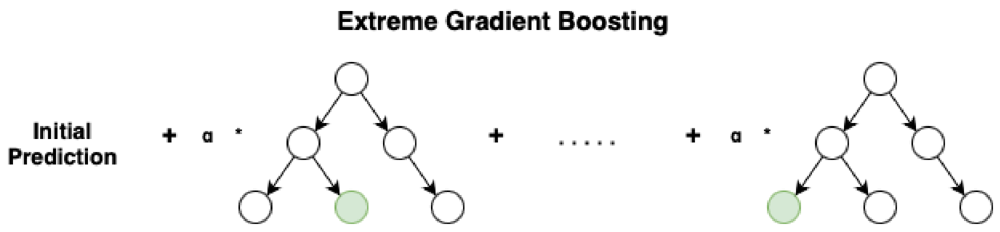


FIGURE 2.4: Extreme Gradient Boosting

Given a data set $X \in \mathbb{R}^{n \times d}$ with n samples and d features, XGBoost (Chen and Guestrin, 2016) predicts the output by using K regression trees.

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{2.1}$$

To learn the set of regression tree models used in equation 2.1, it greedily adds a tree f_t at the t-th iteration to minimize the following loss.

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i{}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.2}$$

where $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2$, $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_{(t-1)}} l(y_i, \hat{y}^{(t-1)}$

When constructing the regression tree in the t-th iteration, it starts from the tree with a depth of 0 and adds a split for each leaf node until reaching the maximum depth. In particular, it maximizes the following equation to determine the best split, where I_L and I_R are the instance spaces of left and right tree nodes after the split.

$$\mathcal{L}_{sp} = \frac{1}{2}\left[\frac{(\sum_{i \in I_l} g_i)}{\sum_{i \in I_l} h_i + \lambda)} + \frac{(\sum_{i \in I_R} g_i)}{\sum_{i \in I_R} h_i + \lambda)} + \frac{(\sum_{i \in I_I} g_i)}{\sum_{i \in I_I} h_i + \lambda)}\right] - \gamma \tag{2.3}$$

After it obtains an optimal tree structure, the optimal weight w_j^* of leaf j can be computed by the following equation, where I_j is the instance space of leaf j.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{2.4}$$

2.1.5 Summary

Tree-based machine learning models can be found in different variants. You have the very simple standard decision trees, and the complex gradient boosting decision trees, such as XGBoost. Each variant has its up- and downsides. The standard decision tree is very fast but the accuracy might be very low. XGBoost is known to score very high accuracies in a lot of machine learning competitions, but training a model can take a very long time. The fundamental idea for tree models does stay the same for every variant: to learn a set of rules that can be used to classify or predict the target value of a new data point. The decision-making process is represented as a tree-like structure, with each node representing a decision or evaluation of a feature variable and each branch leading to one of the possible outcomes. The ultimate results, represented by the tree's leaves, are the predicted target values.

2.2 Private Set Intersection

Private Set intersection (PSI) is a cryptographic technique for computing the intersection of sets held by two or more parties. The intersection is found without revealing the elements that are not in the intersection. Apart from its theoretical significance, this secure multiparty computation has seen several real-life use cases:

- *Finding Contacts* - Many messaging services (e.g. WhatsApp, Telegram) rely on phone numbers, stored locally on your mobile phone, to find other contacts that are using the same service. PSI allows users to share their contacts with these messaging providers without sharing their complete list of contacts.
- *Covid-19 Contact Tracing* - During the COVID-19 pandemic, PSI was used in most government mobile apps to track whether people came into close contact with someone who was infected with the coronavirus without revealing the person's location history.

For real datasets, the unique common identifiers would serve as the input for private set intersection. By using these identifiers, the active and passive parties can compute the intersection privately. By verifying which identifiers are in the intersection, they can determine which entities they have in common.

The most common setup for PSI is the server-client setup. In this scenario, the client wants to request the intersection between their local data and the data stored on the server. For the client the input is $X = x_1, ..., x_n$ and the server's input is $Y = y_1, ..., y_n$. The output for the client is $X \cap Y$ and the output for the server is $\emptyset$. Note: the client only learns the intersection of the two sets, none of the server's data points outside of the intersection is revealed to the client.

Let's have a look at an example:

- The client has the following elements in their data: [2, 10, 4, 7, 13, 9]
- The server has the following elements in their data: [3, 13, 10, 6, 7, 5]
- These two lists will be the input for PSI
- The output given to the client will be: [10, 7, 13]

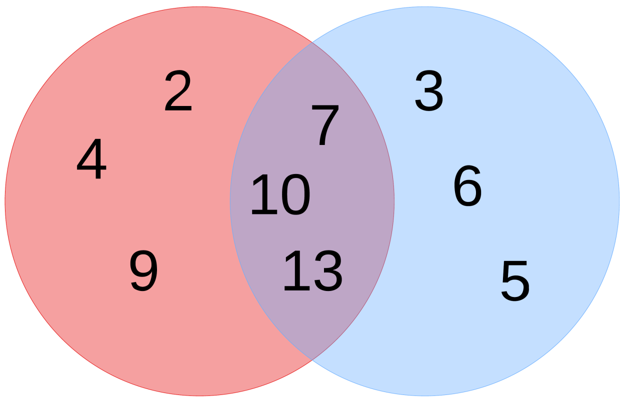


FIGURE 2.5: Venn Diagram of PSI

To understand how our protocol of PSI computes an intersection privately, some concepts need to be explained first: cuckoo hashing, oblivious transfer and pseudo-random functions. These are our building blocks for securely finding an intersection between two sets.

2.2.1 Cuckoo Hashing

Cuckoo hashing is a table data structure used for fast key-value storage and retrieval. It is designed to address some of the limitations of traditional hash tables, such as collisions and the need for resizing. The name is derived from a cuckoo bird's behaviour, where it sometimes kicks its cuckoo chicks out of their nests when new eggs hatch. Analogously, inserting a new value into a cuckoo hashing table may kick an older value out of the table.

In Cuckoo hashing, there are typically two hash functions, H_1 and H_2, and two separate hash tables, T_1 and T_2. When inserting a value k, it will apply $H_1(k)$ and $H_2(k)$ which will result in a position in T_1 and T_2, respectively. If the cell is empty in T_1, we can simply insert k into this cell. In the case that the cell is not empty, it will look at the cell in T_2 and place it there if it is empty.

If both cells are occupied, the current value k' in T_1 will be kicked out and k will be inserted in T_1. k' will be hashed again using H_1 and H_2 and look for an empty cell. It will again kick out any current occupants, and this process will be repeated until an empty cell is found. An infinite loop may occur. In this case, all elements will be removed from the tables, and everything will be rehashed using new hash functions.

2.2.2 Oblivious Transfer

Oblivious transfer (OT) is a two-party protocol involving a sender and a receiver. In this protocol, the sender transfers information to the receiver while remaining completely unaware of the specific information that the receiver ultimately acquires.

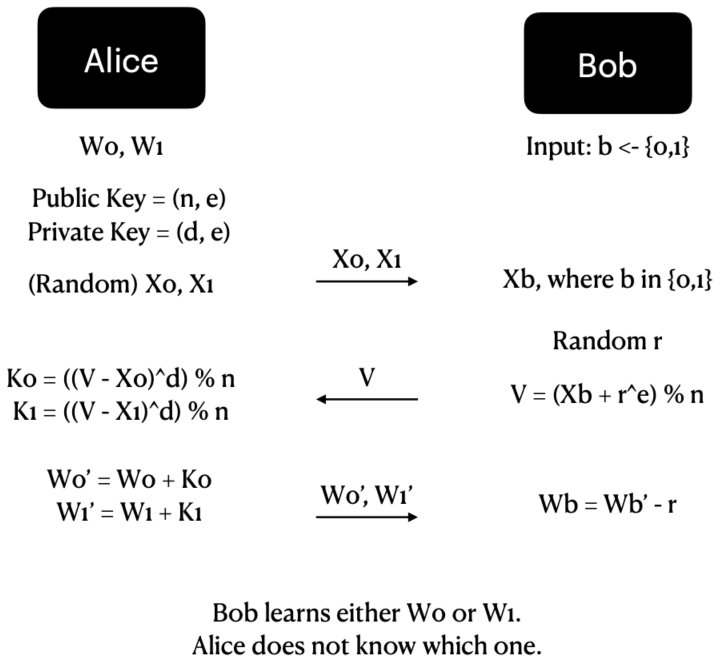


FIGURE 2.6: Oblivious Transfer

In this example, Alice has two values W_0 and W_1. Bob wants to learn one of the two values, but he does not want Alice to know which of the two he learns. Alice generates a public and private key and sends to Bob two random values X_0, X_1 and the public key (n, e). Bob chooses X_b, where b is either 0 or 1. Bob will send V back to Alice, where V is equal to $(X_b + r^e) \bmod n$. Alice can now calculate the two values K_b, where b is 0 and 1, and where

$K_b = ((V - X_b)^d) \bmod n$. Now Alice calculates two values $W'_0 = W_0 + K_0$ and $W'_1 = W_1 + K_1$ and send them over to Bob. Bob can now learn the value that he wanted to learn at the very beginning by calculating $W_b = W'_b - r$.

2.2.3 Pseudo-Random Functions

Pseudo-random functions (PRF) are, as the name suggests, functions that give a pseudo-random answer. Meaning it's a deterministic function that appears to produce a random output, even though it is entirely deterministic and predictable when given the same input. One of the inputs for such functions is usually a seed or key and the output is constructed in such a way that it appears indistinguishable from truly random data. Some key properties of pseudo-random functions include:

- Deterministic: PRF's produce the same output for the same input every time
- Pseudorandomness: The output of a PRF should be statistically indistinguishable from true randomness to an observer who doesn't know the secret key used as input
- Efficiency: PRFs should be computationally efficient to calculate, especially in practical cryptographic applications

2.2.4 Oblivious Pseudo-Random Functions

When combining Oblivious Transfer and Pseudo-random Functions, we get *Oblivious Pseudo-Random Functions* (OPRF). OPRF is a protocol between two parties, where one party has the key k for a secure pseudo-random function F. The other party has an input x for the function F, and at the end of the protocol, it learns $F(k, x)$ and nothing else.

2.2.5 Summary

The building blocks for our PSI protocol consist of cuckoo hashing, oblivious transfer and pseudo-random function. They are all necessary for creating the protocol that can securely compute the intersection between two datasets. We only look at the common unique identifiers of the datasets. By extracting that column from both datasets, we have two lists of id's. By using those two lists as input for our PSI protocol, each party will receive the intersection without knowing which other id's there were in the opposing dataset.

2.3 Federated Learning Settings

Federated Learning (FL) is a special form of machine learning. In classical machine learning, all data that is required is stored in one place. A model can access this data and train itself using the data. In the past, when two organisations collaborated with each other they could send all of their data to one big data silo. A model could then be trained on this big data silo as seen in 2.7.

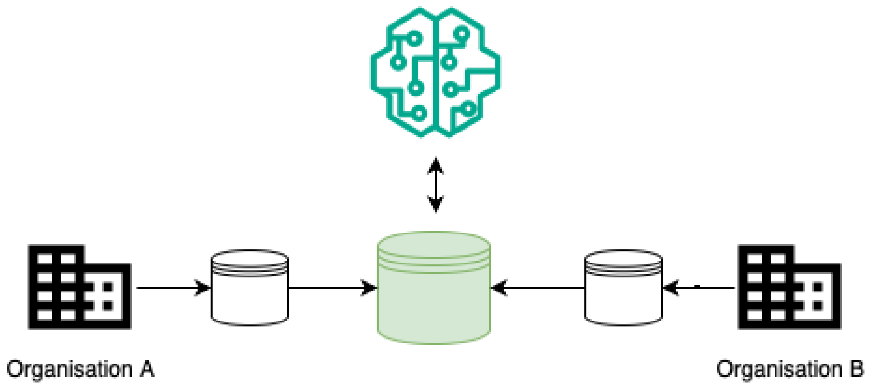


FIGURE 2.7: Traditional Machine Learning

In the past, when organizations A and B collaborated to train a model, they could merge their data into one big data silo without much concern for data security or privacy. All data, whether it was sensitive or not, would often be stored in plain text, easily accessible by anyone with access to the data silo. This approach did allow for seamless data integration and it would greatly facilitate the training of models with large, diverse datasets.

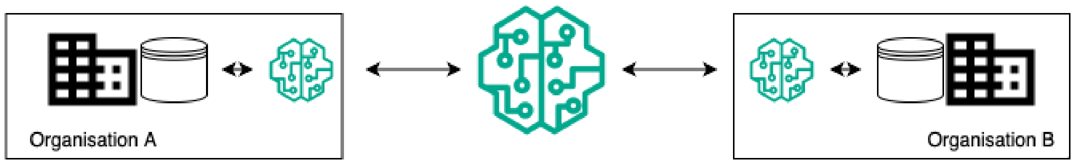


FIGURE 2.8: Federated Learning

However, nowadays, when organizations A and B collaborate on model training, they must adhere to privacy regulations, such as the GDPR. Data must be treated with the utmost care, and encrypting data is no longer an option but a necessity. This kickstarted organizations to employ techniques like federated learning, where data remains decentralized, and models are trained collaboratively without the need to share raw, sensitive data.

Federated learning allows organizations to benefit from each other's data while respecting regulations. Instead of sending data to a central data silo, models are trained on local data. Only model parameters and/or gradients are exchanged. This approach preserves the privacy of individual data points while still allowing for the collective improvement of the models.

There are three settings of federated learning: horizontal, vertical (Yang et al., 2019) and federated transfer learning (Liu, Chen, and Yang, 2018). These three are characterised by how the data is partitioned.

