POLITECNICO DI TORINO

Master degree in
Mathematical Engineering

Master degree thesis

Fully Homomorphic Encryption and applications to Machine Learning

Supervisor
prof. Danilo Bazzanella

Candidate
Edoardo Venturini

Co-supervisors
Veronica Cristiano and Marco Rinaudo

Academic year 2023-2024

Summary

Nowadays, cloud technologies are increasingly embedded in both our personal lives and businesses, transforming how we store, process, and access data. They offer numerous advantages: cloud storage solutions enable remote accessibility from any device, while powerful cloud computing resources allow companies to scale their operations and computational needs, without heavy investments in physical infrastructure. Additionally, cloud services encourage collaboration and innovation by enabling real-time data sharing and easy integration with Artificial Intelligence and Machine Learning tools.

However, as reliance on cloud platforms grows, so do the risks for users' privacy. Data stored in the cloud is often exposed to potential security breaches, unauthorized access and surveillance, as cloud providers hold extensive access to personal and sensitive information. Despite encryption and other protective measures, users remain vulnerable to data exploitation and privacy erosion. Balancing cloud benefits with robust privacy protections is crucial to safeguard user trust and maintain the technology's long-term potential.

In this scenario, this thesis wants to analyze the main features of Fully Homomorphic Encryption (FHE), a revolutionary cryptographic approach that enables computations on encrypted data without needing to decrypt it. The ultimate purpose is to explore its applications to Machine Learning (ML), focusing on the privacy challenges of client-server frameworks, where sensitive data must be shared for processing.
The work begins with a general introduction to cryptography, including both symmetric and asymmetric techniques. It then presents the history and evolution of FHE, describing the development of the most studied schemes through the last 15 years.
After that, the thesis discusses some of the most popular FHE libraries, examining their implementations and potential for real-world applications. Key applications of FHE highlighted in the work include medical research, where it allows for secure analysis of encrypted medical data; cloud services, enabling private data processing without exposing the contents to service providers; and search engines, which can perform encrypted searches on sensitive data without revealing to service providers either the queries or the results.

This encryption technique offers promising solutions to the privacy problems encountered in the context of Machine Learning, where sensitive information is often outsourced to external servers for model inference. More specifically, in a client-server architecture, the server trains a public ML model on some public data, and every client can send it a query, i.e. a new sample to be classified. When the server receives a query, it performs encrypted computations, and then sends back the encrypted response to the client, who can decrypt it to obtain the result.
In this framework, Fully Homomorphic Encryption ensures that the data remains encrypted throughout the entire process, safeguarding user privacy. However, encrypting data with FHE comes with a price: the computational effort can be high and this can

become a problem when considering complex tasks. This thesis explores the integration of FHE with ML algorithms, focusing on the tradeoff between safeguarding data confidentiality and keeping the model accurate and efficient. First of all, we present some solutions to optimize the application of FHE to Machine Learning, proposed with the purpose of reducing the overall computational overhead. Then, we analyze the general ML classification paradigm, going into details of three ML supervised algorithms: K-Nearest Neighbors, Support Vector Machines, Decision Tree (and Random Forest). We compare encrypted versus unencrypted models, describing some of the proposed solutions in literature. Our goal is to give a general idea, showing how TFHE and CCKS, two of the main FHE schemes, are integrated with the aforementioned ML methods. Our work demonstrates the potential of FHE to enable secure ML, so that the clients' data can be processed by servers without any information leakage.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Our work starts with the introduction of the mathematical tools we need: section 1.1 gives some algebraic definitions which are fundamental for the understanding of the thesis. After that, section 1.2 provides a general overview of cryptography, describing its building blocks.

1.1 Mathematical Preliminaries

Before presenting the bases of cryptography, we must introduce some mathematical formalism which constitute the background for the topics of next chapters. Let us start with the definition of groups and fields:

Definition 1.1.1 (Group) *A group $(G, +)$ consists of a set G with an operation $+ : G \times G \to G$, that associates an element of the set to every pair of elements of the set. The operation satisfies the following constrains:*

- *it is associative, i.e. $\forall f, g, h \in G \quad (f + g) + h = f + (g + h)$;*
- *it has an identity element, i.e. $\exists 0_G \in G : g + 0_G = 0_G + g = g \; \forall g \in G$;*
- *every element of the set has an inverse element, i.e. $\forall g \in G \; \exists! -g \in G : g + (-g) = -g + g = 0_G$.*

Definition 1.1.2 (Field) *A field $(F, +, \cdot)$ is a set F endowed with two binary operations $+$ and $\cdot$ called addition and multiplication, such that:*

- *$(F, +)$ is a group and $+$ is commutative, i.e. $\forall f, g \in F \quad f + g = g + f$;*
- *the multiplication is associative, i.e. $\forall f, g, h \in F \quad (f \cdot g) \cdot h = f \cdot (g \cdot h)$;*
- *the multiplication has an identity element, i.e. $\exists 1_F \in F : f \cdot 1_F = 1_F \cdot f = f \; \forall f \in F$;*
- *every element of the set has an inverse with respect to multiplication, i.e. $\forall f \in F \; \exists! f^{-1} \in F : f \cdot f^{-1} = f^{-1} \cdot f = 1_F$;*

- *the multiplication is distributive over addition, i.e.* $\forall f,g,h \in F \quad f \cdot (g+h) = f \cdot g + f \cdot h$.

A finite field $\mathbb{F}_q$ is a field with q elements and it exists if and only if q is prime or a prime power. Tipically, cryptographic messages are defined as belonging to $\mathbb{F}_2$, such that they are represented as binary numbers and they are equipped with the OR and AND operations (corresponding to $+$ and $\cdot$).
A ring is the generalization of a field:

Definition 1.1.3 (Ring) *A ring R is a field whose multiplication does not need to be commutative and whose elements do not need to have the multiplicative inverse.*

Then, we also need to specify what an ideal is.

Definition 1.1.4 (Ideal) *Given a ring* $(R,+,\cdot)$*, an ideal is a subset* I *of* R *that is a subgroup of the additive group of* R *and such that* $\forall x \in I \; rx, xr \in I \; \forall r \in R$.

Given an ideal I, the relation

$$x \sim y \text{ if and only if } x - y \in I$$

is an equivalence relation on R, and the equivalence classes form a set called the quotient of R by I, denoted by R/I. For example, the set of integers $\mathbb{Z}$ forms a ring with ordinary addition and multiplication. The set $3\mathbb{Z}$ formed by the multiples of three forms an ideal, and the quotient $\mathbb{Z}_3 := \mathbb{Z}/3\mathbb{Z}$ has only three elements: $\{0,1,2\}$, each one representing an equivalence class composed by the numbers equivalent to 0,1 and 2 mod 3. One of the most used quotients in Homomorphic Encryption is the real torus $\mathbb{T}$, which is the set $\mathbb{R}/\mathbb{Z}$ of real numbers modulo 1. $\mathbb{T}$ is a group, when using the addition.

The most commonly employed rings in Homomorphic Encryption are the polynomial rings.

Definition 1.1.5 (Ring of polynomials) *Given the set* $\mathbb{Z}[x]$ *of polynomials with integer coefficients and given a monic polynomial* $f(x)$*, a quotient ring* $R = \mathbb{Z}[x]/\langle f(x) \rangle$*, is a ring of polynomials with integer coefficients modulo* $f(x)$.

If the coefficients of the polynomial are in $\mathbb{Z}_q$ then we denote it R_q. It is worth pointing out that the ring R is also a field if and only if $f(x)$ is an irreducible polynomial over $\mathbb{Z}$.

After this brief section, we are now ready to start focusing on cryptography.

1.2 General Principles of Cryptography

Basic cryptography features

Cryptography is a discipline that provides the principles, techniques, and methodologies for securing communication and data in the presence of adversaries. It derives from the Greek words "kryptos", meaning hidden, and "graphein", meaning writing. At its core, cryptography is concerned with the construction and analysis of protocols to prevent third unauthorized parties from reading private messages. In this framework, encryption of data is usually based on two functions:

- Encryption function Enc, which takes as input a plaintext and a key and outputs the ciphertext;
- Decryption function Dec, the inverse of the encryption function, which takes as input a ciphertext and a key and outputs the original plaintext.

An encryption scheme can be *symmetric* or *asymmetric*, as stated in the following definitions:

Definition 1.2.1 (Symmetric cryptography) *An encryption algorithm is symmetric if it uses the same cryptographic keys k for both the encryption of plaintext and the decryption of ciphertext. If k is the key and m is the message, we have that*

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m.$$

Definition 1.2.2 (Asymmetric cryptography) *An asymmetric scheme features a key pair made up of a public key pk and a secret key sk, both belonging to a specific user and linked by some mathematical relation. The encryption function, denoted by Enc_{pk}, is public, so everyone can encrypt a message and send it to the user U using his public key pk. Then, if U wants to decrypt the message, he will use its own decryption function Dec_{sk}, employing its private key sk, to recover the original plaintext. In other words, given a message m, it yields*

$$\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m.$$

For symmetric schemes, intercepting k, which has to be shared between sender and recipient, is enough to break the security of the protocol. On the other hand, asymmetric systems do not present this issue: there is no key exchange, because each user can generate the pair (pk, sk) by itself, employing a certain key generation algorithm. Plus, in this case the number of required keys to perform secure exchanges among n users is $2n$, which is far less than those needed for the symmetric paradigm, namely $\binom{n}{2} \sim n^2/2$. However, asymmetric schemes are usually considerably slower than symmetric ones: for this reason, in real-world applications the two technique are combined. Namely, asymmetric cryptography is employed for key exchange, i.e. securely exchanging the keys used for symmetric encryption, while symmetric cryptography is used to encrypt the messages.

Cryptography usage and limitations

With the necessity of exchanging larger and larger masses of information, cryptography has become more and more indispensable, serving as the cornerstone of secure communications and information security. Its importance is multifaced, addressing critical aspects such as:

- *Confidentiality*: shared data must be inaccessible to unauthorized users, in order to safeguard sensitive information, ensuring that such information remains private until it reaches its intended recipient.
- *Integrity*: during transmission or storage, it has to be verified that data received by the recipient has not been altered, ensuring that it is exactly as it was sent.

- *Authentication*: before granting access to sensitive information or systems, verifying their identity of users and devices involved is fundamental to prevent impersonation.
- *Non-repudiation*: in any communication established, the sender of data has to be provided with proof of delivery and the recipient must be provided with proof of the sender's identity, so neither can later deny having processed the data.

Integrity of a message can be ensured by exploiting some algorithms called *hash functions*.

Definition 1.2.3 (Hash function) *Hash functions are algorithms that take an arbitrary size input and return a fixed-size string of bytes, said a hash value or digest. They are deterministic, thus the same input will always produce the same output, and even a small change in the input will produce a completely different hash value.*

These functions participate in the creation of *digital signatures*, which are employed to secure non-repudiation:

Definition 1.2.4 (Digital signatures) *Digital signatures use asymmetric cryptography to create a distinctive cryptographic object that can only be generated by the private key holder, but can be verified by anyone with access to the corresponding public key. They work as follows:*

- *Key generation: each user generates a pair of keys: a private key and a public key.*
- *Signature: the sender (who is the signer) uses his private key to produce the digital signature.*
- *Verification: when the receiver (the verifier) receives the signed message, he uses the signer's public key to check if the signature is valid.*

Finally, authenticity of data is addressed by *Message Authentication Code* (MAC).

Definition 1.2.5 (MAC) *Message Authentication Code (MAC) is a system which generates authentication tags of a fixed length by processing a message m: the resulting computation is the m's MAC. This tag is generated by using a secure symmetric key $\overline{k}$, only known to the sender and the recipient, and it is appended to the message and transmitted to the recipient. Then, the recipient verifies the authencity of the message employing $\overline{k}$ and the tag.*

All these resources can ensure secure and reliable exchange of information, but, depending on the usage, data needs different kinds of protection. More specifically, data can be classified in three different types:

- *Data-in-transit*: this type of data needs to be transferred between two parties. The major concerns here are to avoid data tampering, where unauthorized parties alter the data during transmission, and to prevent third parties from unauthorized access to sensitive information like credentials. On Internet, the most employed method to protect data in this sense is the TLS protocol [1]. In particular, HTTPS is an implementation of TLS encryption added to the HTTP protocol, which is used by nearly all websites. Any website that uses HTTPS therefore uses TLS encryption.

- *Data-at-rest*: this data does not actively move across networks or systems, but it needs to be securely stored. It is particularly vulnerable to breaches, theft, and unauthorized access, especially in environments like data centers, cloud storage, and local device storage. In order to protect this static data, Windows offers a functionality called BitLocker [2] providing encrypton for large volumes of data, which deals with threats of data theft or exposure from lost or stolen devices.
- *Data-in-use*: this kind of data needs to be securely processed. At the moment, the only way to process encryptedd data is to decrypt it, sacrificing privacy. In order to avoid this, it would be necessary to be able to operate on data maintaining it encrypted.

A *client-server* architecture presents all these three types of data. In this scenario, each client sends encrypted data to a server, requiring *data-in-transit* protection. This information can be stored using users' keys (*data-at-rest* protection is needed), but making any operations on data by keeping it encrypted, i.e. achieving *data-in-use* protection, is a big challenge. This is because, in general, performing operations on encrypted data does not match the result obtained by making the same operations on the unencrypted data. If data were directly elaborated in its encrypted version, a client would be able to make the server do any computations while keeping his data private: this is the problem Homomorphic Encryption tries to solve.

Chapter 2

Homomorphic Encryption

This chapter introduces the Homomorphic Encryption framework, dwelling on its developments and employments. Specifically, section 2.1 details the evolution of Homomorphic Encryption, which eventually lead to FHE. After a brief analysis of the security aspects in section 2.2, we follow a historical perspective, introducing the most popular schemes (section 2.3) and the libraries that implement FHE techniques (section 2.4). Finally, in section 2.5 we explore some practical FHE applications, ranging from enhancing privacy in medical research and cloud computing to secure data retrieval in search engines.

2.1 Evolution

The first idea of Homomorphic Encryption (HE) was introduced in 1978 [3]. This type of encryption represents a class of methods offering a transformative capability in the field of secure data processing. It permits computations to be performed on ciphertexts, generating an encrypted result which, when decrypted, matches the result of operations as if they were performed on the plaintext. This property is paramount for several reasons, particularly in enhancing privacy and security. In a more and more cloud-dependent world, data breaches and unauthorized access to sensitive information constitute an important issue, and Homomorphic Encryption provides a robust framework for preserving the confidentiality of data. It enables the processing of encrypted data without necessitating its decryption, ensuring that the underlying data remains private throughout its lifecycle. This is especially critical in sectors where the protection of personal and sensitive information is essential: HE enables data to be shared across organizations or geographies without revealing the underlying information.
For example, in the medicine field, healthcare providers can analyze patient data across networks while complying with privacy regulations by using encrypted data.
Another main context of employment is finance: FHE can be a game-changer in credit risk assessment by banks. In this application, HE allows a bank to compute an individual's credit score based on encrypted income, assets, debts, and spending habits provided by various financial sources, without revealing the actual values of sensitive data.
Here is a more formal definition for HE:

Definition 2.1.1 (Homomorphic Encryption) *Given two plaintexts m_1 and m_2, an encryption algorithm is said to be homomorphic with respect to two operations $\times_1$ and $\times_2$ if*

$$\mathsf{Enc}(m_1) \times_1 \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 \times_2 m_2), \tag{2.1}$$

where Enc *is the encryption function.*

Unfortunately, in general this property can be subject to some constraints. For this reason, HE is usually described by three subtypes:

- *Partially Homomorphic Encryption* (PHE), if this relation holds just for one specific operation, no matter how many times it is performed;
- *Somewhat Homomorphic Encryption* (SHE), if (2.1) is valid for certain operations among all the possible ones on the plaintext, but there are some limitations on the number of times they can be carried out;
- *Fully Homomorphic Encryption* (FHE) if the scheme possesses this property for every type of operation, with no restrictions at all.

Partially Homomorphic Encryption

The RSA scheme [4] is an example of a PHE system. In this protocol, the sender A acts as follows:

- he chooses two prime numbers p and q;
- he computes $N = pq$ and $\phi(N) = |\{n \in \mathbb{N} : n < N,\ \gcd(n, N) = 1\}| = (p-1)(q-1)$;
- he picks $e \in \mathbb{Z}^*_{\phi(N)} = \{x \in \mathbb{Z}_{\phi(N)} \mid x < \phi(N),\ \gcd(x, \phi(N)) = 1\}$ and calculates d such that $ed \equiv 1 \mod \phi(N)$;
- the couple $(\phi(N), d)$ will constitute the private key, whereas (N, e) will be the public key.

If A wants to encrypt a message m, he will use the function $\mathsf{Enc}(m) := m^e \mod N$ and send this to another user B; for decryption, given a ciphertext c, B will employ $\mathsf{Dec}(c) := c^d \mod N$. Because of the nature of the encryption function, given two plaintexts m_1 and m_2, it trivially yields:

$$\mathsf{Enc}(m_1 \cdot m_2) = (m_1 \cdot m_2)^e \mod N = (m_1)^e \cdot (m_2)^e \mod N = \mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2).$$

Of course, it does not work the same way with the sum operation, because the power of a sum is not the sum of the powers of its terms.

Somewhat Homomorphic Encryption

An example of SHE is the BGN scheme [5], which can be described as follows:

- *Key generation*: The public key is (n, G, G_1, e, g, h), where G, G_1 are groups of order $n = q_1 q_2$, with q_1 and q_2 prime numbers. g is a generator of G, and $e : G \times G \to G_1$ is a bilinear map such that $e(g, g)$ is a generator of G_1. h is defined as $h = u^{q_2}$, where u is a randomly chosen generator of G, different from g. It follows that h is the generator of the subgroup of G with order q_1: this number q_1 will be kept hidden as the secret key.

- *Encryption*: To encrypt a message m, a random number r from the set $\{0, 1, \dots, n-1\}$ is picked and m is encrypted as follows:

$$c = \mathsf{Enc}(m) = g^m h^r \mod n.$$

  We assume the message space consists of integers in the set $\{0, 1, \dots, T\}$ with $T < q_2$. In most applications, bits are encrypted, thus $T = 1$.

- *Decryption*: To decrypt the ciphertext c, one first computes $c' = c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$ (because $h^{q_1} \equiv 1 \mod n$) and $g' = g^{q_1}$ using the secret key q_1. Then, decryption is completed as follows:

$$m = \mathsf{Dec}(c) = log_{g'} c'$$

  Note that decryption in this system takes polynomial time in the size of the message space T. Therefore, the system as described above can only be used to encrypt short messages.

The homomorphic properties belonging to this protocol are:

- *Homomorphism over addition*: Homomorphic addition of plaintexts m_1 and m_2 using ciphertexts $E(m_1) = c_1$ and $E(m_2) = c_2$ are performed as follows:

$$c = \mathsf{Enc}(m_1)\mathsf{Enc}(m_2)h^r = (g^{m_1} h^{r_1})(g^{m_2} h^{r_2})h^r = g^{m_1+m_2} h^{r'} = \mathsf{Enc}(m_1 + m_2),$$

  where $r' = r_1 + r_2 + r$ is uniformly random just like r, because it is essentially a traslation of r itself. It can be seen that $m_1 + m_2$ is easily decryptable from the resulting ciphertext c.

- *Homomorphism over multiplication*: To perform homomorphic multiplication, use g_1 with order n and h_1 with order q_1 and set $g_1 = e(g, g), h_1 = e(g, h)$, and $h = g^{\alpha q_2}$. Then, the homomorphic multiplication of messages m_1 and m_2 using the ciphertexts $c_1 = \mathsf{Enc}(m_1)$ and $c_2 = \mathsf{Enc}(m_2)$ are computed as follows:

$$c = e(c_1, c_2)h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2})h_1^r = g_1^{m_1 m_2} h_1^{m_1 r_2 + r_2 m_1 + \alpha q_2 r_1 r_2 + r} = g_1^{m_1 m_2} h_1^{r'}$$

It is seen that r' is uniformly distributed like r and so $m_1 m_2$ can be correctly decrypted from resulting ciphertext c. However, c is now in the group G_1 instead of G. Therefore, another homomorphic multiplication operation is not allowed in G_1 because the bilinear map e is not available anymore. Nevertheless, resulting ciphertext in G_1 still allows an unlimited number of homomorphic additions.

Fully Homomorphic Encryption

The very first FHE scheme was proposed in 2009 by Craig Gentry in his PhD thesis [6]. One of the issues he tackled is how to deal with the order of magnitude of the random noise. The problem is that this noise increases any time we perform a homomorphic operation, and it affects the possibility to correctly decrypt the ciphertext. To overcome this limitation, Gentry introduced a new technique, called *bootstrapping*, structured as follows. Let $\mathcal{E}$ be a cryptographic scheme: consider two pairs of keys (sk_1, pk_1) and (sk_2, pk_2). Let us denote with Enc the encryption algorithm such that $c = \mathsf{Enc}_{pk_1}(m)$ encrypts m under pk_1, with Dec the decryption algorithm, and with Eval_{evk} the evaluation algorithm. This takes as input the public evaluation key evk, a function f and tuple of ciphertexts $(c_1, \ldots, c_t)$. It outputs a ciphertext c_f, such that it decrypts to the result of the evaluation of $(m_1, \ldots, m_t)$ over f, i.e. $c_f = \mathsf{Eval}_{evk}(f, (c_1, \ldots, c_t))$ and $\mathsf{Dec}_{sk}(c_f) = f(m_1, \ldots, m_t)$. Gentry's procedure is articulated in three steps:

- Encrypting sk_1 under pk_2:
$$\mathsf{Enc}_{pk_2}(sk_1) = \overline{sk_1}.$$

- Encrypting the ciphertext c under pk_2:
$$\mathsf{Enc}_{pk_2}(c) = \overline{c}.$$

- Decrypting homomorphically the new ciphertext using the encrypted secret key, such that an encryption of the same message under the second public key $\mathsf{Enc}_{pk_2}(m)$ is obtained. Namely:
$$\begin{aligned}\mathsf{Eval}_{evk}(\mathsf{Dec}, \overline{c}, \overline{sk_1}) &= \mathsf{Eval}_{evk}(\mathsf{Dec}, \mathsf{Enc}_{pk_2}(c), \mathsf{Enc}_{pk_2}(sk_1)) = \\ &= \mathsf{Enc}_{pk_2}(\mathsf{Dec}_{sk_1}(c)) = \mathsf{Enc}_{pk_2}(m).\end{aligned}$$

This method works also if we apply a certain function φ to the message before encrypting it.
Figure 2.1 schematizes the whole process, with particular attention on the error, represented by the bar on the right of each box. As noticeable from the image, the error of the obtained ciphertext is higher than a fresh ciphertext obtained with the encryption algorithm, but lower than the starting ciphertext obtained after homomorphic evaluating functions with higher depth than the decryption circuit. The dotted line represents the maximum error limit: beyond that threshold, decryption is not correct anymore.