2.3.1 Horizontal Federated Learning

In Horizontal Federated Learning (HFL), the dataset is partitioned horizontally, meaning that the feature space overlaps, but there are (almost) no samples that can be found in both datasets. In figure 2.9, we see a dataset that is partitioned horizontally. Dataset 1 has information about entities U1, U2 and U3. Dataset 2 has information about U4, U5 and U6. Although the two datasets have information about different entities, they do keep track of the same features. They also both have information about the target value. The "virtual dataset" on the right of the figure gives an impression of how the dataset would look if it were to be combined. In horizontal federated learning, the sample space will be increased

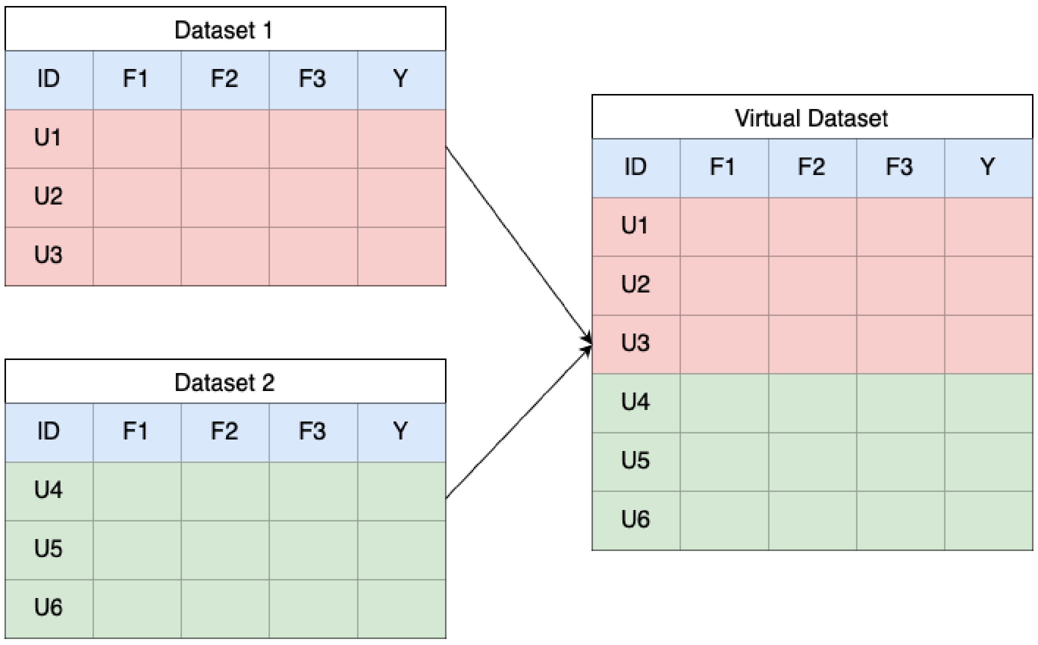


FIGURE 2.9: Horizontal Federated Learning

2.3.2 Federated Transfer Learning

In the setting of Federated Transfer Learning (FTL), there are not a lot of samples and features that overlap. Transfer learning can then be applied in a federated manner. In Figure 2.10, we see that there is some overlap in the entities as well as in the features, however, there will be some values missing, since there is no information. In the figure, these cells are filled with black. This is where transfer learning is used. Transfer learning reuses a pre-trained model on a new problem. The model exploits the knowledge gained from a previous task to improve generalization about another. In federated transfer learning, we can increase the feature and sample space.

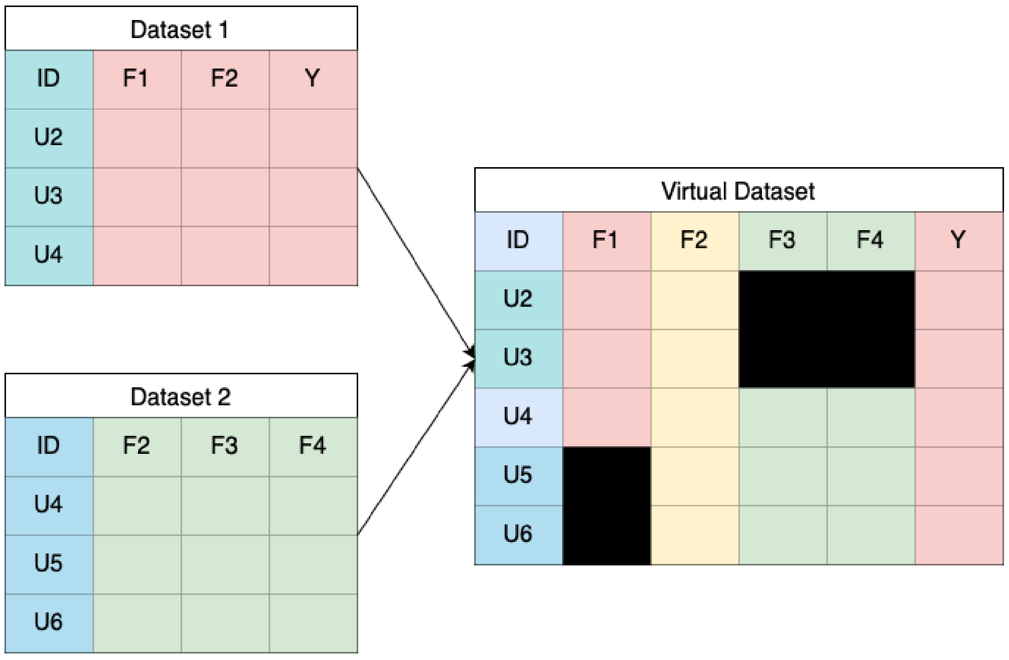


FIGURE 2.10: Federated Transfer Learning

2.3.3 Vertical Federated Learning

In Vertical Federated Learning (VFL), the dataset is partitioned vertically. The samples in both datasets have a lot of overlap, but the two datasets have different features. In Figure 2.11, you can see that both datasets have information about entities U1, U2, and U3. However, the datasets have different features, meaning they track different information about the same entities. Also, only dataset 1 has the target label (Y). This is what makes VFL challenging since the party without a target label has to find a way to contribute to the model without compromising the data. Vice versa, the same challenge also applies to the party with the target label. Leaking the target label might lead to a serious violation of privacy. In vertical federated learning, the feature space will be increased.

In VFL the data is based on features rather than individual data points as seen in Figure 2.12. This means that each participating party in the vertical federated learning process holds only a subset of the features for the same set of data points, and the objective is to collaboratively train a machine learning model without sharing the raw data. By vertically partitioning the data, each party retains control over their data while still contributing to the model's training.

No confidential information is allowed to be shared directly between the parties. Each party has a local model and only encrypted intermediate results are allowed to be exchanged. In every iteration, the two parties will collaboratively update the global model. The global model will then be the starting point as the local model for each party. These iterations will continue, as long as the parties want, since new data can be added.

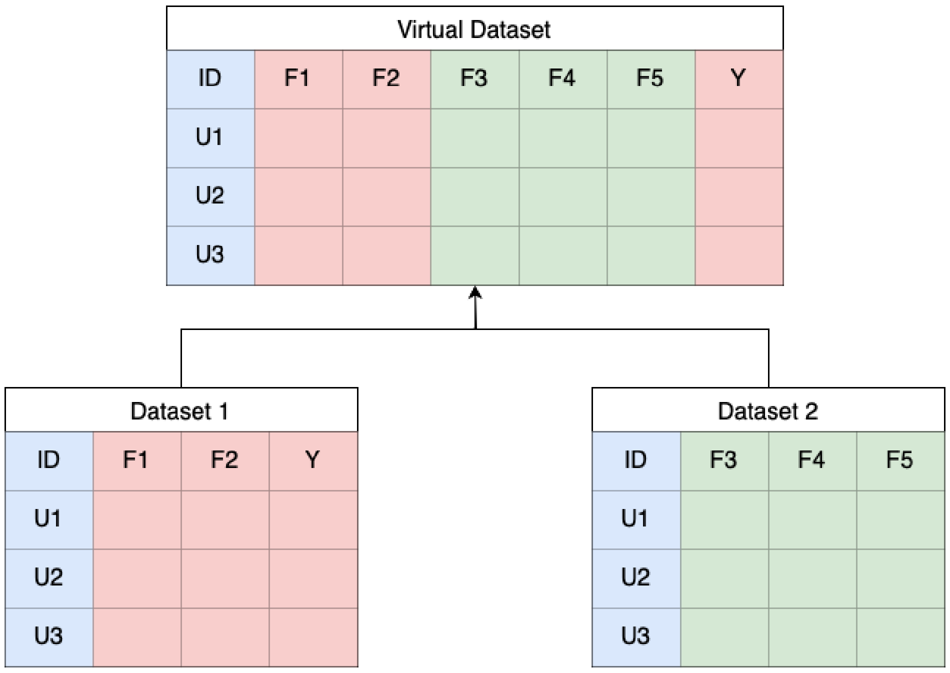


FIGURE 2.11: Vertical Federated Learning

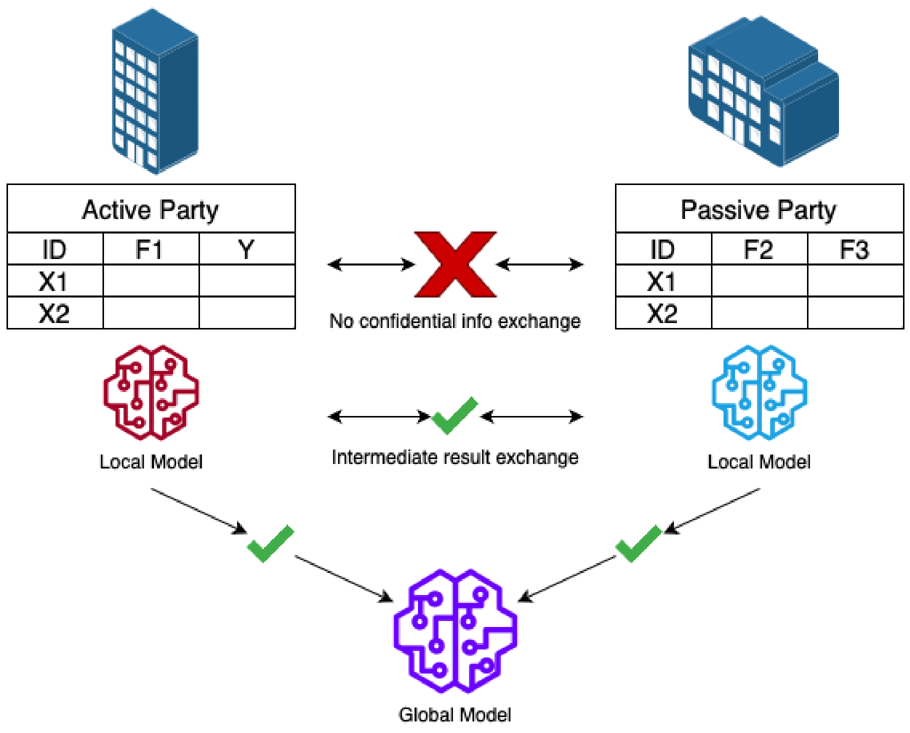


FIGURE 2.12: VFL Setting

Many different underlying machine learning models can be used for VFL, such as neural networks and tree-based models. In this thesis, we will focus on tree-based models. Tree-based models are a class of machine learning algorithms used for both classification and regression tasks. These models are based on the concept of decision trees, which mimic human decision-making processes by making a series of decisions based on input features to reach an outcome.

In traditional machine learning with a tree-based model, all features and data are available in plain text. This makes it easy to calculate the optimal split point for each feature. But, in a VFL setting, the passive parties don't have access to the target label. To overcome this challenge, VFL relies on privacy-preserving techniques such as secure multi-party computation, homomorphic encryption, or differential privacy. These techniques allow the parties to perform computations on the shared features while keeping the target labels hidden. In this way, the different parties can collaborate with each other to build a decision tree without revealing their data.

2.3.4 Summary

Federated learning is a powerful protocol in a world where privacy and security around data are extremely important. Federated learning is still a relatively new technology, but it has the potential to revolutionize the way that machine learning is used in a variety of industries. It is particularly well-suited for applications where data privacy and security are important, such as healthcare, finance, and government.

Vertical federated learning is a variant of federated learning where multiple parties with different features about the same set of entities jointly train machine learning models. In the setting of VFL, each participant has a unique subset of features for the same set of entities. Often, only one party has the target label. For example, one party might have demographic data, such as age and gender, while another participant might have behavioural data, such as purchase history, website visits and the target label to predict how likely someone will buy a certain product. By combining their data, participants can train a more accurate machine learning model than they could on their own.

2.4 Homomorphic Encryption

Homomorphic Encryption (HE) is a specialised form of encryption that allows computations on encrypted data without decrypting it. In traditional encryption schemes, it is not possible to correctly manipulate encrypted data. It needs to be decrypted first before any correct computations are possible. This means that through HE you can keep your data secure while computation on the encrypted values is still possible.

You can encrypt your data and then perform computations on the encrypted data. The results of the computations will also be encrypted, but you can then decrypt the results to get the final answer. An example of homomorphic encryption would be:

- Let x be the number "10"
- Let y be the number "20"

- Now we encrypt x and y homomorphically
- $[[.]]$ denotes the encrypted value
- We get $[[x]]$ and $[[y]]$, respectively
- It is now possible to add these two encrypted values together
- $[[x]] + [[y]] = [[x + y]] = [[10 + 20]] = [[30]]$
- After decrypting $[[10 + 20]]$, we correctly get 30

Suppose you have two numbers, x and y, and you want to add them together. You can encrypt x and y using homomorphic encryption. Then, you can perform the addition operation on the encrypted data. The result of the addition operation will also be encrypted. Finally, you can decrypt the result to get the final answer, which is the sum of x and y.

There are three types of HE and it depends on the necessary context:

- Partially Homomorphic Encryption: Supports one type of mathematical operation, either addition or multiplication, on encrypted data.
- Somewhat Homomorphic Encryption: Supports both addition and multiplication operations on encrypted data but has limitations on the number of operations or the depth of computation that can be performed.
- Fully Homomorphic Encryption: Supports an arbitrary number of addition and multiplication operations on encrypted data, making it the most versatile but also the most computationally intensive form of homomorphic encryption.