There exists also a tecnhique which is able to evaluate a function at the same time as it reduces the noise, which is called *programmable bootstrapping* [8]. Essentially, given a message m and a function f, this method outputs a ciphertext encrypting $f(m)$. This is highly advantageous, as it enables FHE schemes to handle more complex functions with lower computational overhead. Of course, bootstrapping is a special case of programmable bootstrapping when f is the identity function.

It is worth considering that Homomorphic Encryption schemes can be computationally intensive and inefficient. In order to mitigate this issue, Gentry introduced the so-called *squashing* technique. This method involves transforming the decryption function into a

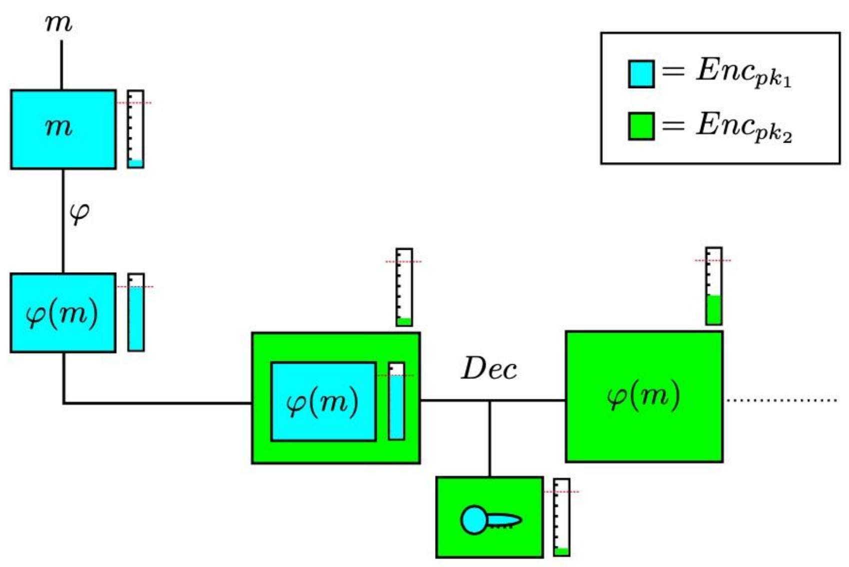


Figure 2.1. Idea of the bootstrapping procedure [7]

simpler form that requires less computational effort to execute. This is typically achieved by introducing additional structure into the encryption scheme or by applying certain transformations that reduce the complexity of the decryption circuit. The goal is to make the decryption process feasible and efficient while maintaining the security of the scheme.

2.2 Security

A homomorphic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, with KeyGen denoting the key generation algorithm, is secure if and only if it is *semantically secure*, i.e. indistinguishable under chosen-plaintext attack (IND-CPA security), where an attacker can obtain encryptions of arbitrary plaintexts, but it cannot decrypt arbitrary ciphertexts.

The definition is formalized as a game between two players, which for the case of public key encryption is the following:

Definition 2.2.1 (IND-CPA) *Let C be the challenger, and A be the adversary. Considering an encryption scheme $\mathcal{E}$, C and A act as follows:*

1. *C generates a keypair based on some security parameter k, e.g. a key size in bits, and sends the public key pk to the adversary A, keeping hidden the secret key sk.*
2. *A selects a pair of messages m_0, m_1 of equal length and sends them to the challenger.*
3. *C picks a random bit $b \in \{0,1\}$ and encrypts the message m_b as $c = \mathsf{Enc}_{pk}(m_b)$. He sends back c to the adversary.*

4. A can make a polynomial number of queries to the encryption oracle, and eventually he has to come up with a guess b'.

The scheme is told to be IND-CPA secure if, for all possible (time-bounded) adversaries, it holds that

$$\mathbb{P}[b' = b] - \frac{1}{2} = \epsilon(k),$$

where $\epsilon(k)$ is a negligible function in the security parameter k, that is for every nonzero polynomial function f there exists k_0 such that $|\epsilon(k)| < |\frac{1}{f(k)}|$ for all $k > k_0$.

One obvious note about the IND-CPA game is that the attacker has the public key, so the following strategy can be proposed for winning the game:

- The adversary picks two messages m_0, m_1 and then encrypts both of them using the public key.
- When the adversary receives the ciphertext c, he just compares that ciphertext to the two he generated himself.

Proceeding like this, the adversary can always figure out which message was encrypted. Therefore, the implication is that in order to satisfy the IND-CPA definition, any public-key encryption scheme must be randomized. Namely, it must take in some random bits as part of the encryption algorithm, and it must use these bits in generating a ciphertext. This implies that encrypting the same message multiple times gives different ciphertexts as a result.

Furthermore, if we want bootstrapping to be performed securely, the following property is also fundamental.

Definition 2.2.2 (Circular security) *An encryption scheme that is secure against adversaries who observe an encryption of the scheme's secret key under its public key is called circular secure.*

Unfortunately, this is not proven to hold on most FHE schemes: there exist a formal demonstration just for some protocols, like [9] and [10]. Thus, it is in general taken as an assumption on top of the scheme's underlying security assumptions.

Optionally, the system can be required to be *function private.*

Definition 2.2.3 (Function privacy) *A FHE scheme is function private when the evaluation of a function f homomorphically over a ciphertext does not reveal any information about the function itself, beyond the outputs for the queried inputs.*

Note that for a scheme to be function private, the property has to hold even against an adversary that knows the secret key and can decrypt any ciphertext.

2.3 Developments

Gentry's innovation has stimulated many researches on this topic, as summarized in [11]: the FHE schemes developed after 2009 can be divided into four generations.

First generation

The first generation is essentially composed of two families: the first one is characterized by schemes based on ideal lattices, the second one on the Approximate Greatest Common Divisor (AGCD) problem. Lattices are mathematical structures consisting of points in space that are arranged in a regular grid-like pattern. More specifically, a k-dimensional lattice is a discrete additive subgroup of $\mathbb{R}^n$.

Definition 2.3.1 (Lattice) *Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$ be linearly independent vectors in $\mathbb{R}^n$, then the lattice generated by B is*

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{k} \gamma_i \mathbf{b}_i : \quad \gamma_i \in \mathbb{Z}, \mathbf{b}_i \in B \right\}.$$

Ideal lattices are a special type of lattice that are derived from ideals in ring theory. In particular:

Definition 2.3.2 (Ideal lattice) *An ideal lattice is an integer lattice $\mathcal{L}(B) \subseteq \mathbb{Z}^n$ where $B = \{g \mod f : g \in I\}$, $I \subseteq \mathbb{Z}[x]/\langle f \rangle$ is an ideal, and f a monic polynomial of degree n.*

The use of ideal lattices in cryptography offers advantages in terms of efficiency and security. They allow operations to be performed more quickly than general lattice-based systems, making them well-suited for constructing FHE schemes. Every scheme in this category has a common origin, which is Gentry's initial idea itself. It was initially implemented by [12], and then improved by other studies, like [13], [14] and [15].

On the other hand, the AGCD problem can be formulated as follows. Let us say that one chooses a secret integer p, then samples n random integers q_i and defines $m_i = pq_i \ \forall i = 1, \ldots n$. Now, given these m_i, it is easy to recover p as $\gcd(m_1, \ldots, m_n)$. We talk about AGCD problem if we are given just "approximate multiples" of p, i.e. $x_i = pq_i + r_i \ \forall i = 1, \ldots n$, where each r_i is a small integer.
Here, the pioneers are van Dijk, Gentry, Halevi and Vaikuntanathan, who introduced a Fully Homomorphic Encryption scheme over integers, named DGHV [16], simpler than the ideal-lattice based one. The basic construction of their idea is the following:

- *Key generation*: outputs the secret key p, i.e. an odd random integer, and the public key $(x_0, \ldots, x_n)$ where x_0 is odd and $x_0 > x_i = pq_i + r_i \ \forall i$, with q_i, r_i random integers.
- *Encryption*: the message $m \in \mathbb{F}_2$ is encoded into the ciphertext $c = (m + 2r + 2\sum_{i \in S} x_i) \mod x_0$, where r is a random integer and S is a random subset of $\{1, \ldots, n\}$.
- *Decryption*: computes $(c \mod p) \mod 2$.

Other algorithms in this family essentially consist of different optimizations of DGHV.

Second generation

The second generation schemes can also be divided into two classes. The first one is based on the Learning With Errors (LWE) and Ring Learning With Errors (RWLE) problems, the second one on the NTRU lattice-based encryption method [17]. Given a vector $b \in \mathbb{Z}_q^m$ and a matrix $A \in (Z_q)^{m \times n}$, the LWE problem consists in finding $s \in \mathbb{Z}_q^n$ such that

$$As + e = b \mod q,$$

where $e \in \mathbb{Z}_q^m$ is sampled coordinate-wise from an error distribution χ. The RLWE problem is the reformulation of LWE in ring theory. Let us define $R_q = \mathbb{Z}_q[x]/\langle f(x) \rangle$, where $f(x) \in \mathbb{Z}[x]$ is a monic, irreducible polynomial of degree d and q is a prime. We are given an arbitrary number of independent samples $(a, b = s \cdot a + e) \in R_q \times R_q$, where a is chosen uniformly at random in R_q, and $e \in R_q$ is sampled from an error distribution χ. The goal of the RWLE problem is to discover the vector $s \in R_q$ used to generate the samples.
BV [18] is an example of LWE-based system, which is structured as follows:

- *Encryption*: the message $m \in \mathbb{F}_2$ is encoded into a ciphertext c such that $c = (a, b = \langle a, s \rangle + 2e + m) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where e is the error randomly chosen from an error distribution χ and $s \in \mathbb{Z}_q^n$ is the secret key composed of random elements in $\mathbb{Z}_q$.

- *Decryption*: outputs the plaintext $(b - \langle a, s \rangle \mod q) \mod 2$, which is equal to $(2e + m \mod q) \mod 2$. The decryption works properly if $e < q/2$.

A more practical version of this scheme was proposed by Brakerski, Gentry and Vaikunthanathan (BGV) in [19]. At the same time, Fan and Vercauteren also optimized this protocol by implementing the FV system [20]. These schemes are *scale-invariant*, i.e. their security and correctness do not depend on the ciphertext "scale" or modulus size, allowing efficient homomorphic operations at multiple scales without needing to change the underlying security parameters.

The other class of schemes is based on NTRU [17], a protocol which was first presented by Hoffstein, Pipher, and Silverman in 1996. From its introduction, its security features sparked debates within the academic community. This continued until 2011, when Stehle and Steinfeld [21] made minor adjustments to the original protocol, resulting in a version whose security relies on the RLWE assumption. Following this, in 2012, Lopez-Alt, Tromer, and Vaikuntanathan developed the first FHE scheme [22], drawing inspiration from the Stehle-Steinfeld modification of NTRU.
Second generation schemes feature an efficient data-packing technique, which allows multiple plaintext data values to be encrypted into a single ciphertext. This innovation significantly improves the efficiency of homomorphic operations, especially in applications that process large datasets. They utilize a structure known as Single Instruction, Multiple Data (SIMD) within the ciphertext, allowing multiple slots in the plaintext space to store different data values. As a result, a single homomorphic operation on the ciphertext is performed across all slots simultaneously, as if they were individual elements in parallel.

Third generation

Third generation schemes include a second family of LWE schemes, which started with Gentry, Sahai and Waters [23] (GSW). Their construction is as follows:

- *Key generation*: outputs the secret key $\mathbf{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{Z}_q^n$, where $s_1 = 1$ and the s_i's are chosen at random and a public key $A \in \mathbb{Z}_q^{n \times n}$ such that $A\mathbf{s} = \mathbf{e} \approx 0$;
- *Encryption*: computes the ciphertext $C = mI_n + RA$, where $m \in \mathbb{Z}_q$ is the message, I_n is the identity matrix, R is a random matrix with size $n \times n$ and with coefficients in $\mathbb{F}_2$.
- *Decryption*: First, computes $C\mathbf{s} = mI_n\mathbf{s} + RA\mathbf{s}$, which is approximately equal to $mI_n\mathbf{s}$, because R has small entries (belonging to $\mathbb{F}_2$) and $A\mathbf{s} \approx 0$. Finally, outputs the first element of the vector $x \approx mI_n\mathbf{s} \approx (ms_1, \ldots, ms_n)$, which is m, since $s_1 = 1$.

There exists also the corresponding RWLE version of this scheme, created by Khedr, Gulak and Vaikuntanathan and illustrated in [24].

It is also worth mentioning another important scheme, which consists of three components realizing FHE on the Real Torus $\mathbb{T}$:

- TLWE, which is a generalized version of the LWE problem for the Torus;
- TRLWE, which is its ring variant;
- TRGSW, which improves the ring version of GSW scheme.

These three methods are usually grouped under the name TFHE [25].

Fourth generation

In 2017, Cheon, Kim, Kim, and Song [26] unveiled a new wave of FHE frameworks, introducing a technique for creating an encryption system tailored for Approximate Arithmetic Numbers. This innovation was accompanied by the release of an open-source library for the scheme's implementation. Initially dubbed HEAAN, the scheme later became known as CKKS, an acronym derived from the last names of its creators, while HEAAN now specifically refers to the library implementing this scheme. A year following its introduction, the method was enhanced to support FHE by Cheon, Han, Kim, Kim, and Song [27].

The fourth-generation schemes share similarities with their second-generation counterparts, with the primary distinction being their reliance on approximate computations. This approach significantly boosts speed by embedding the message space within a complex hyperplane and incorporating encryption errors as part of the calculation's inherent approximation error. A notable aspect of CKKS is its ability to perform homomorphic operations on approximate real numbers, rendering it particularly effective for floating-point arithmetic tasks. Like its predecessors of second generation, this one also features efficient data packing methods and is optimized for quick addition and multiplication operations, though any nonlinear operation demands considerable computational resources.

Summarizing considerations

Among all the presented schemes, the most effective and popular ones currently are BGV, BV, FV, TFHE and CKKS. The BGV, BV and FV schemes, belonging to the second generation, are optimized for operations within finite fields using modular exact arithmetic. These schemes come with efficient data packing capabilities, making them ideal for processing large datasets in parallel.

However, for scenarios requiring bootstrapping, or the implementation of non-linear functions, second-generation schemes fall short. In such cases, third-generation schemes, particularly TFHE, are recommended. TFHE excels in bit-wise operations and is more suited for computations modeled as boolean circuits, whereas it limits its efficiency in handling large-scale parallel data processing compared to its predecessors.

For arithmetic operations involving real numbers, the fourth-generation CKKS scheme stands out as the superior choice. This scheme is specifically tailored for handling computations with real numbers, offering a significant advantage in scenarios requiring such operations.

2.4 Implementations

The primary goal of FHE libraries is to offer access to FHE scheme operations through an Application Programming Interface (API). In addition to the fundamental capabilities offered by key generation, encryption, decryption and evaluation, most prominent libraries also include extra functionalities. These functionalities facilitate the management and manipulation of ciphertexts, specifically addressing the challenge of noise growth during computations, and provide methods for homomorphic addition and multiplication. A vast majority of these libraries is written in C++, as those presented by [28], but there also some other languages. Table 2.1 shows an overview of some of the most recent implementations.

The SEAL library [29], for example, made open source by Microsoft towards the end of 2018, is a library with no dependencies for some two second-generation homomorphic schemes. In particular, it features the scale-invariant BV and FV schemes of Brakerski [30], with its implementation in [20], and the CKKS scheme, as introduced by Cheon et al. [26].

Another important library is HElib [31], designed and maintained by IBM. The first implementation included one for the BGV scheme [19] and provided support for data manipulation instructions, e.g. the packing techniques as introduced by Smart et al. [32]. As of the 1.0.0 beta release, HElib also includes partial support for the CKKS scheme.

We also cannot forget PALISADE [33], which is a general-purpose library providing implementations of various building blocks for lattice-based cryptography along with implementations of advanced lattice-based cryptographic protocols. This modular design approach makes it possible to achieve both implementations of standard protocols that can be used out of the box for building applications on top and as a platform for more advanced users, allowing experimentation and the possibility to combine their specific implementations with those provided by the library. In terms of HE scheme capabilities, PALISADE most notably implements the second generation BV and FV [20] and BGV [19] schemes and some of their variants.

Library	Languages	Schemes
SEAL	C++, C#	BV, FV, BGV, CKKS
Helib	C++	BGV, CKKS
PALISADE	C++	BV, FV, BGV, CKKS, FHEW, TFHE
OpenFHE	C++	BV, FV, BGV, CKKS, FHEW, TFHE
FHEW	C++	FHEW
TFHE	C++, C	TFHE
TFHE-rs	Rust	TFHE
Concrete	Python	TFHE
ConcreteML	Python	TFHE for Machine Learning
Lattigo	Go	BV, FV, CKKS

Table 2.1. Main FHE open-source libraries

In 2022, PALISADE's creator decided not to update anymore their library, but they implemented OpenFHE [34], which is its enhanced version. This library supports all common FHE schemes; its design is based on PALISADE, but it introduces some new design features, as detailed in [35]. For example, OpenFHE includes both user-friendly modes, where maintenance operations (such as bootstrapping) are automatically invoked by the library, and compiler-friendly modes where an external compiler makes these decisions. OpenFHE design also supports "importing" functionality from its predecessors, such as HElib.

Concerning the third-generation schemes, it is worth mentioning the FHEW library [36] by Ducas and Micciancio, even if it has not been updated since 2017, and the TFHE library [37]. This was provided by the authors of the TFHE paper [25], it is also available in C and is considered to be the successor of the FHEW library.
TFHE has been particularly studied by Zama [38], an open source cryptography company which has developed some libraries for this scheme. In particular, we have TFHE-rs [39], written in Rust, and Concrete [40], implemented in Python. The latter features a sublibrary, i.e. ConcreteML [41], which integrates privacy-preserving features of FHE into Machine Learning (ML) use cases. Its interface is very similar to Scikit-learn [42], a famous ML library written in Python.

Finally, Lattigo [43] is also a milestone: it was proposed by Mouchet, Bossuat, TroncosoPastoriza and Hubaux [44] and is the first library written in Go. It includes the implementation of the BFV, BGV and CKKS schemes.

2.5 Applications

Cloud computing

Cloud services are one of the most common applications, where client sends encrypted data to server (*data-in-transit* protection). This data can be securely stored using users' keys (*data-at-rest* protection) via the cloud storage functionality. Unfortunately, it is not

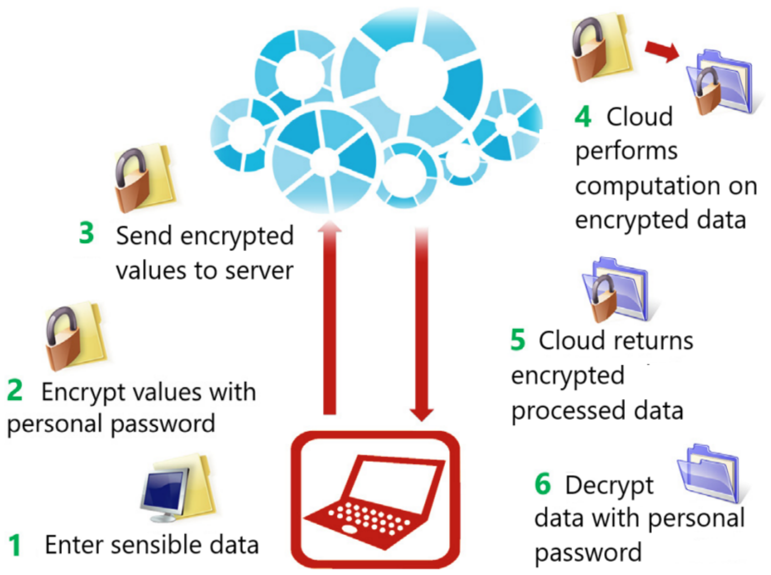

4 Cloud performs computation on encrypted data
3 Send encrypted values to server
2 Encrypt values with personal password
5 Cloud returns encrypted processed data
6 Decrypt data with personal password
1 Enter sensible data

Figure 2.2. Cloud service for privately computing operations on encrypted data

possible to exploit the potential of cloud engine to make any operations on data (no *data-in-use* protection) by keeping it encrypted in cloud. On the contrary, if data is directly elaborated in its encrypted version via the usage of HE, a client can make the server do some computations while keeping his data private. This idea is well represented by Figure 2.2, which outlines operations and information exchanges carried out during the user-cloud communication process.