2.4.1 HE application for FL

Homomorphic encryption is a cryptographic scheme that is used in several federated learning systems. In federated learning, the gradient exchange between the users and the server may leak private information. Homomorphic encryption can solve this problem very well, it can deal with the encrypted model without affecting the training results of the model. This is important for federated learning because it allows participants to participate in the training of the model without having to worry about their data being compromised. It allows participants to keep their data private, which is important for sensitive data. Some benefits of using HE in a federated learning setting:

- Privacy: Homomorphic encryption ensures that the data of each participant remains private, even from the central server that is training the model. This is important for sensitive data, such as healthcare data or financial data.
- Security: Homomorphic encryption also helps to protect the security of the federated learning system. If an attacker can gain access to the

encrypted data, they will not be able to decrypt it and view the original data.

2.4.2 Summary

Homomorphic encryption is a form of cryptography that allows for the computation of encrypted data. Data that is homomorphically encrypted can still be used for mathematical operations, such as additions and multiplications. There are three different types of HE: partially homomorphic encryption, somewhat homomorphic encryption, and fully homomorphic encryption. There is no "best" type since it depends on your own system's needs and each type has its drawbacks and benefits. Homomorphic encryption is also used in federated learning systems. HE protects the intermediate data that is transferred between parties. Participants of FL can assume that their data stays secure and private, while still contributing to improving a collaborative model. Some benefits of using HE in an FL setting include privacy, security and scalability.

Chapter 3

Related Work

3.1 PSI Protocols

Private Set Intersection (PSI) allows two parties to compute the intersection of their set in a privacy-preserving way. In this scenario, neither parties reveal their full set to the other party except for the elements in the intersection. There are many variants of PSI, all based on different encryption methods, such as Rivest-Shamir-Adleman (RSA) (De Cristofaro and Tsudik, 2010) and Diffie–Hellman (DH) (Huberman, Franklin, and Hogg, 1999).

There are many different implementations for a private set intersection protocol. Each with a slight variation on how they compute the intersection. In this chapter, we will have a look at different underlying concepts.

3.1.1 Homomorphic Encryption-based PSI

Just as in VFL, there are private set intersection protocols based on the implementation of homomorphic encryption [(Acar et al., 2018)]. Some protocols rely on additive and multiplicative HE [(Will and Ko, 2015; Cheon et al., 2017)]. One of the papers that based their PSI protocol on HE is "Fast Private Set Intersection from Homomorphic Encryption" [Chen, Laine, and Rindal, 2017]. They use fully HE to construct a fast PSI protocol. In the setting where one set is much smaller than the other one, that is where their work shines the most. The execution speed is drastically reduced, due to a smaller communication overhead in comparison to other PSI protocols.

The protocol is based on the observation that given two sets of data encrypted using HE, it is possible to compute the intersection of the two sets without ever revealing the underlying data and the output has no other data other than the intersection itself.

3.1.2 Bloom Filter-based PSI

There are also PSI systems based on Bloom Filters [Inbar, Omri, and Pinkas, 2018], where a lot of the communication overhead can be reduced. A Bloom Filter [Patgiri, Nayak, and Borgohain, 2018] is a space-efficient probabilistic data structure that tests whether an element is present in a set. This is done by hashing. A hash function takes input and outputs a unique identifier of fixed length which is used for identification of the input. Because of the probabilistic nature of the data structure, the filter may generate false positives, meaning that the filter says that a certain element is in the set, while it is not. For PSI this is not ideal, since we expect an exact set for the intersection.

3.1.3 State of the Art

As mentioned in 2.2.4, the state of the art uses a combination of cuckoo hashing, oblivious transfer and pseudo-random function ((Pinkas, Schneider, and Zohner, 2018)). In this protocol, a receiver has an input r. The sender gets an output s and the receiver gets $F(s, r)$, where F is a pseudo-random function and s is a random seed. The hashing is done with cuckoo hashing. One of the main advantages of cuckoo hashing is its ability to resolve hash collisions efficiently. The combination of these concepts results in a private set intersection protocol that is both fast and secure.

3.2 Federated Learning

Federated learning (FL) (Zhang et al., 2021) is a kind of encrypted distributed machine learning technology, in which participants can build a model without disclosing the underlying data so that the self-owned data of each enterprise does not leave the local database. Through the parameter exchange under the encryption mechanism, a virtual common model is established. With FL a machine learning model is trained without sharing data, thus allowing to address critical issues such as data privacy, data security, data access rights and access to heterogeneous data. Formally, it is assumed that N parties $\{P_1, ..., P_n\}$ own their own database $\{D_1, ..., D_2\}$. Each party can not access the data from the other parties. Federated learning is to learn a model by collecting training information from each party. The basic steps are:

1. The server sends the initial model W' to each participating party.
2. Each party P_i trains the model with their local data D_i.
3. The server aggregates the collected local models $\{W_1, ..., W_n\}$ to the global model W'
4. The server sends the updated global model W' to each party.
5. The parties replace their local model with the updated global model.

Although none of the local data leaves the party, there are still measurements needed to not leak the data. In the past, *anonymised* data from Netflix could be broken (Narayanan and Shmatikov, 2006), and local models for FL could be reverse-engineered to be exposed as the initial data.

3.2.1 FL Models

Neural Network Models

A neural network (Hopfield, 1982) is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial. Neural networks have been applied in training unmanned aerial vehicles, commonly known as drones, in a federated manner (Zeng et al., 2020). Normally training one drone using a neural network is not a problem, but training a group of drones is harder, because of the lack of continuous communication between the ground base group and the drones. Federated learning can be applied here by assigning one of the drones as the leading drone. Each drone can then train its local model and send its updates to the main drone. The main drone can then update the global model, which then can be distributed between all drones.

Linear Models

The three main categories for federated linear models are linear regression, ridge regression and lasso regression. Since linear models are the more "simple" models, the accuracy of models in a federated manner is already the same as non-private solutions (Du, Han, and Chen, 2004). For ridge regression, a federated system (Nikolaenko et al., 2013) has been proposed based on homomorphic regression and Yao's protocol (Lindell and Pinkas, 2009) that can almost get the same result as traditional models.

BlindFL [Fangcheng Fu, 2022] is a paper that introduces a state-of-the-art VFL system that is based on linear algebra models. They found that current VFL frameworks either support limited kinds of input features or suffer from potential data leakage during the federated execution. This paper provides a solution for the limited kinds of input by introducing federated source layers. It efficiently supports dense, sparse, numerical and categorical features. This paper also proposes a new security method by combining homomorphic encryption with secret sharing.

Secret sharing is a cryptographic technique for dividing a secret into multiple shares, such that the secret can only be reconstructed by combining a sufficient number of shares. Secret sharing is often used to distribute encryption keys so that no individual can compromise the entire key. In this paper secret sharing is used to protect sensitive data.

Tree-based Models

Federated Learning can be used to train one or multiple decision trees, such as random forests and gradient-boosting decision trees. Gradient Boosting Decision Trees (GBDT) is a widely used method in the field of machine learning, due to its great performance in classification and regression tasks. The first federated system that utilised this was mentioned in (Zhao et al., 2018). The system was used for data mining, and its security was based on Differential Privacy.

One of the reasons to create a vertical federated learning system based on tree models for this thesis is its good performance on tabular data. The paper "Why do tree-based models still outperform deep learning on tabular data?" (Grinsztajn, Oyallon, and Varoquaux, 2022) explains why tree-based models are still the preferred model for tabular data: Tree models are better at learning irregular patterns in data. Uninformative data affect deep learning a lot more and tabular data is non-invariant by rotations, and so are tree models.

3.3 Tree-based Vertical Federated Learning

This chapter will have a deeper focus on tree-based vertical federated learning systems and their underlying security protocols since it is the main focus of this thesis. We will have a look at the current state-of-the-art system regarding tree models in a vertical federated setting.

SecureBoost One of the first tree-based vertical federated frameworks is SecureBoost [Cheng et al., 2019] which utilises XGBoost [Chen and Guestrin, 2016] to train their model. In their paper, they let different parties securely collaborate on building decision trees. By analysing the math behind XGBoost, they figured out that only certain values from the target column need to be encrypted to find the optimal split points for each feature from the passive parties.

The system's security protocol relies on homomorphic encryption. The sensitive data that is communicated between the two parties is homomorphically encrypted, such that computations are still possible.

This is also the system on which we will build upon in this thesis. In chapter 4.1.1, we will dive deeper into the algorithm behind this system.

FederBoost The paper Federboost [Tian et al., 2020] proposes a vertical federated GBDT algorithm. They claim their approach doesn't need any cryptographic operations. Their security protocol is based on the concept of Differential Privacy (DP).

DP [Dwork, 2008] is a relatively new concept first proposed in 2006 to solve the problem of privacy disclosure in statistical databases. With DP, the results from the calculation of the database are insensitive to the changes of a specific record. A single record in the dataset or not in the dataset has little impact on the calculation result. Therefore, the addition of a single record will not increase the risk of privacy disclosure. Also, the risk can be controlled in a very small and acceptable range. An attacker cannot obtain accurate individual information by observing the calculation result. It tries to achieve the same as adding noise to data for traditional machine learning and deep learning. In practice, Lacplace mechanisms and exponential mechanisms are used to achieve differential privacy protection. Two aspects needs to be balanced, protection and validity. Adding too much noise affects the validity and adding too little decreases the level of protection.

VF^{2}Boost VF^{2}Boost [Fu et al., 2021] is a paper that builds upon SecureBoost and it introduces two optimisations. The first one is concurrent training. They found that while either party is encrypting their data, the other party is idle. Since they are using homomorphic encryption this could take a very long time. They introduce a blaster-style encryption method, which reduces the time that either party is idle. The second one is a customised cryptography algorithm which speeds up the encryption and decryption.

The underlying security technique is still the same as SecureBoost. The system makes use of homomorphic encryption to safely communicate sensitive data between the involved parties.

Chapter 4

Approach

The approach consists of two parts: private set intersection and tree-based vertical federated learning. Before training a model through vertical federated learning between two parties, the data has to be aligned between them. We will use private set intersection to find the intersection between two datasets. For our approach, we assume the following setting:

- Two parties
- Active party: The party's dataset has the class label
- Passive party: The party's dataset does not have the class label
- Tabular data
- Data between the parties have overlapping entities
- Data between the parties have almost no overlapping features

4.1 Tree-based VFL

4.1.1 The Protocol

The prototype is based on the implementation of the paper: "SecureBoost: A Lossless Federated Learning Framework" (Cheng et al., 2019). SecureBoost is a lossless vertical federated learning system based on the XGBoost algorithm (Chen and Guestrin, 2016).

Federated Tree Building

By analysing the math behind XGBoost, which can be found in 2.1.4, the authors make the following observations:

1. The evaluation of split candidates and the calculation of the optimal weight of a leaf only depends on g_i and h_i
2. The class label can be inferred from g_i and h_i.

Following the first observation, passive parties can find their local optimal split for each feature with gradient g_i and hessian h_i. According to equation

2.3, the optimal split can be found if $g_l = \sum_{i \in I_l} g_i$ and $h_l = \sum_{i \in h_l} h_i$ are calculated for every possible split. However, according to the second observation, g_i and h_i are sensitive data since the class label can be inferred from them. The active party is therefore required to keep these values confidential by encrypting them before sending them to the passive party. The encryption is possible with homomorphic encryption, such that we can do computations with the values.

First, the passive party computes $Enc(g_l)$ and $Enc(h_l)$ for all possible splits locally. These values will be sent to the active party. The active party can then decipher all $Enc(g_l)$ and $Enc(h_l)$ and calculate the global optimal split by adopting the approximation scheme used by the original XGBoost algorithm:

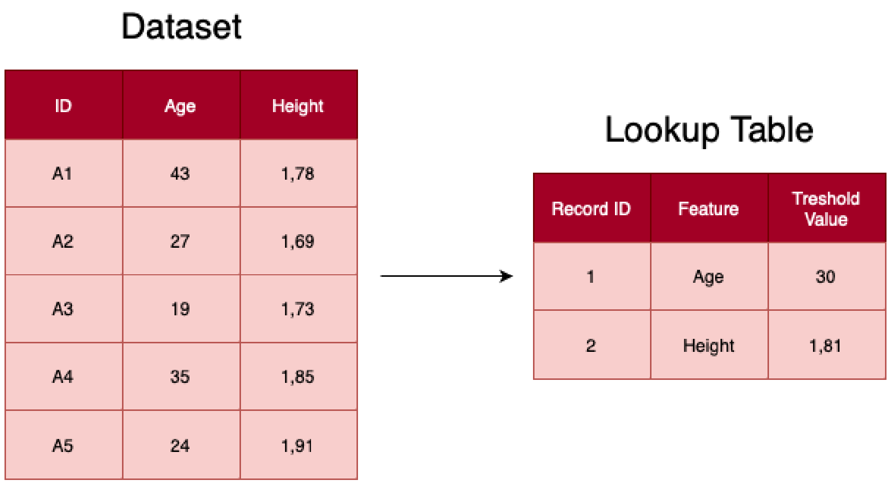
Algorithm 1 SecureBoost's Aggregate Encrypted Gradient Statistics

```
Input: $I$, instance space of current node
Input: $d$, feature dimension
Input: $\{Enc(g_i), Enc(h_i)\}_{i \in I}$
for $k = 0$ to $d$ do
    Propose $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ by percentile of feature $k$
end for
for $k = 0$ to $d$ do
    $G_{kv} = \sum_{i \in \{i | s_{k,v} \leq x_{i,k} > s_{k,v-1}\}} Enc(g_i)$
    $H_{kv} = \sum_{i \in \{i | s_{k,v} \leq x_{i,k} > s_{k,v-1}\}} Enc(h_i)$
end for
Output: $G \in \mathbb{R}^{d \times l}$, $H \in \mathbb{R}^{d \times l}$
```

Algorithm 2 SecureBoost's Split Finding Algorithm

```
Input: $I$, instance space of current node
Input: $\{G^i, H^i\}_{i=1}^m$, aggregated encrypted gradient statistics from $m$ parties
$g \leftarrow \sum_{i \in I} g_i$
$h \leftarrow \sum_{i \in I} h_i$
for $i = 0$ to $m$ do
    for $k = 0$ to $d_i$ do
        $g_l \leftarrow 0, h_l \leftarrow 0$
        for $v = 0$ to $l_k$ do
            get decrypted values $Dec(G^i_{kv})$ and $Dec(H^i_{kv})$
            $g_l \leftarrow g_l + Dec(G^i_{kv}), h_l \leftarrow h_l + Dec(H^i_{kv})$
            $g_r \leftarrow g - g_l, h_r \leftarrow h - h_l$
            $score \leftarrow max(score, \frac{g_l^2}{h_l+\lambda} + \frac{g_r^2}{h_r+\lambda} + \frac{g^2}{h+\lambda})$
        end for
    end for
end for
Output:
Partition current instance space according to the selected attribute's value
```

The split finding algorithm remains largely the same as XGBoost. However, due to separation of features between the parties, the passive party should keep a lookup table. The table should contain split thresholds for each feature: [feature k, threshold value v] and a unique id r. The unique id is used for indexing the table, to look up split conditions during inference. In Figure 4.3, you can see such a lookup table. The dataset has two features: "Age" and "Height". After computing the threshold values for both features, the data owner will save a local lookup table.