In this scenario, a significant challenge is to enable collaborative computations between different users. Since ciphertexts coming from different users are encrypted with different keys, they need to be re-encrypted under a common key: this is done by a process known as Homomorphic Proxy Re-Encryption (HPRE). This method originates from Proxy Re-Encryption (PRE), a technique commonly utilized in cloud computing for traditional encryption methods, which facilitates the transformation of a ciphertext from one user (the delegator) to another (the delegatee) via a *proxy*. This is how it works:

1. A user encrypts a message with their public key. This encryption can only be decrypted by the owner's private key.

2. The user generates a special re-encryption key and gives it to the proxy. This re-encryption key is specific to the relationship between the data owner and the intended recipient.

3. The proxy uses the re-encryption key to convert the original ciphertext into a form that can be decrypted by the recipient's private key. Importantly, the proxy never learns anything about the underlying plaintext.

4. The recipient decrypts the re-encrypted message using their private key, gaining access to the plaintext without requiring the user's direct involvement.

This procedure allows the delegatee to decrypt the message intended for the delegator without accessing the original secret key. HPRE extends this functionality by enabling the cloud service to apply homomorphic operations on the re-encrypted ciphertexts, enhancing the utility of PRE in scenarios requiring encrypted computations.

In his thesis, Gentry outlines a straightforward method for implementing HPRE. This involves the delegator creating two specific ciphertexts: one that homomorphically encrypts the delegator's secret key using the delegatee's public key, and another that encrypts the actual data using the delegator's own public key. The proxy then applies the decryption algorithm of the homomorphic encryption scheme to the data ciphertext, effectively re-encrypting it for the delegatee's public key. This process is essentially analogous to the bootstrapping technique utilized in HE schemes.

Medical research

Collaboration in the medical research field among institutions of different countries can be very complex. This happens because of different rules in personal data treatment, often incompatible one with each other. HE allows to carry out studies while saving patients' privacy. For example, the Cryptography Research Team at Microsoft has developed a prototype system for assessing a patient's cardiovascular risk using their encrypted health information, as outlined in [45]. This system showcases a cloud-based service capable of executing predictive algorithms on secure, encrypted medical data. Specifically, it utilizes an algorithm to evaluate the risk of a heart attack by analyzing certain physical metrics. The architecture of this service includes a client-side application on a personal device and a cloud-based application hosted on the Microsoft Windows Azure platform. The client-side application gathers health metrics from the user, encrypts this data, and then transmits the encrypted information to the cloud service. There, the prediction algorithm processes the encrypted data without ever decrypting it, ensuring privacy. The cloud service then generates an encrypted output detailing the heart attack risk, which is sent back to the client application. The user has the ability to decrypt this result to understand their potential risk of experiencing a heart attack.

Search engines

Nowadays, maintaining privacy during web searches is a critical concern. Homomorphic Encryption (HE) offers a solution by keeping the search content hidden from the search engine. In this vein, Jie Li, Yamin Liu, and Shuang Wu introduced Pipa [46], a system that leverages homomorphic encryption to protect user privacy. Pipa's framework includes a server that holds a database of compromised accounts, storing only the hash values of account details rather than the actual usernames and passwords. To enhance search efficiency within this potentially vast database, the data is segmented into smaller blocks using the initial bits of each account's hash value, known as the hash prefix. The size of these blocks is carefully balanced to optimize homomorphic processing speed without compromising user privacy.

On the user side, a HE module is prepared during a pre-computation phase, generating necessary keys based on parameters received from the server. These keys are securely stored on the user's device.
When a user wants to check if his account details are compromised, the HE module calculates the hash of the username for the prefix and the full hash of the account information, encrypting the latter with the homomorphic encryption scheme. A checkup request is then sent to the server, including the hash prefix and the encrypted account hash.
Upon receiving a request, the server identifies a subset of hashes that match the provided prefix, and then determines if the account hash is in the database. Importantly, the server cannot decrypt the result.
Finally, the server returns the encrypted result to the user, where the HE module decrypts it and notifies the user of their account's status. This process ensures that the user's account details remain private, with the server only learning minimal, non-sensitive information.

Chapter 3

The Machine Learning Paradigm

This chapter's purpose is to introduce the main characteristics of Machine Learning (ML) and some of the most used methods. In particular, section 3.1 presents the general ideas behind ML, while section 3.2 describes the ML algorithms we decided to analyze.

3.1 Machine Learning Foundations

The primary objective of Machine Learning (ML) is to make predictions about an output or target variable y based on an input vector $\mathbf{x}$, whose components are called *features*. For instance, $\mathbf{x}$ could denote the weight and smoking status of a pregnant woman, while y might be the newborn's weight. This predictive endeavor is encapsulated in a mathematical model g, known as the prediction function, which accepts $\mathbf{x}$ as input and produces an estimate $\hat{y} = g(\mathbf{x})$ for y. Essentially, g captures the dynamics between $\mathbf{x}$ and y, discounting the influence of randomness and chance. From now on, we will denote vectors and matrices in bold type, and numbers in italics.

If y is restricted to a discrete set, such as $y \in \{0, \dots, c-1\}$, the task of predicting y equates to categorizing $\mathbf{x}$ into one of c distinct classes, transforming prediction into a classification challenge. The precision of a prediction $\hat{y} = g(\mathbf{x})$ relative to the actual response y can be evaluated using a specific loss function $\mathscr{L}(y, \hat{y})$. In classification scenarios, the zero-one loss function, $\mathscr{L}(y, \hat{y}) := \mathbb{1}_{\{y \neq \hat{y}\}}$, i.e.

$$\mathscr{L}(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y}, \end{cases}$$

is frequently utilized.
Among all the possible prediction functions g belonging to a certain class of functions $\mathcal{F}$, the best one would be the minimizer of the loss, i.e.

$$g^* = \operatorname*{argmin}_{g \in \mathcal{F}} \mathscr{L}(y, g(\mathbf{x})).$$

In this context, we usually have available a finite number of independent samples $(\mathbf{x}, y)$, interpretable as realizations of two random variables and coming from the joint density $f(\mathbf{x}, y)$. The optimal prediction function g^* depends on the unknown joint distribution, and our goal is to approximate it at best just by using our samples, which form the *training set*, defined as follows

Definition 3.1.1 (Training set) *The sample* $\tau = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ *is defined as the training set with* n *examples.*

All the vectors $\mathbf{x}_i$ belong to $\mathbb{R}^p$, where p is the number of features; thus, they can be represented as $(x_{i1}, \dots, x_{ip})$ and arranged to form the training matrix

$$\mathbf{X} = \begin{pmatrix} x_{11} & \dots & x_{1p} \\ \dots & \dots & \dots \\ x_{n1} & \dots & x_{np} \end{pmatrix}.$$

We represent the optimal approximation of g, based on a given criterion, using g_τ, which is derived from the training dataset τ. This scenario can be likened to a teacher-student dynamic. In this analogy, the "teacher" presents n instances demonstrating the true connection between the input $\mathbf{x}_i$ and the output y_i for $i = 1, \dots, n$, educating the student, g_τ, on making predictions for new inputs $\mathbf{x}'$ without the teacher specifying the correct output y' (which remains unknown). These new inputs constitute the so-called *test set*:

Definition 3.1.2 (Test set) *The sample* $\tau' = \{(\mathbf{x}'_1, y'_1), \dots, (\mathbf{x}'_{n'}, y'_{n'})\}$ *is defined as the test set with* n' *examples.*

This framework is identified as *supervised learning*: the process typically involves using $\mathbf{x}$, a set of explanatory variables, to predict y. The name *supervised* is due to the fact that the label of samples in the training set, called *ground truth*, is known.

In order to find the function g_τ, we usually minimize the *training loss* with respect to g, looking for the optimal function in a specific family $\mathcal{F}$.

Definition 3.1.3 (Training loss) *Given a training set* $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$*, the training loss is defined as*

$$\ell_\tau(g) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, g(\mathbf{x}_i)). \tag{3.1}$$

After having g_τ, we can use it to make inference, i.e. we can apply it to the data in the test set. We can then evaluate the predictions by computing the *test loss.*

Definition 3.1.4 (Test loss) *For any outcome* τ *of the training data, given a prediction function* g_τ *and a test set* τ'*, the test loss is*

$$\ell_{\tau'}(g_\tau) := \frac{1}{n'} \sum_{i=1}^{n'} \mathscr{L}(y'_i, g_\tau(\mathbf{x}'_i)).$$

There are also other metrics to measure how a classifier is performing: the most frequently employed is *accuracy* A, which is the fraction of correctly labelled samples in the test set, i.e.

$$A = \frac{1}{n'} \sum_{i=1}^{n'} \mathbb{1}\{y_i = \hat{y}_i\}.$$

This indicator is often preferred to the test loss because the former gives us a relative measure, whereas the latter computes an absolute measure. Namely, a large accuracy indicates a well-performing model, whereas a high test loss does not necessarily prove the model to be poor. Some other very common metrics are:

- *recall* of a class c, which is the ratio between the number of correctly classified samples of class c and the number of times that class occurs in the data sample.
- *precision* of a class c, which is the ratio between the number of correctly classified samples of class c and the number of times the classifier has predicted that class.
- *F1 score* of a class c, which is the harmonic mean of recall and precision for that class.

Note that these measures refer to a specific class, thus they are usually averaged through all possible labels to have a general idea of the performance of the algorithm.

In this context, we need a way to choose training and test sets. Normally, these sets are derived from the available data: assume we have N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. We can randomly partition this collection of samples into two parts and use one as the training set, and one as the test set (this procedure is called *hold-out*, and a standard proportion is 70%-30%). Then, if we want to establish the best among a certain class of learners, we can employ a standard technique, named *k-fold cross validation*. It basically consists in dividing the training set into k partitions. Then, $k-1$ of these partitions are used as the training set, and the remaining one as the test set, which is called *validation set*. The model is trained on the former and tested on the latter, and the procedure is repeated k times, choosing every time a different partition as the validation set, and the remaining data as the test set. At the end, performance is evaluated by taking the average accuracy of all k inferences (a tipical choice is $k = 5$ or $k = 10$). The idea is that in this way we can rank models, discriminating with respect to the hyperparameters we use. If we try different combinations of parameters, we will choose the one associated to the model which scores the highest accuracy. Eventually, the best-performing method is employed to make predictions on the test set.

3.2 ML Algorithms

After this general introduction, we are now ready to go into details of how some ML methods work. More specifically, we are going to describe the following algorithms:

- K-Nearest Neighbors in subsection 3.2.1;
- Support Vector Machines in subsection 3.2.2;

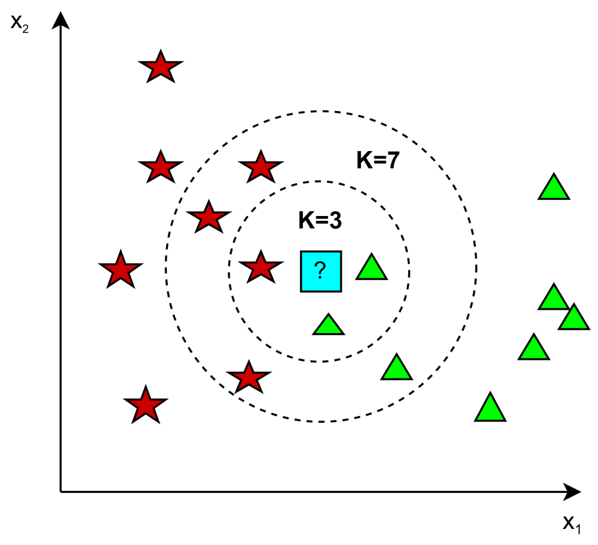


Figure 3.1. Working idea of the KNN algorithm

- Decision Tree and Random Forest in subsection 3.2.3.

All the following descriptions follow what is presented in [47].

3.2.1 K-Nearest Neighbors

Let $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training set, with $y_i \in \{0, \dots, c-1\}$ and $\mathbf{x}_i \in \mathbb{R}^p$, and let $\mathbf{q}$ be a new feature vector. The K-Nearest Neighbors (KNN) method works as follows:

1. Sort the training data by closeness to $\mathbf{q}$ in some distance; let $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ be the ordered feature vectors.
2. Choose the subset of τ that contains the K feature vectors that are closest to $\mathbf{q}$, say $\tau(\mathbf{q}) := \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(K)}; y^{(K)})\}$. The choice of this parameter K is quite important, because it directly affects the label to be assigned to $\mathbf{q}$.
3. Classify $\mathbf{q}$ by majority vote, according to the most frequently occurring class labels in $\tau(\mathbf{q})$. If two or more labels receive the same number of votes, the feature vector is classified by selecting one of these labels randomly with equal probability.

Figure 3.1 gives a graphical interpretation: as we can see, with $K = 3$ the new point, represented by the blue square with a question mark, would be classified as a green triangle, whereas with $K = 7$ it would be labelled as a red star. For this reason, K has to be properly tuned: a good choice is tipically a number which is not too small (otherwise the model is very sensitive to outliers) nor too high (because this can lead to underfitting, i.e. the model can be unable to generalize to new data). The new sample can be also labelled via a weigthed vote procedure, where the influence on the output class of each

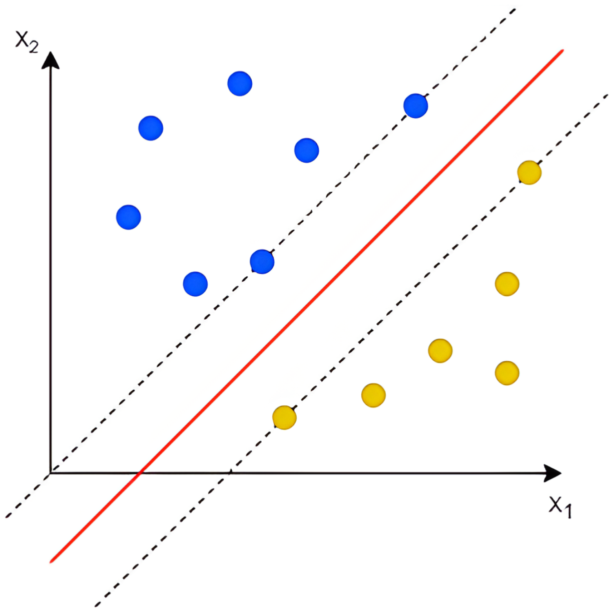


Figure 3.2. SVM plane division

sample $\mathbf{x}^{(i)}$ is inversely proportional to its distance from $\mathbf{q}$.
There exist many possible measures of distance, such as:

- the Manhattan distance $||\mathbf{x}^{(i)} - \mathbf{q}||_1 = \sum_{j=1}^{p} |x_j^{(i)} - q_j|$;
- the Euclidean distance $||\mathbf{x}^{(i)} - \mathbf{q}||_2 = \sqrt{\sum_{j=1}^{p} (x_j^{(i)} - q_j)^2}$;
- the Chebyshev distance $||\mathbf{x}^{(i)} - \mathbf{q}||_\infty = \sup_{j=1,\dots,p} |x_j^{(i)} - q_j|$.

3.2.2 Support Vector Machines

The next method we want to analyze is Support Vector Machines (SVM): suppose we are given the training set $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ where each response y_i can be -1 or 1, and we wish to construct a classifier taking those values. We are looking for a function

$$\begin{aligned} g : \mathbb{R}^n &\to \mathbb{R} \\ \mathbf{x} &\mapsto \theta_0 + \theta^T \mathbf{x}, \end{aligned}$$

which minimizes

$$\ell_\tau(g) = \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i g(\mathbf{x}_i)\}.$$

After having found the minimizer $g_\tau(\mathbf{x})$, we are simply taking as a classifier $\operatorname{sgn}(g_\tau(\mathbf{x}))$: this is a typical *binary classification* task.

Let us give a geometric interpretation to this problem: the data $\{\mathbf{x}_i\}_{i=1}^n$ can be seen as points in $\mathbb{R}^p$, where p is the number of features. For the sake of simplicity, let us initially assume the points are linearly separable, i.e. there exists an hyperplane dividing them into two different classes. The idea of SVM algorithms is to find the best hyperplane, which is the one maximizing the gap between it and the points, as shown in Figure 3.2. More formally, let us define

$$g_\mathcal{T}(\mathbf{x}) = \theta_0^* + \theta^{*T}\mathbf{x},$$

where θ^* is such that $g_\mathcal{T}(\mathbf{x}) = 0$ is the best separator hyperplane, called *decision boundary* (the red line in the image). Then, the two hyperplanes $g_\mathcal{T}(\mathbf{x}) = 1$ and $g_\mathcal{T}(\mathbf{x}) = -1$ represent the *margins* (the dotted lines in the image). The points lying on the margins are usually called *support vectors* (SV); without loss of generality, we can consider $\theta_0 = 0$. The scalar product $\theta^T\mathbf{x}$ can be written as:

$$\theta^T\mathbf{x} = ||\theta||_2||\mathbf{x}||_2 \cos\theta = r||\theta||_2,$$

where r is the length of the projection of the support vectors onto θ. Since for the support vectors $\theta^T\mathbf{x} = \pm 1$, then $r = \frac{1}{||\theta||_2}$, and the margin is simply $2r$. Therefore, if we want to maximize the margin, we will have to solve

$$\begin{aligned} &\min_\theta \frac{1}{2}\sum_{j=1}^p \theta_j^2 \\ &\text{s.t. } y_i(\theta^T\mathbf{x}_i) \geq 1 \; \forall i, \end{aligned} \tag{3.2}$$

In this formulation, the object to be minimized is often called *objective function.* The constraint is due to the following:

- if $\theta^T\mathbf{x}_i > 1$, it means that the point i is above the margin $g_\mathcal{T}(\mathbf{x}) = 1$ in the image, so it has to be classified with $y_i = 1$;
- if $\theta^T\mathbf{x}_i < -1$, it means that the point i is below the margin $g_\mathcal{T}(\mathbf{x}) = -1$ in the image, so it has to be classified with $y_i = -1$.

The optimal solution is usually found by employing the so-called *Lagrange function.* First of all, let us introduce the following theorem.

Theorem 3.2.1 (Lagrange multipliers) *Consider the optimization problem*

$$\begin{aligned} &\min_\theta f(\theta) \\ &s.t. \; g(\theta) = \mathbf{0}_m, \\ &\quad\;\; h(\theta) > \mathbf{0}_q, \end{aligned}$$

with $g : \mathbb{R}^p \to \mathbb{R}^m$ and $h : \mathbb{R}^p \to \mathbb{R}^q$, where m is the number of equality constraints and q the number of inequality constrains. Let us define

$$L(\theta, \alpha, \beta) = f(\theta) + \alpha^T g(\theta) + \beta^T h(\theta)$$

as the Lagrangian function, where $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^q$. If $\nabla f(\theta)$ and $\nabla g(\theta)$, and $\nabla f(\theta)$ and $\nabla h(\theta)$ are linearly independent, then there exists a unique vector of Lagrange multipliers (α^*, β^*) such that

$$\begin{cases} \nabla_\theta L(\theta^*, \alpha^*, \beta^*) = \mathbf{0} \\ \nabla_\alpha L(\theta^*, \alpha^*, \beta^*) = \mathbf{0} \\ \nabla_\beta L(\theta^*, \alpha^*, \beta^*) = \mathbf{0}. \end{cases}$$

If we solve the following optimization problem, named *dual problem*

$$\max_{\alpha,\beta} \min_\theta L(\theta, \alpha, \beta)$$

we get $(\theta^*, \alpha^*, \beta^*)$: strong duality states that if there exists a point θ strictly satisfying all constraints in (3.2), (α^*, β^*) coincides with the Lagrange multipliers vector, and θ^* is the solution of (3.2). This is true only if the initial problem is convex, such as in the case we are currently analyzing.
In this context, after having solved the dual problem, we obtain

$$g_\mathcal{T}(\mathbf{x}) = \sum_{i \in SV} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + \theta_0^*, \tag{3.3}$$

where α_i^* is the Lagrange multiplier associated to the i-th constraint and

$$\theta_0^* = \frac{1}{|SV|} \sum_{i \in SV} \left(y_i - \sum_{j \in SV} \alpha_j^* y_j \langle \mathbf{x}_i \mathbf{x}_j \rangle \right).$$

Unfortunately, this model is utopic, because it assumes that there are no points in the neutral region $\{\mathbf{x} : -1 < g_\mathcal{T}(\mathbf{x}) < 1\}$. Plus, the linear separability between points is not guaranteed at all.

Since these assumptions may lead to an unfeasible problem, we can introduce some slack variables $e_i > 0$ for $i = 1, \dots, n$ and modify (3.2) as follows:

$$\begin{aligned} &\min_{\theta, \mathbf{e}} \frac{1}{2} \sum_{j=1}^p \theta_j^2 + C f(\mathbf{e}) \\ &\text{s.t. } y_i(\theta^T \mathbf{x}_i) \geq 1 - e_i \ \forall i, \end{aligned} \tag{3.4}$$

where $f(\mathbf{e})$ is a merit function, which penalizes every unperfect classification by a factor C, i.e. a parameter to be tuned. Proceeding like this, we are allowing the model for misclassification error if $e_i > 1$, and for leaving some points in the neutral region if $e_i \in [0,1]$. The name of the classifier depends on the form of the merit function: if it is quadratic we talk about Least Square Support Vector Machines (LSSVM), if it is linear we refer to the method as Linear Support Vector Machines (LSVM). In this two cases, we usually have that $f(\mathbf{e}) = \frac{1}{2} \sum_{i=1}^n e_i^2$ and $f(\mathbf{e}) = \sum_{i=1}^n e_i$, respectively.