Dataset

ID	Age	Height
A1	43	1,78
A2	27	1,69
A3	19	1,73
A4	35	1,85
A5	24	1,91

Lookup Table

Record ID	Feature	Treshold Value
1	Age	30
2	Height	1,81

FIGURE 4.1: Lookup Table

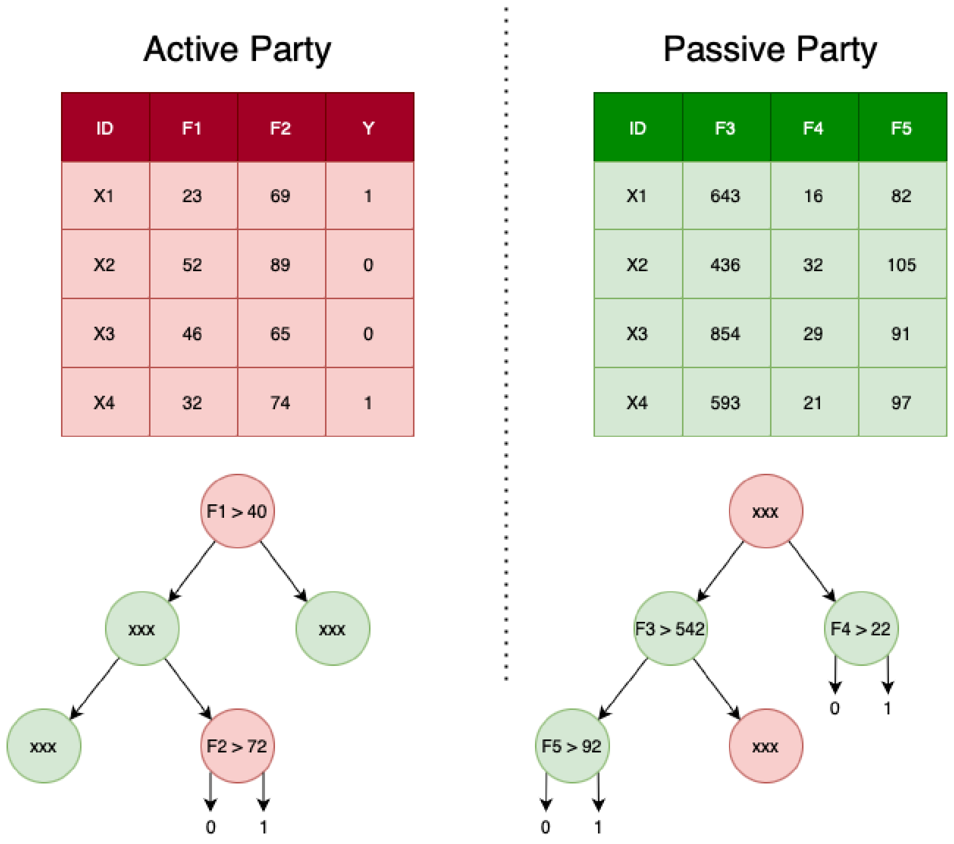
Meanwhile, the active party has their features locally. For the active party to know which passive party to deliver an instance to, as well as instruct the passive party which split condition to use at inference time, it associates every tree node with a pair (party id i, record id r). The optimal weight of leaf j, only depends on $\sum_{i \in Ij} g_i$ and $\sum_{i \in Ij} h_i$. The idea for finding the optimal weight follows a similar process as split finding. Once a leaf node is reached, the passive party sends $Enc(\sum_{i \in I_j} g_i)$ and $Enc(\sum_{i \in I_j} h_i)$ to the active party. The active party can then decipher the values and compute the corresponding weight through equation 2.4.

Federated Inference

The prediction process of SecureBoost is orchestrated by the active party. By referring to the record of the root, it will say [party id: x, record id i]. Party x will retrieve the corresponding attribute from its lookup table. It will find the threshold value, and decide on whether to go to the left or right child. This process will continue until a leaf is found.

In Figure 4.2 you can find a simplified illustration of vertical federated XGBoost. In this example, there are two parties with each their dataset. Together they have collaboratively built a decision tree. The active and passive parties can only see the threshold values for the nodes with their features.

Let's have a look at what happens when a new entry comes in with the following values: {ID: X5, F1: 42, F2: 70, F3: 531, F4: 30, F5: 80}. It will first look at the root note. F1 is a feature from the active party. The active party sees that the value for F1 from the incoming data is 42, which is more than 40, so we go to the left. Note: in decision trees, it is the convention to go to the left child whenever the test is true. The next node is a feature from the passive party. The active party will ask the passive party what to do. The passive party will use his lookup table and see that for F3, 533 is less than 542, so we go to the right child node. F2 is again a feature from the active party, so they see that 70 is less than 72, so we go to the right leaf, which is 1. That is also the prediction for this new data.



ID	F1	F2	Y
X1	23	69	1
X2	52	89	0
X3	46	65	0
X4	32	74	1

ID	F3	F4	F5
X1	643	16	82
X2	436	32	105
X3	854	29	91
X4	593	21	97

FIGURE 4.2: Vertical Federated XGBoost

4.2 Private Set Intersection

For data alignment between the active and passive party, we use the private set intersection implementation of the paper "Efficient Batched Oblivious PRF with Applications to Private Set Intersection" (Kolesnikov et al., 2016).

For our private data alignment protocol, we only need the column that allows us to find a match between two datasets, such as an "ID" column. This column can be extracted from both datasets, and the values in those two sets will be used as input for our PSI protocol.

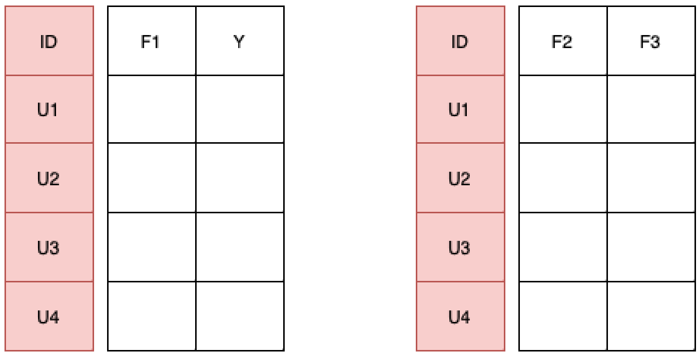


FIGURE 4.3: Extract the ID column for PSI

In the underlying algorithm of the PSI algorithm, the two parties will first hash their elements using the cuckoo hashing methods as described in chapter 2.2.1. The hashing function is an agreed-upon pseudo-random function F. With oblivious transfer, it is possible to exchange these two hash tables. The two parties look up each element of their set in the cuckoo hash table. If the element is found in the cuckoo hash table, then the element is in both sets. Otherwise, the element is not in both sets.

After the intersection is found, the two parties will exactly know which elements are in the intersection. They can then drop every entity from their dataset that is not in the dataset.

4.3 My Approach

The fundamental idea behind all tree models is finding the optimal split point for each feature. The process of finding the optimal split relies on calculating the split "score" for every possible split. For classification tasks, this could be done by calculating the Gini Impurity or Entropy. For regression tasks, the method of Mean Squared Error could be used. One of the steps in the XGBoost [Chen and Guestrin, 2016 is its split finding algorithm which enumerates all the possible splits on all the features. You can imagine that for large datasets with a lot of features, this can take a very long time. Inside this split-finding algorithm is a step where each feature gets sorted. The idea is to extract this step by providing the split algorithm pre-sorted data. By doing this, the overall time of building a decision tree will be reduced.

For XGBoost and the other three-based algorithms, data needs to be sorted. The idea is to pre-sort the data during the pre-processing phase. In this case, that means during data alignment. When the parties are performing private set intersection, the features should already be sorted per feature such that the time needed for building SecureBoost trees can be decreased.

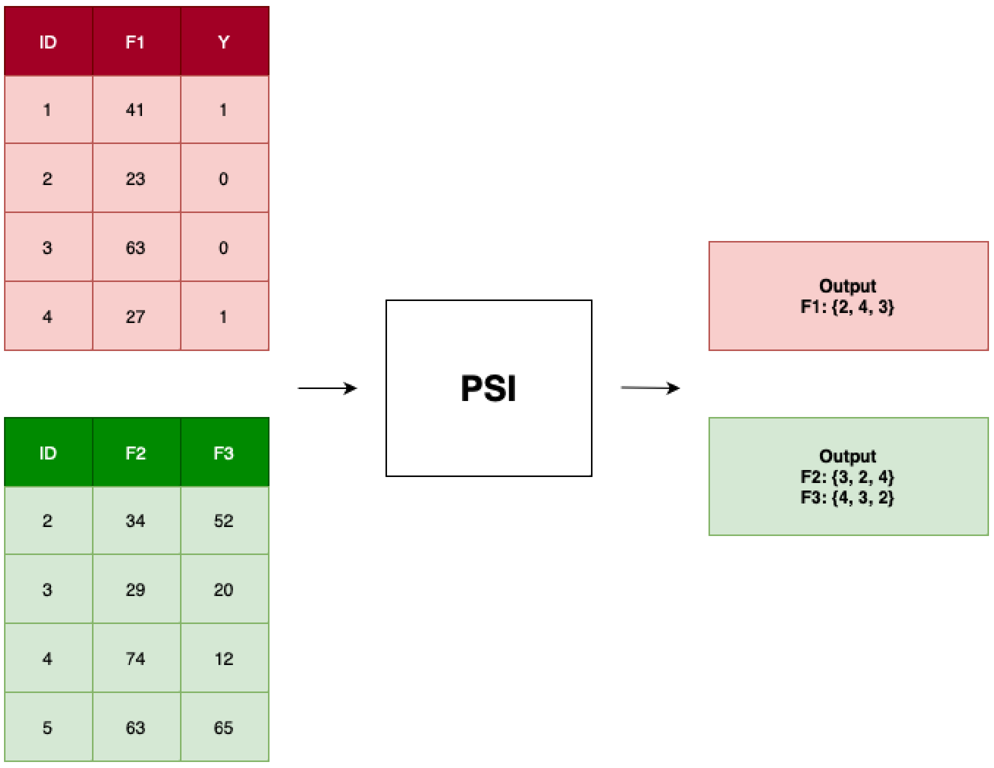


FIGURE 4.4: PSI Input and Output

A possible data structure for saving this sorted list is to use a skip list. A skip list is a probabilistic data structure that allows for fast search, insert, and delete operations. It is based on a linked list, but it also maintains a series of "skip pointers" that allow it to skip over large sections of the list when searching. Skip lists are well-suited for use in XGBoost and other tree-based algorithms because they allow for fast splitting of the data. To split the data on a given feature, the algorithm simply needs to start at the beginning of the list and follow the skip pointers until it finds a value that is greater than or equal to the split point. The algorithm can then split the list at that point.

In an ideal world each organisation that participates in collaboratively building a tree model, should have their data sorted at all times. Unfortunately, this is not something we can expect from all organisations.

This means for our PSI protocol that extra computation is necessary after the intersection itself is found. Each party has to put their whole dataset into the PSI protocol. After finding the intersection, for each feature, we will need to sort the data and output the ID's in ascended order according to their value.

For our VFL protocol, we won't have to change much. We will only need to alter the split-finding algorithm. Since the data is already sorted, we can skip that step. Everything else can stay the same.

4.3.1 System Architecture

In this chapter, we will have a closer look at the individual components that make up the whole system.

Data Flow

We first have a look at the flow of the data in Figure 4.5. The input is the dataset from the active and passive parties. The data will first go through the private set intersection process and the output will be the intersection and the sorted list per feature. The output of the PSI process will then be used as input for the vertical federated learning process. The final output will be the XGBoost trees, collaboratively built by the two parties.