Nevertheless, even with this alternative formulation, we might achieve a poor performance. This is because there could exist a different coordinate system where it is easier to

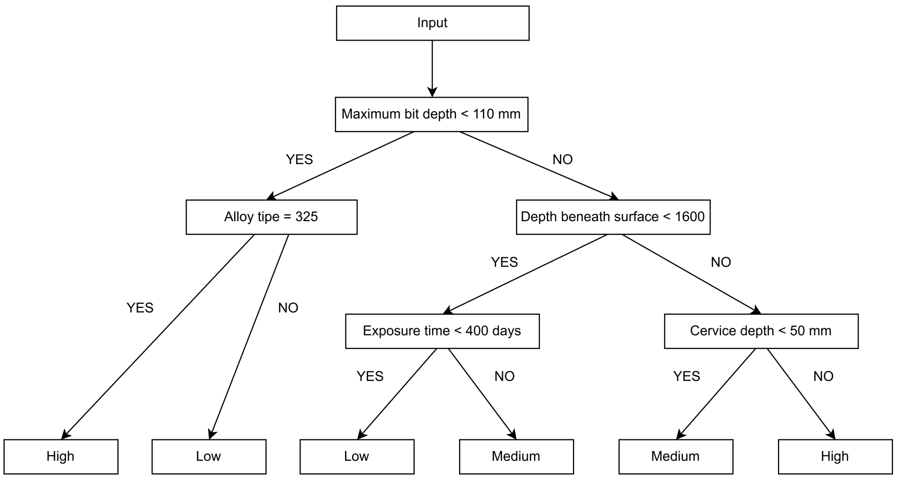


Figure 3.3. Sketch of a decision tree

find a hyperplane separating the data. In this case, we need a feature map Φ to represent the points in a new space. For example, if $\mathbf{x}_i \in \mathbb{R}^2$:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_{i1}, x_{i2}) \mapsto (x_{i1}, x_{i2}, x_{i1}^2 + x_{i2}^2)$$

If the transformation is not complex, it is enough to run the algorithm on $\{\Phi(\mathbf{x}_i)\}_{i=1}^n$, but sometimes this can result to be very expensive. For this reason, instead of computing $g_{\mathcal{T}}$ using $\langle\Phi(\mathbf{x}_i), \Phi(\mathbf{x})\rangle$, we replace this dot product with $K(\mathbf{x}_i, \mathbf{x})$. The purpose of this function, named *kernel*, is to calculate the dot product without having to evaluate Φ. Some of the most popular kernels are:

- the polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \langle\mathbf{x}_i, \mathbf{x}_j\rangle^q$;
- the Radial Basis Function (RBF) kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\sigma^2}\right)$ (σ is a hyper-parameter to be tuned);
- the cosine similarity kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{||\mathbf{x}_i||||\mathbf{x}_j||}$.

3.2.3 Decision Tree and Random Forest

The purpose of a binary decision tree (DT) is to partition the feature space $\mathcal{X}$ into a certain number of regions by using some *decision* or *splitting* rules, expressing a condition on a specific feature. Tipically, given a sample $\mathbf{x}_i$, the possible decision rules are $s(\mathbf{x}_i) = \mathbb{1}\{x_{ij} \leq k\}$ for numerical features and $s(\mathbf{x}_i) = \mathbb{1}\{x_{ij} \in S\}$ for categorical ones, where S is a subset of the support of feature j. The name of this classifier is due to the fact that every decision rule can be represented as a node of a tree, whose leafs are the above-mentioned regions of $\mathcal{X}$. A typical example of a decision tree is illustrated in Figure 3.3, where seawater corrosion of stainless steel is described in terms of:

- the type of stainless steel alloy,
- the depth in meters under the sea surface at which the corrosion took place,
- the period in days over which the metal alloy was exposed to seawater,
- the maximum depth in millimeters of pitting that was observed,
- the depth in millimeters to which crevices formed in the metal.

The class variable is the amount of corrosion that occurred, i.e. Low, Medium or High.

Let us denote the set of leaf nodes by $\mathcal{W}$. The overall prediction function g that corresponds to the tree can then be written as

$$g(\mathbf{x}) = \sum_{w \in \mathcal{W}} g^w(\mathbf{x}) \mathbb{1}\{\mathbf{x} \in \mathcal{R}_w\}, \tag{3.5}$$

where $\mathbb{1}$ denotes the indicator function, $\mathcal{R}_w$ is the subregion of $\mathcal{X}$ associated to the leaf w and g^w is the regional prediction function. In a classification setting with class labels $0, \ldots, c-1$, the regional prediction function for $w \in \mathcal{W}$ is chosen to be:

$$g^w(\mathbf{x}) = \underset{z \in \{0,\ldots,c-1\}}{\arg\max}\; p_z^w,$$

where p_z^w is the proportion of feature vectors in $\mathcal{R}_w$ that have class label z. For example, in Figure 3.3 the rightmost leaf is associated with Class High, implying that the majority of training samples with

- maximum pit depth greater than 110 mm;
- depth beneath surface greater than 1600 m;
- crevice depth greater than 50 mm

belong to this class.
For the construction of the tree, given the training set $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the goal is always to minimize the training loss (3.1). Since g is in the form (3.5), we have that:

$$\ell_\tau(g) = \sum_{w \in \mathcal{W}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\mathbf{x}_i \in \mathcal{R}_w\} \mathscr{L}(y_i, g^w(\mathbf{x}_i)).$$

The key aspect in the loss minimization is the choice of the splitting rules. Taking an arbitrary training subset $\sigma \subseteq \tau$, a splitting rule divides it into two subdatasets σ_T (the one for which the proposition of the rule is true) and σ_F (the other one). Let y_T^* be the most prevalent class in σ_T and y_F^* be the most prevalent class in σ_F. Using the indicator loss and a constant regional prediction function, the best splitting rule minimizes

$$\frac{1}{n} \sum_{(\mathbf{x},y) \in \sigma_T} \mathbb{1}\{y \neq y_T^*\} + \frac{1}{n} \sum_{(\mathbf{x},y) \in \sigma_F} \mathbb{1}\{y \neq y_F^*\}. \tag{3.6}$$

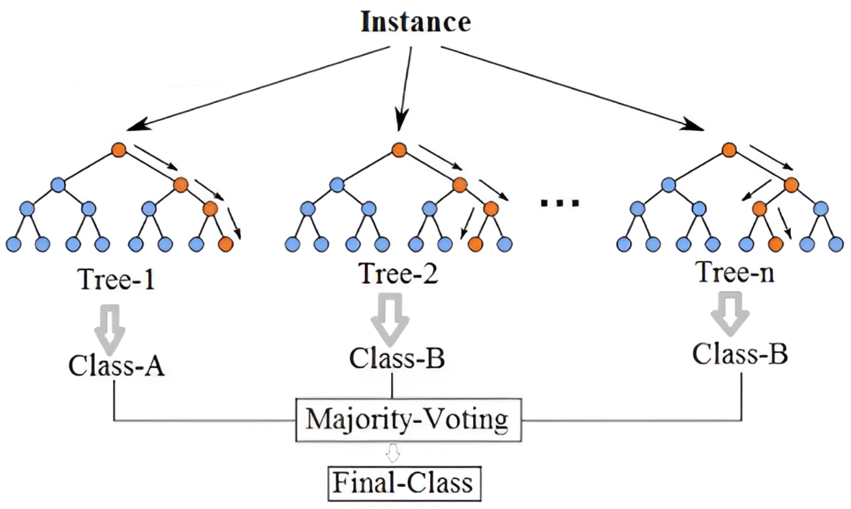


Figure 3.4. Scheme of a random forest [48].

The first split is performed on τ with the goal of minimizing (3.6), and new splits are carried out until a certain stopping criteria is satisfied, e.g. a threshold on the maximum depth of the tree.
The reason why (3.6) has to be minimized is the following: let us say that p_z is the proportion of data points in σ with label z. The quantity

$$p_\sigma = 1 - \max_{z\in\{0,\dots,c-1\}} p_z$$

measures the diversity of the labels in σ and is called the *misclassification impurity*. Consequently, (3.6) is the weighted sum of the misclassification impurities of σ_T and σ_F, with weights $|\sigma_T|/n$ and $|\sigma_F|/n$, respectively. Thus, the goal is to have rules which split σ less uniformly as possible: p_σ reaches its maximum value when every label has proportion $1/c$, its minimum, i.e. 0, when all samples in σ have the same label.
Instead of minimizing the misclassification impurity to decide how to split a dataset, it is possible to utilize other impurity measures that only depend on the label proportions, such as the Gini Impurity (GI) index:

$$\mathrm{GI} = \frac{1}{2}\left(1 - \sum_{z=0}^{c-1} s_z^2\right).$$

Decision trees serve as a building block for the Random Forest (RF) classifier, which consists of B trees, where the parameter B must be carefully chosen. Tipically, the larger this number, the better the performance, but if it is too high, the computational effort becomes really heavy. For this reason, this parameter is often chosen pre-training, by trying some values to find the best tradeoff between time needed and accuracy. However, each tree $b = 1, \dots, B$ is trained on a dataset $\mathcal{T}_b^*$, composed by a random sample of the original observations. Furthermore, every tree uses only a randomly selected subset of $\lfloor \sqrt{p} \rfloor$ features for the splitting rules, where p is the total number of features. For the inference phase, every tree will make a prediction, and then the label will be assigned via majority vote: the most frequent class will be selected. (see Figure 3.4).

Chapter 4

Fully Homomorphic Encryption for Machine Learning

FHE and ML can work together in a powerful way to enable secure computation on encrypted data. This collaboration is particularly important in scenarios where sensitive data needs to be analyzed without compromising privacy.
In this chapter, we are going to analyze the applications of some ML methods, particularly focusing on the advantages and disadvantages of the usage of FHE. First of all we are going to introduce the working framework in section 4.1, and after that we will present the solution of some scientific papers in this field. More in detail, the chapter is articulated as follows:

- In section 4.2 we present the solutions proposed by [49][50][51][52] for an efficient homomorphic KNN inference;
- In section 4.3 we describe how [53][54][55][56][57] integrate FHE and SVM;
- In section 4.4 we show the differences between the homorphic implementation of Decision Tree and Random Forest by [58] and [59].

4.1 Framework

The context we are going to analyze features two main protagonists:

- A client, who has some data and needs to perform some operations on it;
- A server or Service Provider (SP), which can carry out some complex tasks and is able to store a large quantity of data.