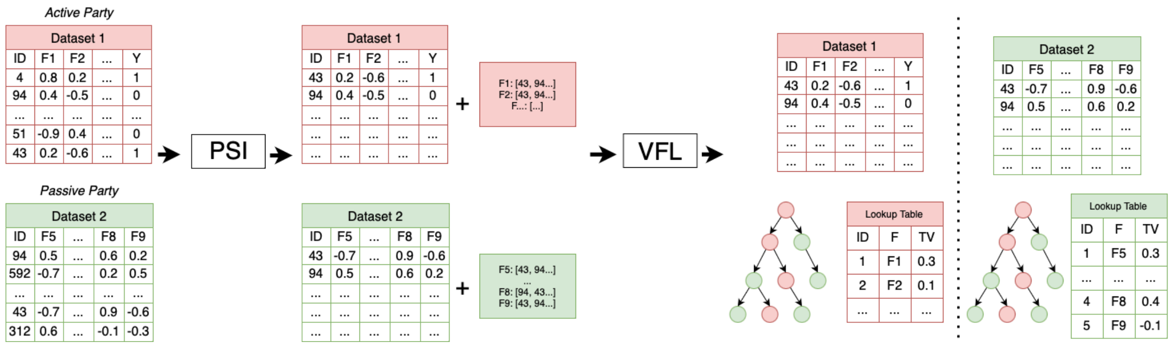


FIGURE 4.5: Dataflow

Private Set Intersection

The PSI architecture can be found in Figure 4.6. For private set intersection, we need both parties to extract their ID column. Both parties will apply cuckoo hashing locally on their set. With the help of oblivious transfer, the datasets will be exchanged between the two parties. Then the two parties can both look up which elements are in the intersection. The intersection that both parties get should have an equal size. Now that both parties know the intersection, per feature the datasets should be sorted. This sorted list should use the skip list data structure.

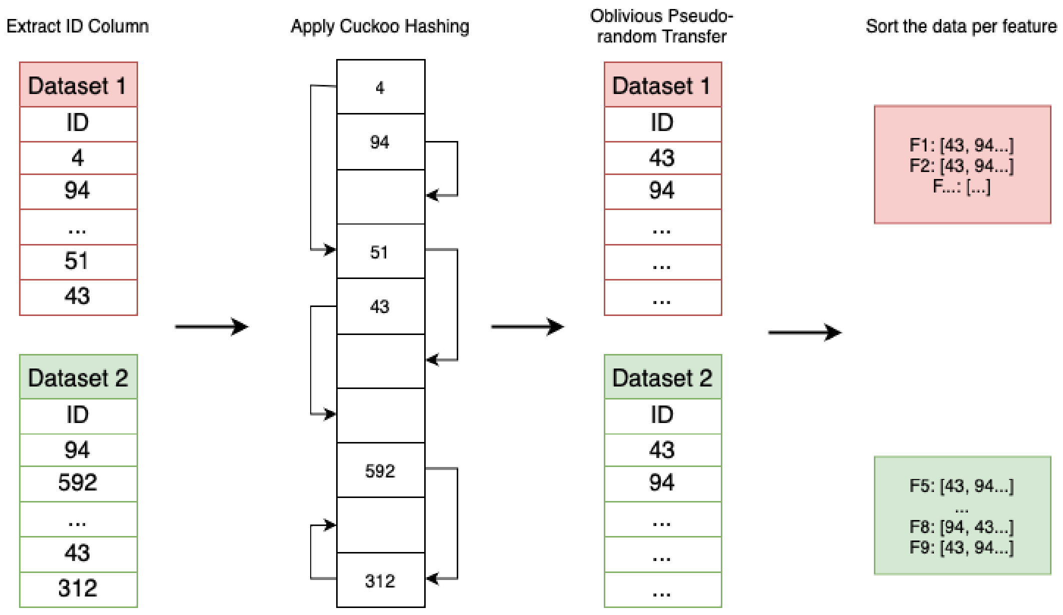
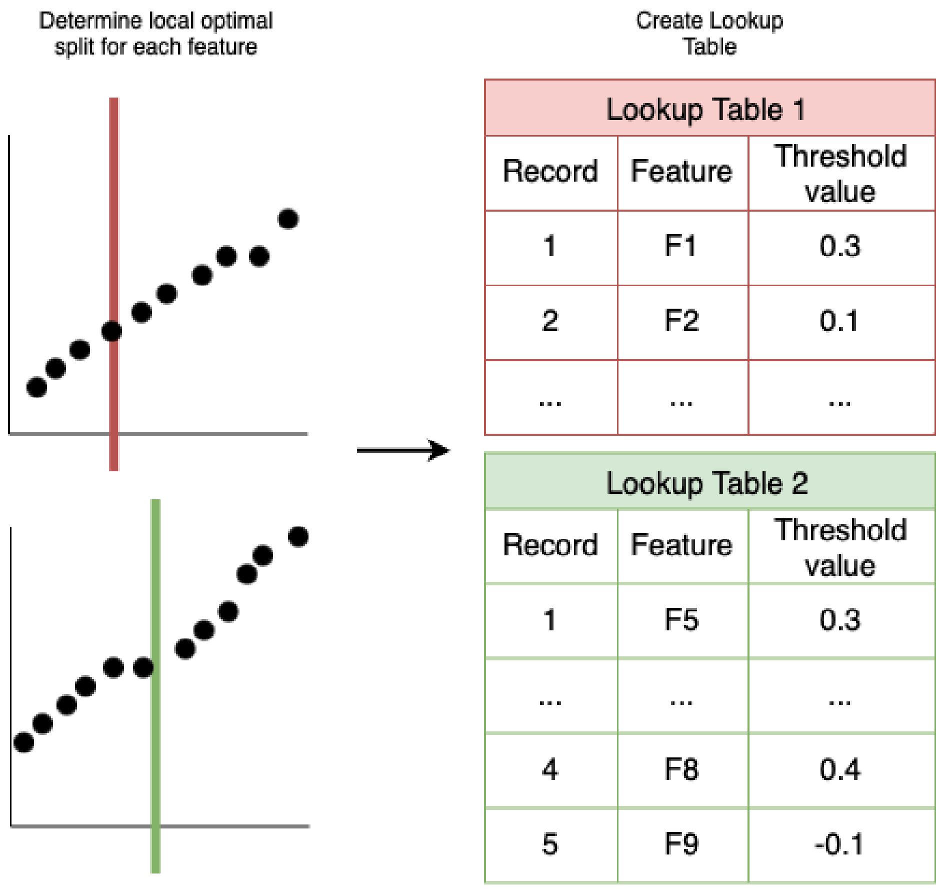


FIGURE 4.6: PSI Architecture

VFL

For the VFL flow in Figure 4.7, we use the output of PSI. Both parties need to create a lookup table with threshold values by finding the optimal split point for each feature. The active party can do this easily since it has access to the target label. The passive party will receive the encrypted gradient and hessian from the active party. The passive party will now send every possible split to the active party. The active party can then find the optimal split point by finding the split that gives the highest split score.



Lookup Table 1		
Record	Feature	Threshold value
1	F1	0.3
2	F2	0.1
...	...	...

Lookup Table 2		
Record	Feature	Threshold value
1	F5	0.3
...	...	...
4	F8	0.4
5	F9	-0.1

FIGURE 4.7: VFL Architecture

Complete System

Figure 4.8 depicts the complete high-level architecture of the system. At the top of the system, you can see the complete flow of the data and how it is manipulated. Zoomed in you can see the underlying algorithm for Private Set Intersection and Vertical Federated Learning.

The system works as follows:

1. We start with two datasets belonging to two different data holders. Only one party holds the target label, called the *Active Party*. The other party is called the *Passive Party*. The two datasets will be the input for PSI.

2. In the PSI protocol, the data will first be hashed using cuckoo hashing. Via pseudo-random oblivious transfer, the two parties can compute the intersection and both parties can sort the data per feature.

3. The intersection and sorted data will be used as the input for the VFL protocol. By working together, where the active party provides the encrypted gradients and hessians, the passive party can construct a lookup table to compute the best possible split for each feature. These split points will then be used for each node in the tree.

4. The output of this system, will be a vertical federated xgboost algorithm that can predict new data that is available at both the active and passive party.

In our system, we use a private set intersection protocol to combine the two datasets without revealing any sensitive information. This allows us to train a model on the combined dataset without compromising the privacy of the individual datasets. We use a vertical federated learning algorithm to train a tree model on the combined dataset. VFL allows us to train the model without sharing the data between the different parties involved.

By combining the current state-of-the-art for private set intersection and vertical federated learning for tree models, we have created a complete system that can train a model and predict new data. In this thesis, we implemented an optimisation for vertical federated tree models, by providing it pre-sorted data. This allows the algorithm to converge more quickly and efficiently, without compromising the accuracy of the system.

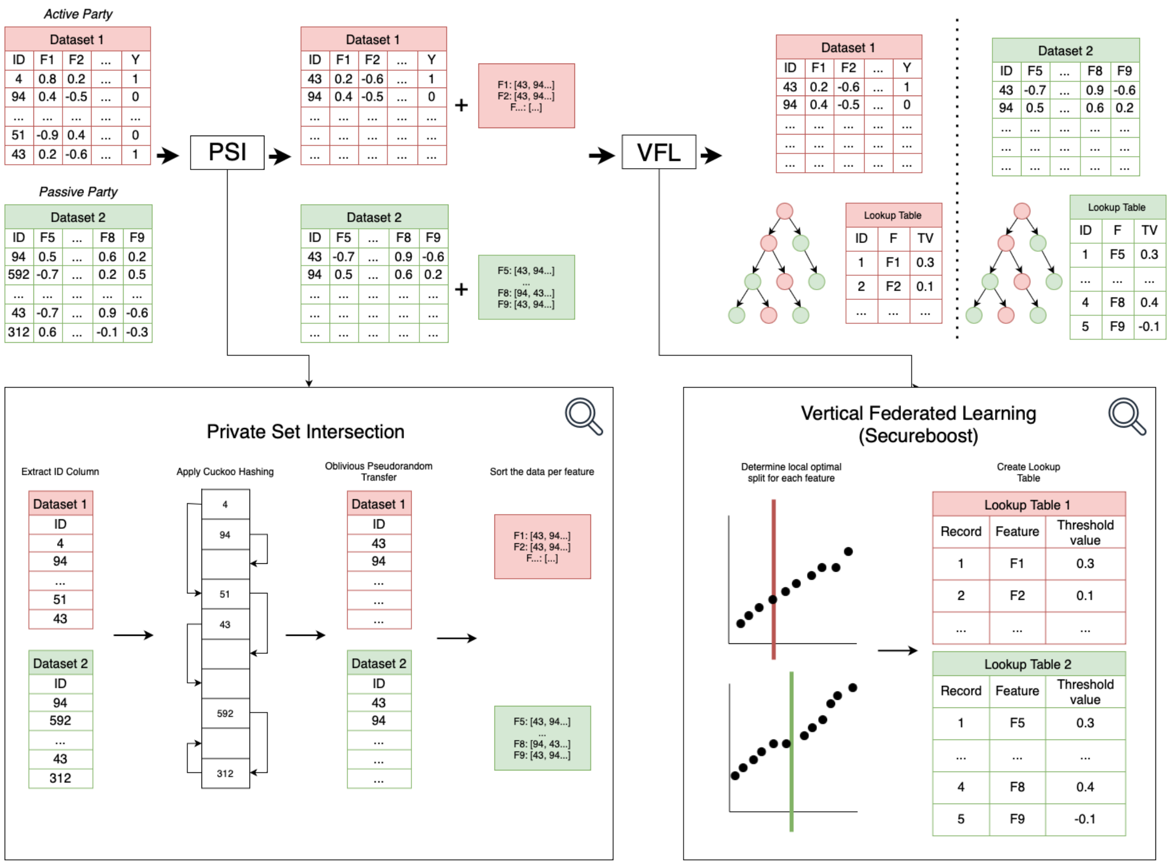


FIGURE 4.8: System Architecture

4.4 Privacy

One of the goals of this thesis is to not reduce the level of privacy while introducing the pre-sorting of data step into the system. Since there is no universal agreed-upon metric for privacy, I will create my framework based on my interpretation of privacy. I will test the system before pre-sorting is applied against this framework to figure out to which level the system belongs. We will then have a look at whether pre-sorting the data has any effect on the level of privacy.

4.4.1 Levels of privacy

Below you can find my definitions for the levels of privacy

- **Level 0** - At this level, there are no privacy mechanisms. All communications between different parties happen over plain data. The data that is being sent between the parties is not encrypted and each party receives the data in plain text.
- **Level 1** - At this level, the data that is being exchanged between parties is kept separate, but not encrypted. Local gradients can then still be calculated through global gradients from the data. This allows for, for example, target values to be calculated from backpropagations.

- **Level 2** - At this level, the data that is being exchanged is encrypted, but in such a way that computations are still possible. An example would be homomorphic encryption. When data is homomorphically encrypted, raw values are not available. Mathematical operations are, however, still possible on the data. Computations are therefore still available and somewhat still at risk if applied on data
- **Level 3** - At this level, all data that is being exchanged is encrypted and no one can decrypt it except for the sender. This will be the last level, but in terms of usability for communication between different parties, this is not really helpful.

4.4.2 Privacy analysis before pre-sorting

First, we will have a look at the "baseline" ER and VFL systems. The ER system is built upon a private set intersection protocol. The used PSI protocol has two steps. In the first step, the parties use cuckoo hashing to hash their elements. All parties use the same agreed-upon hash function F. In the second step, the two hash tables will be exchanged through pseudo-random oblivious transfer to find the intersection.

I would argue that this process belongs to Level 2. The way this protocol is designed makes it possible for only specific elements can be retrieved from the exchanged hash tables, namely the elements that are in intersection. By cleverly, extending the oblivious transfer protocol to include pseudo-random functions. the system works well in hiding the elements that are not in the intersection. As a side note on why only simply hashing the elements is not safe, imagine we use phone numbers to find the intersection. Phone numbers have a fixed length, which means that the space to look for is limited. A malicious user could brute force this by simply putting all available phone numbers as their input. The user will then receive all phone numbers that the other party has.

The VFL system is based on the SecureBoost algorithm. In this algorithm, the target value for each feature that is being exchanged between the parties is homomorphically encrypted. For all passive parties to find the optimal split point of their features, they need the target value from the active party. Since the target value is considered as sensitive data, it will be homomorphically encrypted. The reason why this algorithm is chosen for homomorphic encryption is that it allows the passive party to do the computations necessary to find the optimal split. During the inference step, when the complete trees are built, one part of the communication that is being exchanged without any encryption is whether the value for a specific feature is higher or lower than the threshold value. This could potentially be a vulnerable point of the system.

I would still consider the VFL part to be in Level 2. During the building phase of the algorithm, all data that is exchanged is homomorphically encrypted.

Specifically, it uses the Paillier cryptosystem. The Paillier system is unique in the sense that it uses non-deterministic encryption. That means that the same plain text could be encrypted into two different cypher texts. This adds another layer of security.

All computations happen locally. As long as there is no vulnerability found in the encryption, the data will stay safe. As for during the inference step figuring out the threshold value for each feature from the passive party might be possible, but I wouldn't think this is a huge deal, since it only is an estimate from the subset of the passive party. It is much more important to keep the target values safe.

Overall, I would put the entire system in Level 2.

4.4.3 Privacy analysis after pre-sorting

The optimisation that is introduced in this thesis happens directly after finding the intersection, but before starting the VFL phase. Between those two phases, each party sorts their data per feature. This process happens locally and no exchange of data has to be communicated between the parties.

Since the introduced changes happen locally, I would argue that the system stays in Level 2. The changes don't expose any additional data to the other parties. So, any information that they could get before stays the same. This ensures that the privacy guarantees of the VFL framework remain the same, as no additional data is exposed beyond what is already available in the intersection data.

Instead of unsorted data, each party receives the sorted data, but this data is already available in each party's local data. The pre-sorted data is only used to eliminate the sorting step of the SecureBoost algorithm. So, in conclusion, sorting the data doesn't affect the privacy level of the system.

Chapter 5

Evaluation

5.1 Experimental Setup

All experiments are running locally on a MacBook Pro (2021) with the following specifications:

- M1 Pro
- 8-core CPU @ 3.22 GHz
- 14-core GPU @ 1.29 GHz
- 16 GB RAM
- MacOS 13.5.2
- Python 3.8.13

For each experiment, we will be using the same default model configuration as specified in the original SecureBoost framework:

- Task type: Binary classification
- Number of trees: 5
- Learning rate: 0.3
- Subsample feature rate: 0.8
- Number of bins: 32

The conducted experimented should give an insight into the training speed with and without pre-sorting. For our baseline experiments, we measure the training speed by simply putting the result of the found intersection as the input for vertical federated learning. We will compare the baseline results with the results of the experiments where we did implement pre-sorting.

Our analysis focuses on the training time. By comparing the training with and without pre-sorting, we will gain a deeper understanding of the impact of pre-sorting on the performance of vertical federated tree building. This understanding will inform future strategies for optimizing vertical federated learning algorithms and frameworks based on tree models.

For all our experiments, we consider party A to be the *active party* and party B to be the *passive party*.

5.2 Datasets

For our experiments, we will use a combination of real and synthetic datasets. Synthetic datasets offer several advantages for conducting experiments in machine learning research. Synthetic data generation allows for precise control over the properties and characteristics of the generated data. In our case, we can generate data and manipulate the number of entities and features. In our experiments, we want to look at how different configurations react to pre-sorting the data.

We will also make use of synthetic datasets generated by scikit-learn's *make gaussian quantiles*[1] function. With these functions, it is possible to generate datasets by taking a multi-dimensional standard normal distribution and defining classes separated by nested concentric multi-dimensional spheres. The inputs that will be used are: n_samples, to determine the number of samples; n_features, to determine the number of features and set n_classes to 2, so the task will always be binary classification. In Table 5.1, you can find the datasets where each party has the same number of features that will be used for our experiments.

Dataset	Size	Features	
		Party A	Party B
Dataset 1	1000	5	5
Dataset 2	1000	10	10
Dataset 3	1000	15	15
Dataset 4	1000	20	20
Dataset 5	2000	5	5
Dataset 6	2000	10	10
Dataset 7	2000	15	15
Dataset 8	2000	20	20
Dataset 9	5000	5	5
Dataset 10	5000	10	10
Dataset 11	5000	15	15
Dataset 12	5000	20	20
Dataset 13	50000	5	5
Dataset 14	50000	10	10
Dataset 15	50000	15	15
Dataset 16	50000	20	20

TABLE 5.1: Synthetic datasets with an equal number of features per party

[1]https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_gaussian_quantiles.html

We will also conduct experiments on the datasets in Table 5.2 which have unequal numbers of features per party. By studying unequal numbers of features, we will see how they differ from the experiments with equal numbers of features. We will also see if there are any differences when the active or passive party has the larger portion of features.

Dataset	Size	Features	
		Party A	Party B
Dataset 1	1000	20	5
Dataset 2	1000	5	20
Dataset 3	2000	20	5
Dataset 4	2000	5	20
Dataset 5	5000	20	5
Dataset 6	5000	5	20

TABLE 5.2: Synthetic datasets with unequal number of features per party

Alongside the synthetic datasets, will also make use of real datasets. Real datasets are a reflection of the real world. By using real datasets, we can verify whether the optimisations also work with real-life data. It will also allow us to verify any insights obtained from the synthetic datasets. To save time and energy, we will make use of the online available real datasets, instead of collecting the data ourselves. These datasets are also designed for a binary classification task, similar to the synthetic datasets.

We will make use of the following real datasets:

- Breast Cancer Detection[2]
- A9A[3]

The first dataset is from scikit-learn. It is a binary classification dataset with a dimension of 569x30. As the name of the dataset suggests, it tries to predict whether a person has breast cancer. A9A is a dataset from the LIBSVM data repository. Every feature is a numeric type and the target column is a boolean.

The datasets have the following properties:

Dataset	Size	Features
BREAST CANCER	569	30
A9A	48,842	123

These datasets are normally used in classical machine learning, so we need to split the datasets vertically to prepare the datasets for our experiments. With two parties, we split the feature space in half. The first party gets the first half of the features and the second party gets the other half of the features.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html
[3]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html

We end up with the following dataset structure:

Dataset	Size	Features	
		Party A	Party B
BREAST_CANCER	569	15	15
A9A	48,842	62	61

5.3 Results

In this section, we will present the results from the experiments for the synthetic and real datasets.

5.3.1 Synthetic datasets

The first experiments were conducted on synthetic datasets, which provides a controlled environment for evaluating the performance of our algorithm. These datasets were generated with varying numbers of entities and features to study the impact of these factors on the training speed.

We will first present the results of the experiment where the tables are grouped by the number of entities. By keeping the number of entities fixed, it will give us an idea on how the number of features affect the training speed. First we have the results for 1000 entities with equal number of features per party:

#Features A	#Features B	Baseline (sec)	Optimisation (sec)	Time Diff. (sec)
5	5	20,4193	18,0059	2,4134
10	10	35,1062	31,2279	3,8783
15	15	47,4432	42,1878	5,2554
20	20	59,5872	52,5231	7,0641

TABLE 5.3: 1000 entities

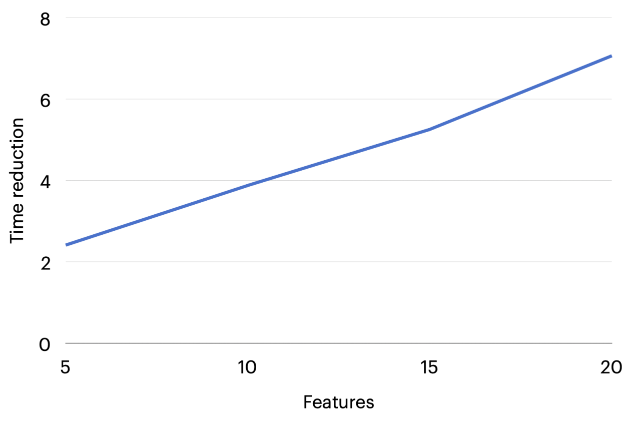


FIGURE 5.1: 1000 entities

We have the results for 2000 entities:

#Features A	#Features B	Baseline (sec)	Optimisation (sec)	Time Diff. (sec)
5	5	35,9133	33,1749	2,7384
10	10	54,5797	50,4216	4,1581
15	15	75,0203	68,5529	6,4674
20	20	95,4971	87,5973	7,8998

TABLE 5.4: 2000 entities

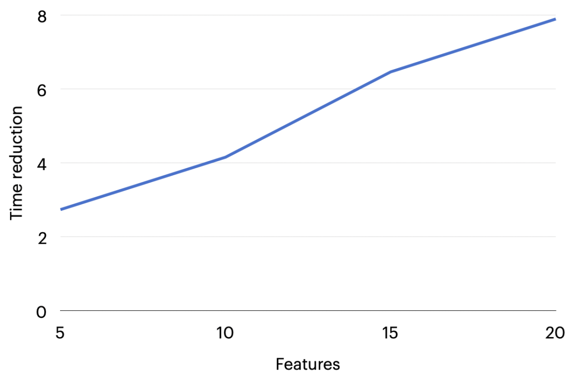


FIGURE 5.2: 2000 entities

We have the results for 5000 entities:

#Features A	#Features B	Baseline (sec)	Optimisation (sec)	Time Diff. (sec)
5	5	77,0485	73,9276	3,1209
10	10	116,0294	110,979	5,0504
15	15	149,9158	143,3239	6,5919
20	20	187,4538	179,4423	8,0115

TABLE 5.5: 5000 entities

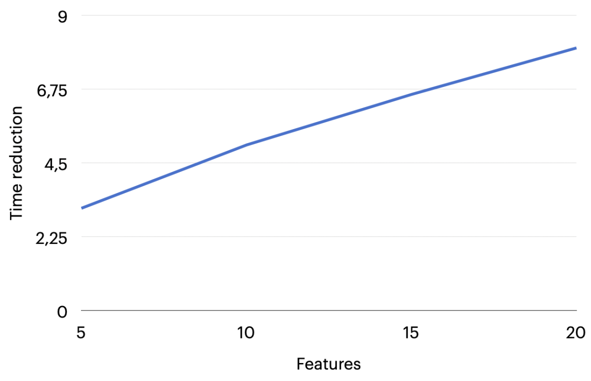


FIGURE 5.3: 5000 entities

We have the results for 50000 entities:

#Features A	#Features B	Baseline (sec)	Optimisation (sec)	Time Diff. (sec)
5	5	699,1740	687,8609	11,3131
10	10	985,7825	972,0544	13,7281
15	15	1270,3210	1254,2578	16,0632
20	20	1544,4057	1525,8692	18,5365

TABLE 5.6: 50000 entities

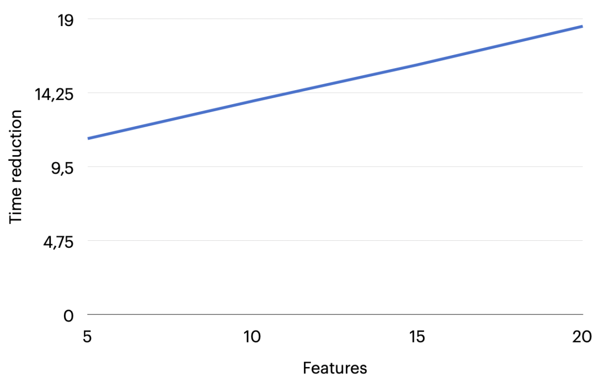


FIGURE 5.4: 50000 entities

A notable observation is that pre-sorting consistently improves the algorithm's speed across all experiments. We also observe that everytime we increase the number of features, the time reduction becomes larger. This makes sense, since in our algorithm there are now more features that don't have to be sorted anymore. Consequently, this means we save time in the overall training speed.

In the tables below, we present the results of the experiment where the tables are grouped by the number of features. By keeping the number of features fixed, we can see how the number of entities affect the training speed. We present the results for the following number of features: 5, 10, 15 and 20. They are the same number of features that we have used before.

We first start with the results for 5 features:

Entities per party	Baseline (sec)	Optimisation (sec)	Time diff. (sec)
1000	20,4193	18,0059	2,4134
2000	35,9133	33,1749	2,7384
5000	77,0485	73,9276	3,1209
50000	699,1740	687,8609	11,3131

TABLE 5.7: 5 features

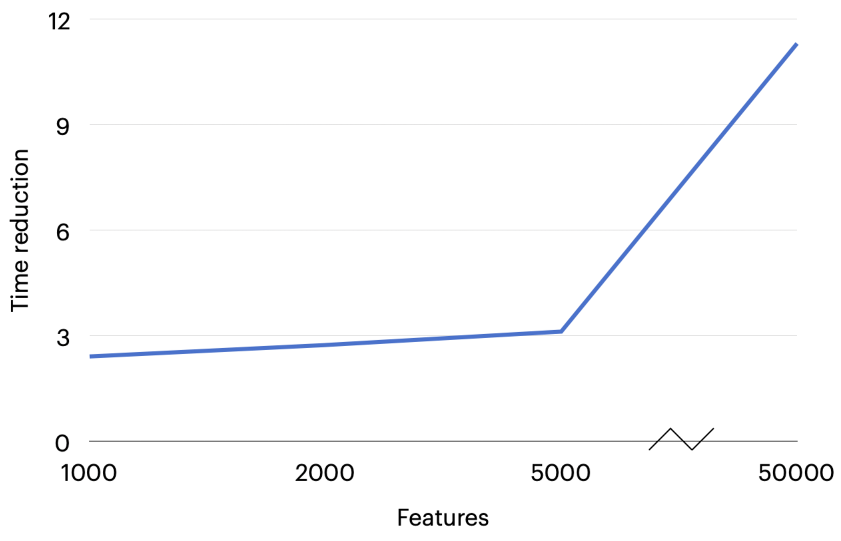


FIGURE 5.5: 5 features

We have the result for 10 features:

Entities per party	Baseline (sec)	Optimisation (sec)	Time diff. (sec)
1000	35,1062	31,2279	3,8783
2000	54,5797	50,4216	4,1581
5000	116,2294	110,6790	5,5504
50000	985,7825	972,0544	13,7281

TABLE 5.8: 10 features

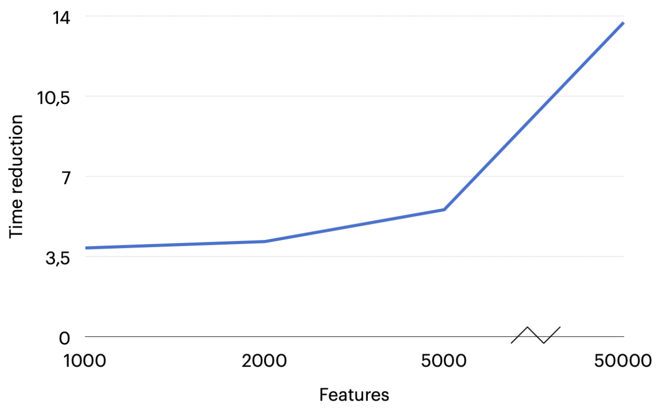


FIGURE 5.6: 10 features

We have the result for 15 features:

Entities per party	Baseline (sec)	Optimisation (sec)	Time diff. (sec)
1000	47,4432	42,1878	5,2554
2000	75,0203	68,5529	6,4674
5000	149,9158	143,3239	6,5919
50000	1270,3210	1254,2578	16,0632

TABLE 5.9: 15 features

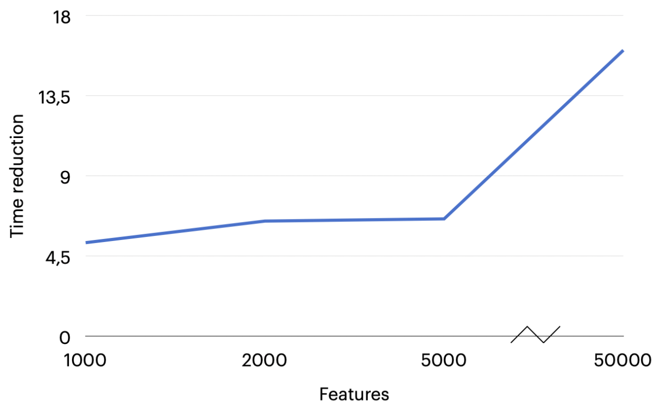


FIGURE 5.7: 15 features

We have the result for 20 features:

Entities per party	Baseline (sec)	Optimisation (sec)	Time diff. (sec)
1000	59,5872	52,5231	7,0641
2000	95,4971	87,5973	7,8998
5000	187,4538	178,4423	9,0115
50000	1544,4057	1525,8692	18,5365

TABLE 5.10: 20 features

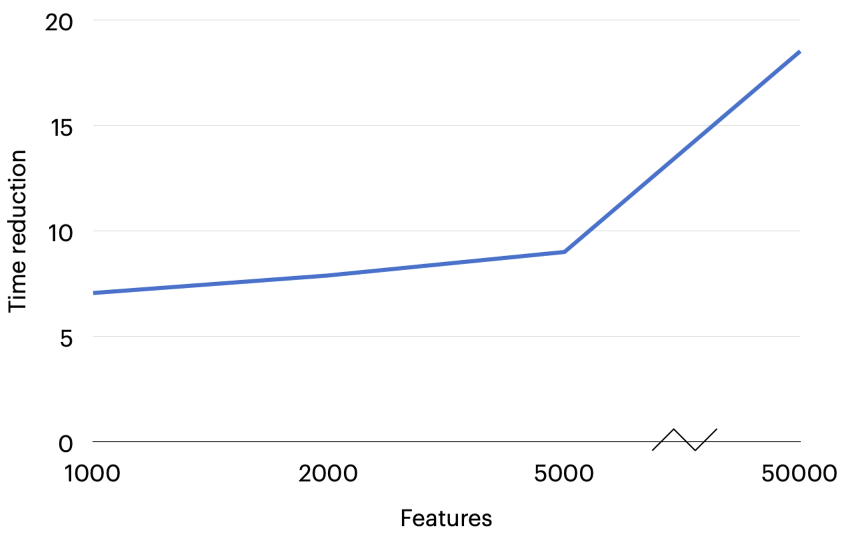


FIGURE 5.8: 20 features

Again, one of the first observations is that pre-sorting the data improves the algorithm's speed for all numbers of entities. We also observe that the time difference increases as the entities per party increases.

We will also have a look at the results from the experiments where the parties have an unequal number of features. The results can be found in Table 5.11.

Entities	#Features A	#Features B	Baseline	Optimisation	Time Diff. (Sec)
1000	5	20	59,7328	52,7942	6,9386
1000	20	5	21,6056	19,3017	2,3039
2000	5	20	95,7685	87,9697	7,7988
2000	20	5	36,3885	2,8624	8,1295
5000	5	20	190,6992	182,5697	8,1295
5000	20	5	78,3519	75,1992	3,1527

TABLE 5.11: Unequal number of features

A quick observation tells us that the optimisation results and the total training time heavily rely on the number of features for both the active and passive parties.

5.3.2 Real Datasets

In the first experiment, we look at the time it costs to train a model in seconds. We use the real datasets: Breast Cancer Detection, A9A and Gisette. We first run the baseline experiments. This is the standard protocol of SecureBoost without any optimisations. The results from this experiment, we will use as the baseline. Then we will feed the system the pre-sorted data. We will compare our approach to the baseline to see how the optimisations perform. The results can be found in Table 5.12.

Dataset	Experiment	Time (s)
Breast Cancer Detection	Baseline #1	29,5156
	Baseline #2	29,4328
	Baseline #3	28,7942
	Baseline #4	28,8045
	Baseline #5	28,5636
	Baseline Avg.	**29,0221**
	Optimisation #1	23,1921
	Optimisation #2	22,6360
	Optimisation #3	23,8732
	Optimisation #4	22,9643
	Optimisation #5	23,5523
	Optimisation Avg.	**23,2436**
A9A	Baseline #1	3186.4961
	Baseline #2	3227.0975
	Baseline #3	3205.3597
	Baseline #4	3199.5252
	Baseline #5	3197.4737
	Baseline Avg.	**3203,1904**
	Optimisation #1	3141.9522
	Optimisation #2	3160.6200
	Optimisation #3	3184.6824
	Optimisation #4	3163.5258
	Optimisation #5	3179.0982
	Optimisation Avg.	**3165,9757**

TABLE 5.12: VFL experiments with real data

Once again, as expected, we see that pre-sorting the data for real datasets also speeds up the training time. For both datasets, we ran the experiments a couple of times and took the average. We observe for that for the Breast Cancer detection we realise a speed-up of around 5.7786 seconds. For the A9A dataset, we observe a speed-up of around 37.2147 seconds.

5.4 Discussion

In this section, we will discuss the results found from the experiments for synthetic and real datasets.

5.4.1 Training speed

Pre-sorting data in every experiment for VFL reduces the training time compared to the baseline. This improvement is consistent across different numbers of features and entities, indicating that the optimization effectively eliminates the sorting process for each feature. The impact of pre-sorting becomes more noticeable as the number of features and/or entities increases, as there are more opportunities to skip the sorting step.

The consistent improvement in training time highlights the effectiveness of pre-sorting data in VFL. This optimization can enhance the performance of VFL algorithms based on tree models, particularly when dealing with a large number of entities and/or features. By eliminating the need for sorting during each experience, pre-sorting can significantly reduce the computational overhead and accelerate the overall training process.

Opposed to a percentage-based approach, it is more meaningful to evaluate this optimization's impact in terms of absolute time reductions. The optimisation gives a flat amount of time that reduces irrespective of the model configuration. If we analyse the algorithm, we always reduce the training time by the same flat amount of seconds for the same number of entities and features, regardless of the model configuration. If we increase, for example, the number of trees in our model configuration, the whole training process will take longer, but the flat time reduction stays the same. Since sorting data is an independent operation from the rest of the algorithm, it doesn't scale with model parameters. Evaluating its time reduction as a percentage of the overall training process would therefore be an inaccurate comparison.

We have also observed the results from the experiments when parties have an unequal number of features. One of the observations is that the total training time also relies on how the features are divided. More features and entities for the passive party means a longer training time since more cryptographic computations need to be done. Consequently, that means that our pre-sorting optimisation also relies on the division of features. We see similar results when we compare the experiments where we take the same number of features for the passive party with the experiments where we have an equal number of features for both parties.

In Table 5.13 below you can see that the optimisations with different divisions of features. x/y in the table means that the active party has x amount of features and the passive party has y amount of features. When comparing the result we see that the optimisation that we can expect from a certain dataset size relies on the number of features from the passive party, e.g. 5/5 has roughly the same speed-up as 20/5.

Entities	Div. feat. #1	Time diff. (sec)	Div. feat. #2	Time diff. (sec)
1000	5/5	2,4134	20/5	2,3039
1000	20/20	7,0641	5/20	6,9386
2000	5/5	2,7384	20/5	7,7988
2000	20/20	7,8998	5/29	2,8624
5000	5/5	3,1209	20/5	8,1295
5000	20/20	8,0115	5/20	3,1527

TABLE 5.13: Time difference of different divisions of features

5.4.2 Accuracy

The assumption is that pre-sorting does not affect the accuracy of the model. The same optimal split point would be found, regardless of whether we apply pre-sorting or not.

A simple experiment has been conducted with the *breast cancer detection* dataset. If we don't apply pre-sorting, we get an accuracy of **97.37%**. When we do apply pre-sorting, we also get the same **97.37%** accuracy. This means that our optimisation is lossless since we don't alter the accuracy of the model.

5.4.3 Analysis of realised speed-up

The speed-up that we have seen from the real datasets is not unexpected. If we look at the results from the synthetic datasets we can estimate how much time we reduce per feature and entity.

First, we have a look at the result from the perspective of fixed entities and how features impact the training time:

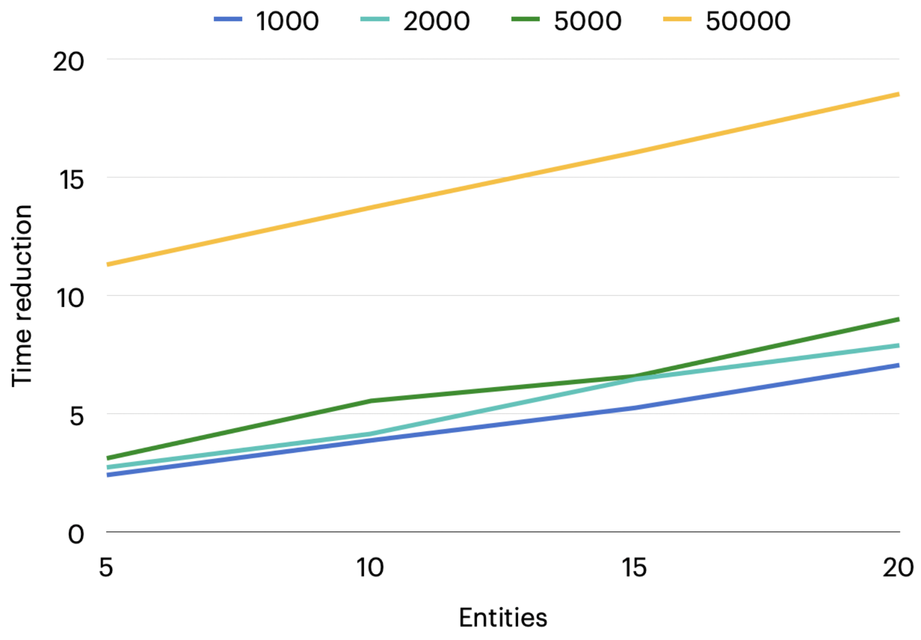


FIGURE 5.9

From the results, we can calculate the average slope of the results, which is around 0.3654 seconds per feature. So, we save around 0.3654 seconds per feature for each feature that the passive passive has.

We can do the same from the perspective of fixed features and observe how entities impact the training time:

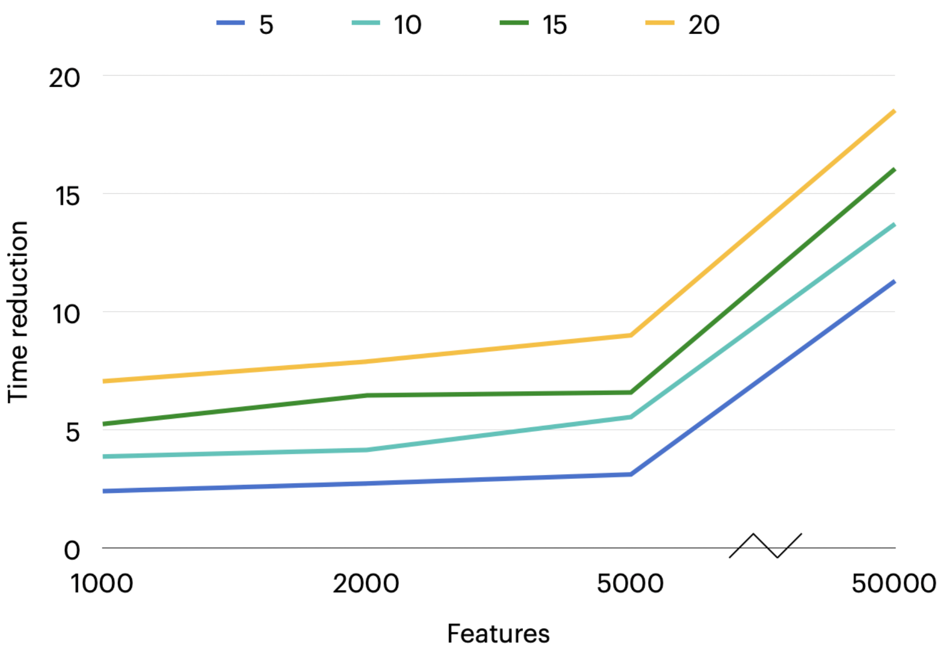


FIGURE 5.10

We can again calculate the average slope of the results and this time it is around 0.000209334 per entity. So around 0.2093 seconds per 1000 entities.

If we look at the results from earlier from the *breast cancer detection* dataset, we see pre-sorting the data results in, on average, 5,7786 seconds. The *breast cancer detection* dataset per party has a size of 569 entities and 15 features. So the expected speed-up is $(15 * 0.3654) + (0,569 * 0.2093) = 5.6006$, which is close to the actual speed-up.

We can do the same for the *A9A* dataset. The passive party has the partitioned dataset with 49,842 entities and 61 features. The actual speed-up was 37,21472 seconds. The expected speed-up is $(61 * 0.36543425) + (48.842 * 0.2093) = 32.5141$. For this experiment, the actual and expected speed-up are a bit more off. This could be explained by a rather small sample size and a high variance. From the five experiments that we have conducted, there are around 40 seconds between the fastest and slowest training time. For a better estimate, two things need to be done: more experiments with synthetic to get a better estimate and more experiments with the real dataset to get to the true average.

Chapter 6

Conlusion and Future Work

In this chapter, we conclude the research for this thesis by answering the research questions and we explore follow-up ideas for future work.

6.1 Conclusion

The ultimate goal in the bigger picture is to make the process of vertical federated learning based on tree models computationally faster. However, that scope is way too broad for a master's thesis. Instead, we will research a smaller part of the algorithm and gain insight into how this part affects the whole process. The part that was researched was the sorting process of the data per feature when training a vertical federated tree model and how this can be optimised.

The optimisation idea was to pre-sort the data per feature to eliminate the sorting process. But while designing this optimisation, we have to take privacy into account. Federated learning evolved from machine learning with a heavy emphasis on privacy. So, while we do want to speed up the total process, we should not violate any privacy properties.

A prototype system that can perform private set intersection, pre-sorting, and vertical federated learning has been created for this thesis. With this system, experiments can be conducted to answer the research questions.

6.1.1 How does pre-sorting the data per feature affect the computation time of building vertical federated learning trees?

After researching the vertical federated version of XGBoost (SecureBoost), we analysed the algorithm behind this protocol. One of the steps that we found was that it sorts the data per feature. This process is necessary for finding the optimal split point for a feature. The idea was to remove this sorting process by providing the tree-building protocol pre-sorted data.

Sorting the data can happen any time before the start of the tree-building process, as well as during the process of private set intersection. In our system, we use PSI to find the intersection between the dataset of two parties privately. This is necessary, because otherwise the data wouldn't be correctly aligned, i.e. the entity of the first row of the active party is a different entity from the first row of the passive party.

In our experiments, we run the system with and without pre-sorting. The baseline is the system without pre-sorting, and we compare results with the system where we do apply pre-sorting. We have conducted experiments on various numbers of entities and features.

We have used real datasets and synthetic datasets. A synthetic datasets generator allowed us to generate datasets with specific numbers of entities and features. We could then use these datasets for our experiments and observe in the results how different numbers of entities and features affect the training speed in combination with the pre-sorting optimisation.

The real datasets we have used are normally used in classical machine learning settings. We had to vertically partition the datasets ourselves, such that both parties had common entities but a different subset of the features.

With our experiments, we have gained the following insights:

- The first trivial insight is that pre-sorting lowers the training time of the system. This makes sense since we removed the sorting step with hardly any drawbacks. The improvement can be observed across all different numbers of features and entities, for both synthetic and real datasets.
- The realised speed-up has to be viewed as a fixed value, rather than a percentage of the training process. This is because the total training time also depends on the model parameters. By increasing, for example, the number-of-trees parameter, the training time will increase. But the speed-up from pre-sorting will stay the same, which means it is not a fair comparison if we look at it from a percentage perspective.
- The total training time and speed-up of the pre-sorting optimisation depends on the number of features at the passive party. The passive party is the one that does the heavy computations with homomorphic encryption and the results of the experiments with unequal feature sizes reflect this.
- The system's accuracy during inference is not affected by the pre-sorting optimisation. Regardless of whether the system receives unsorted or sorted data, the system will still find the same optimal split point for each feature.
- It is possible to estimate the actual speed-up. We have seen that from the results with synthetic datasets, we can save around 0.3654 seconds per feature and around 0.2093 seconds per 1000 entities. These numbers are calculated from the limited experiments that we have done so far, but for the true estimates, a lot more experiments with various sizes need to be run.

6.1.2 Do we maintain the level of privacy during PSI and VFL while introducing modifications to the system?

In Chapter 4.4, we have made our framework to determine where our system fits in. Privacy is a difficult matter to quantify and everyone has their judgement and interpretation. Therefore, we have created our own framework.

In this framework, there are four levels. In the first level, level 0, there are no privacy constraints, all data that is being communicated between different parties are the raw values for everyone to see. In level 1, data is being kept separate, but global gradients are still available for local gradient calculations. In level 2, the data that is being exchanged between parties is encrypted, but in such a way that computations are still possible. Homomorphic encryption is an example of a protocol that belongs in level 2. The last level, level 3, consists of protocols where all data, that is being exchanged, is encrypted and no one, except for the sender of the data, can decrypt it.

The baseline system consists of private set intersection, through cuckoo hashing and oblivious pseudo-random transfer, and the SecureBoost protocol, a vertical federated version of XGBoost. In our judgment, we have placed this system on level 2.

In our analysis, we have looked at the system in two different parts: the entity resolution and the machine learning. Both of these systems were put into level 2.

For entity resolution through private set intersection, we see that the elements are hashed. But with oblivious pseudo-random transfer, a party can find the elements from the other parties only if they own the same element in their set as well. Through this protocol, the intersection can be privately computed, where each party can only see the elements that are in the intersection, and nothing else.

The machine learning part uses SecureBoost, a protocol for vertical federated XGBoost. The data that is being exchanged between the parties are the target values for each entity. This data is homomorphically encrypted such that the optimal split point can be found without revealing the actual target value.

The optimisation was to apply the pre-sorting of data for each feature. The pre-sorting happens locally and no additional data has to be exchanged between the parties. We, therefore, judged that pre-sorting the data does not affect the privacy constraints from the baseline system. This means that we do maintain the level of privacy, while we pre-sort the data, under the assumptions of our privacy framework.

6.2 Future Work

Federated learning is a relatively new specialised form of machine learning. Vertical federated learning is a setting inside the federated learning protocol.

In this thesis, we have looked very specifically at the sorting process of a vertical federated learning protocol based on tree models.

This means that there are still a lot of opportunities for research and improvements since it is such a young research topic.

Let's first have a look at vertical federated learning based on tree models.

In this thesis, we have researched the sorting process of the training part. We have optimised this in the training process by removing sorting altogether by pre-sorting the data during the entity resolution process.

The training and inference sections, however, consist of way more parts, such as tree-building, encryption, and communication. Each of these parts has its challenges, and future research can investigate these challenges and identify areas for improvement.

In my experiments, I initially had a third real dataset that I wanted to use. The Gisette dataset. This dataset has 13,500 entities and 5000 features. Unfortunately, my laptop couldn't handle such a large dataset. Some future work could include larger datasets and verify whether the expected speed-ups still hold.

Additionally, in this thesis, we solely focused on numerical values in our datasets. It would also be a good idea to see what pre-sorting would look like for different types, such as booleans and strings.

If we look at vertical federated learning in general, there is a process that is specific to the vertical setting of federated learning, as opposed to horizontal federated learning. That is entity resolution.

Entity resolution is necessary for vertical federated learning since we need to make sure that every participating party owns the correct subset of entities before we can start vertical federated learning. But most of the available research doesn't pay a lot of attention to this part. Most of the papers mention that they use private set intersection. A downside of this protocol, however, is that it needs a unique identifier that both parties use. In the real world, that might be hard to find. If we are talking about toys, two different stores could internally refer to the same toy. However, they use different identifiers, which means that private set intersection does not find this element in the intersection while both identifiers refer to the same toy.

Private set intersection is an exact algorithm that always finds the exact intersection, but it comes with the cost of high computation time. A possible research opportunity could be to look at different exact and approximate private entity resolution methods to find the intersection.

In conclusion, a lot of research and improvements are possible in the field of vertical federated learning and federated learning in general. While this topic is very young, I expect a lot of new papers in the coming years that will try to improve this.

Bibliography

Acar, Abbas et al. (2018). "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". In: *ACM Comput. Surv.* 51.4. ISSN: 0360-0300. DOI: `10.1145/3214303`. URL: `https://doi.org/10.1145/3214303`.

Breiman, Leo (1996). "Bagging Predictors". In: *Machine Learning* 24.2, pp. 123–140.

Chen, Hao, Kim Laine, and Peter Rindal (Oct. 2017). "Fast Private Set Intersection from Homomorphic Encryption". In: pp. 1243–1255. ISBN: 978-1-4503-4946-8. DOI: `10.1145/3133956.3134061`.

Chen, Tianqi and Carlos Guestrin (Aug. 2016). "XGBoost: A Scalable Tree Boosting System". In: pp. 785–794. DOI: `10.1145/2939672.2939785`.

Cheng, Kewei et al. (Jan. 2019). "SecureBoost: A Lossless Federated Learning Framework". In.

Cheon, Jung et al. (Nov. 2017). "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: pp. 409–437. ISBN: 978-3-319-70693-1. DOI: `10.1007/978-3-319-70694-8_15`.

De Cristofaro, Emiliano and Gene Tsudik (Jan. 2010). "Practical Private Set Intersection Protocols with Linear Complexity". In: vol. 6052, pp. 143–159. ISBN: 978-3-642-14576-6. DOI: `10.1007/978-3-642-14577-3_13`.

Du, Wenliang, Yunghsiang Han, and Shigang Chen (Apr. 2004). "Privacy-Preserving Multivariate Statistical Analysis: Linear Regression And Classification". In: DOI: `10.1137/1.9781611972740.21`.

Dwork, Cynthia (Apr. 2008). "Differential Privacy: A Survey of Results". In: vol. 4978, pp. 1–19. ISBN: 978-3-540-79227-7. DOI: `10.1007/978-3-540-79228-4_1`.

EU (2018). In: *EPDS*. URL: `https://edps.europa.eu/press-publications/press-news/press-releases/2018/data-protection-and-privacy-2018-going-beyond-gdpr_en`.

Fangcheng Fu Huanran Xue, Yong Cheng Yangyu Tao Bin Cui (June 2022). "BlindFL: Vertical Federated Machine Learning without Peeking into Your Data". In.

Fu, Fangcheng et al. (June 2021). "VF2Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning". In: pp. 563–576. DOI: `10.1145/3448016.3457241`.

Grinsztajn, Léo, Edouard Oyallon, and Gael Varoquaux (July 2022). "Why do tree-based models still outperform deep learning on tabular data?" In: DOI: `10.48550/arXiv.2207.08815`.

Hopfield, John (May 1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". In: *Proceedings of the National Academy of Sciences of the United States of America* 79, pp. 2554–8. DOI: 10.1073/pnas.79.8.2554.

Huberman, Bernardo, Matt Franklin, and Tad Hogg (Sept. 1999). "Enhancing Privacy and Trust in Electronic Communities". In: *ACM International Conference Proceeding Series*. DOI: 10.1145/336992.337012.

Inbar, Roi, Eran Omri, and Benny Pinkas (Aug. 2018). "Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters: 11th International Conference, SCN 2018, Amalfi, Italy, September 5–7, 2018, Proceedings". In: pp. 235–252. ISBN: 978-3-319-98112-3. DOI: 10.1007/978-3-319-98113-0_13.

Kolesnikov, Vladimir et al. (Oct. 2016). "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: pp. 818–829. DOI: 10.1145/2976749.2978381.

Lindell, Yehuda and Benny Pinkas (Apr. 2009). "A Proof of Security of Yao's Protocol for Two-Party Computation". In: *Journal of Cryptology* 22, pp. 161–188. DOI: 10.1007/s00145-008-9036-8.

Liu, Yang, Tianjian Chen, and Qiang Yang (Dec. 2018). "Secure Federated Transfer Learning". In.

McMahan, Brendan and Daniel Ramage (2017). *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. URL: https://blog.research.google/2017/04/federated-learning-collaborative.html.

Narayanan, Arvind and Vitaly Shmatikov (Jan. 2006). "Shmatikov How To Break Anonymity of the Netflix Prize Dataset. arxiv cs/0610105". In.

Nikolaenko, Valeria et al. (May 2013). "Privacy-Preserving Ridge Regression on Hundreds of Millions of Records". In: pp. 334–348. ISBN: 978-1-4673-6166-8. DOI: 10.1109/SP.2013.30.

Patgiri, Ripon, Sabuzima Nayak, and Samir Borgohain (Jan. 2018). "Role of Bloom Filter in Big Data Research: A Survey". In: *International Journal of Advanced Computer Science and Applications* 9, pp. 655–661. DOI: 10.14569/IJACSA.2018.091193.

Pinkas, Benny, Thomas Schneider, and Michael Zohner (Jan. 2018). "Faster private set intersection based on OT extension". In: *ACM Transactions on Privacy and Security* 21, pp. 797–812. DOI: 10.1145/3154794.

Tian, Zhihua et al. (Nov. 2020). "FederBoost: Private Federated Learning for GBDT". In.

Will, Mark and Ryan Ko (Dec. 2015). "A guide to homomorphic encryption". In: pp. 101–127. ISBN: 9780128015957. DOI: 10.1016/B978-0-12-801595-7.00005-7.

Yang, Qiang et al. (Feb. 2019). "Federated Machine Learning: Concept and Applications". In.

Zeng, Tengchan et al. (2020). "Federated Learning in the Sky: Joint Power Allocation and Scheduling with UAV Swarms". In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148776.

Zhang, Chen et al. (2021). “A survey on federated learning”. In: *Knowledge-Based Systems* 216, p. 106775. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.106775. URL: https://www.sciencedirect.com/science/article/pii/S0950705121000381.

Zhao, Lingchen et al. (Apr. 2018). “InPrivate Digging: Enabling Tree-based Distributed Data Mining with Differential Privacy”. In: pp. 2087–2095. DOI: 10.1109/INFOCOM.2018.8486352.