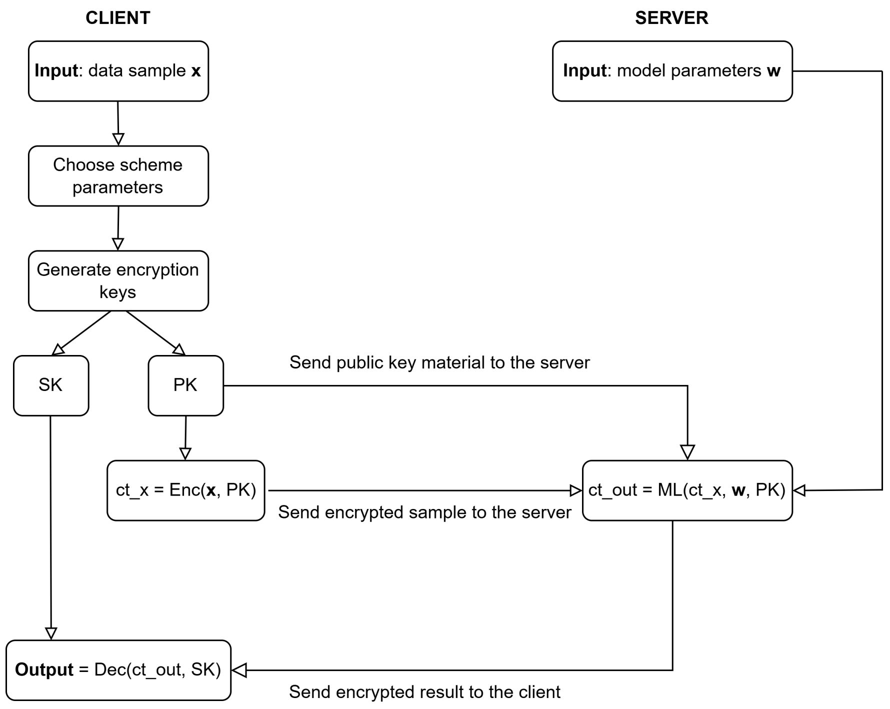

CLIENT
SERVER
Input: data sample **x**
Choose scheme parameters
Generate encryption keys
SK
PK
ct_x = Enc(**x**, PK)
Input: model parameters **w**
Send public key material to the server
ct_out = ML(ct_x, **w**, PK)
Send encrypted sample to the server
Output = Dec(ct_out, SK)
Send encrypted result to the client

Figure 4.1. Working idea of the client/server architecture. PK and SK stand for public and secret key, pt and ct for plaintext and ciphertext respectively

These two parties can interact one with each other in different ways. For instance, the client can use the SP just to store his data, but can also ask him to perform some computations on it. In the latter case, a paramount concern is how to make SP do that without revealing the client's data, and this is where FHE comes to play. In particular, we are going to study how SP can classify encrypted data using some ML tools. The standard process is the following:

1. The SP trains a public ML model on some public data. After the training phase, the hyperparameters of the model are kept secret, therefore only the SP knows them;
2. The client sends to SP a query $\mathsf{Enc}(x)$ encrypted with a FHE scheme;
3. The SP performs encrypted computations, applying a certain function f to the ciphertext, thus obtaining $f(\mathsf{Enc}(x))$;
4. The SP sends back the encrypted response to the client, which decrypts it to get the result $\tilde{f}(x)$.

The whole process is summarized in Figure 4.1. In some applications, the third point is not entirely up to SP, depending on the type of computations to be done. Furthermore, some studies also developed a private training phase, where even the training data is encrypted. This is possible, but it drastically impacts the overall computational overhead. Finally, some tasks can be very challenging to address while operating homomorphically, e.g. comparison or conditional operations. For this reason, sometimes SP does most of the job, but the last part of inference is left to the client.

Difficulties

Mathematical structures and cryptographic primitives of FHE are already computationally intensive by themselves. When applied to ML, the situation gets worse, and can become unsubstainable depending on the ML algorithm or on the hardness of classification (for example, this is the case while dealing with images). For instance, a simple division in plaintext might correspond to several underlying cryptographic operations in the encrypted domain.

For this reason, at its first lights, the application of FHE to ML was not possible at all: the homomorphic algorithms, for example [6], performed an encrypted operation about 1 billion times slower than the time needed to do it on plaintext. Year after year, the gap has been progressively reduced: FHE algorithms can still be 4 orders of magnitudes slower than standard encryption schemes, but applications are possible. This enhancement is due to some new brand solutions developed in the last years, e.g. hardware acceleration. One example in this sense is the work presented by [60], where they introduce the first programmable accelerator. This product is a 10 nm unit, which is tailored to complex operations on long vectors and challenging data movements, i.e. the main characteristics of FHE. It demonstrates sensible speedups over state-of-the-art software implementations: this idea and other methods, such as [61] and [62], paved the way to new FHE applications.

Storage problems must also be taken into account: FHE ciphertexts are often much larger than their plaintext counterparts. The size of the ciphertexts grows significantly due to the need to encapsulate enough information to support multiple homomorphic operations without losing the ability to decrypt correctly: the larger the ciphertext, the higher the memory demand and increased computational overhead. Furthermore, in order to achieve a high level of security, FHE schemes use large parameters (e.g., large key sizes), which make each cryptographic operation even more expensive in terms of computation and memory. More specifically, in the client-server framework, if every clients send a massive quantity of data to the server, the cost of whole process becomes unbearable.
This issue can be addressed by Hybrid Homorphic Encryption (HHE). The main idea behind HHE is the following: instead of encrypting the data with HE schemes, encrypt the data with a symmetric cipher and send the ciphertexts to the server. The server then first homomorphically performs the symmetric decryption circuit to transform the symmetric ciphertext into a homomorphic ciphertext and then proceeds with performing the actual computations. This procedure trades bandwidth requirements with a more expensive computation on the server and requires that the data holder first sends the symmetric key encrypted with the HE scheme [63] [64].

Noise also plays a significant role, because FHE schemes introduce it into the ciphertexts to ensure security. Every homomorphic operation (addition or multiplication) increases this noise, whose management requires additional computational resources. Bootstrapping is usually needed to control the growth of the error of the encrypted data, but it is computationally intensive and significantly contributes to the overall cost. However, if the noise grows too large, it can lead to incorrect decryption, affecting the results of the ML algorithm.

Accuracy of methods combining FHE and ML can be reduced compared to standard ML algorithms, because of many reasons. First of all, FHE schemes often operate on fixed-point numbers with limited precision or on integers by quantization, which can lead to loss of accuracy in computations that require floating-point arithmetic. This is because quantization can disrupt the algebraic structures supported by FHE by introducing non-linearities and discontinuities in the data representation. This incompatibility can worsen the performance of homomorphic operations, as the schemes may not handle the quantized values as efficiently or accurately as they do with the unquantized ones. Plus, certain mathematical operations, such as division or non-linear functions, are not directly supported by FHE and must be approximated. These approximations can introduce errors that degrade the performance of ML models. It is worth underlining that this loss in accuracy is due to implementative reasons, because conceptually there is no theoretical motivation. Indeed, mathematical-wise a standard ML algorithm should be as accurate as a FHE - ML method.

Model complexity is also a major concern: in order to make algorithms compatible with FHE, some methods might need to be simplified. For instance, deep neural networks with many layers might be replaced with shallower networks, leading to reduced accuracy. In general, FHE-friendly models often avoid or approximate operations that are hard to implement efficiently with FHE.

Finally, quantization processes can lead to loss of information and worse performance. These methods essentially consist in the conversion of continuous features into discrete values to fit the integer-based arithmetic of FHE. Unfortunately, they can introduce rounding errors in the ciphertexts, which accumulate during the multiple homomorphic operations, leading to significant deviations from the true result.
Then, the quantization process can affect the noise in the ciphertexts. When values are quantized, the small errors introduced by rounding are magnified through homomorphic operations, potentially causing the noise to grow faster than it would without quantization. This can result in the ciphertext becoming too noisy to decrypt correctly after a certain number of operations, thus limiting the depth of feasible computations.
Furthermore, quantization limits the range of representable values, potentially causing overflow or underflow issues during computations. This constraint can restrict the types of operations that can be performed homomorphically and reduce the overall functionality of the scheme.

For all these reasons, it is clear that all the presented ML algorithms need some adjustments when data necessitates protection from a FHE scheme. We noticed that TFHE [25] and CKKS [26] are the mostly employed schemes when coping with ML, so we are going to focus on these two. In general, in the following sections we are going to analyze some issues that may arise in the transition to the encrypted domain homomorphic context, and present some of the proposed solutions. We will study both encrypted and unencrypted algorithms, trying to understand in which contexts it is worth to lose time and accuracy in order to gain privacy and security.

Algorithm 1 SquaredEuclideanDistance(X,Y)

```
Input: ciphertext arrays $X = (x_1, \dots, x_n), Y = (y_1, \dots, y_n)$
Output: ciphertext $c_d$
  $D \leftarrow zeros(n)$
  for $i = 1, \dots, n$ do
      $s \leftarrow Add(X[i], -Y[i])$
      $D[i] \leftarrow Mult(s, s)$
  end for
  $c_d = D[1]$
  for $i = 2, dots, n$ do
      $c_d \leftarrow Add(c_d, D[i])$
  end for
```

We want to precise that the following mentioned papers have been preferred with respect to others on the same topics because of completeness and correctness. In the research process, we actually found a high number of papers on these topics, but we chose only the most recent ones.

4.2 Homomorphic KNN

When dealing with homomorphic inference of a KNN algorithm, the three most problematic aspects are distances computation, distances sorting and majority vote.

One of the main obstacles to face when calculating distances is the type of metric to use. KNN offers a few metrics to operate (such as those listed in subsection 3.2.1), but some of them are more suitable to FHE than others. More specifically, every distance measure featuring a division needs additional effort to approximate it. For this reason, many studies [49][50][51][52] avoid this complication by using the squared Euclidean metric only. This metric features just sums and multiplications, which are far easier to be computed homomorphically.

Given a training set τ, for a new sample $\mathbf{q}$ and a point $\mathbf{x}_i \in \tau$, both belonging to $\mathbb{R}^p$, let us define $c_q = \mathsf{Enc}(\mathbf{q})$ and $c_i = \mathsf{Enc}(\mathbf{x}_i)$. The squared distance in the plaintext domain is

$$d_i := d(\mathbf{x}_i, \mathbf{q}) = \sum_{j=1}^{p} (x_{ij} - q_j)^2. \tag{4.1}$$

In [49], this formula is homomorphically computed by Algorithm 1, where *Add* and *Mult* are to be intended as the homomorphic sum and multiplication.

After having computed all the distances, a major concern is how to put them in ascending order, such that we can select the K closest points to $\mathbf{q}$ to make inference. In order to do that, we need to compare distances one by one, thus we have to use a comparison method. Cheon et al. [50] presented a solution based on polynomial composition which is based on the following: let the comparison function $\mathrm{comp}(a, b)$ and the sign function sign(x) be defined as

$$\text{comp}(a,b) = \begin{cases} 1 & \text{if } a > b \\ \frac{1}{2} & \text{if } a = b \\ 0 & \text{if } a < b \end{cases}, \qquad \text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}.$$

It holds that

$$\text{comp}(a,b) = \frac{\text{sign}(a-b)+1}{2}.$$

The idea is that we can compare two numbers via this relation, approximating the sign function with a polynomial $p(x) = f_n^r(x) \circ g_n^s(x)$, where the powers stand for the r or s-times composition, whereas f_n and g_n are polynomials of degree n. The authors have proved that $p(x)$ converges to sign(x) if and only if r and s are high enough, and if the two polynomials satisfy some properties. Namely:

- $f_n(-x) = -f_n(x) \quad \forall x;$
- $f_n(1) = 1, \ f_n(-1) = -1;$
- $f_n(x)$ is convex in $[-1{,}0]$ and concave in $[0{,}1]$,

and

- $g_n(-x) = -g_n(x) \quad \forall x;$
- $\exists \gamma \in (0{,}1) : x < g_n(x) \leq 1 \quad \forall x \in (0,\gamma);$
- $\exists \tau \in (0{,}1) : g_n([\gamma{,}1]) \subseteq [1-\tau{,}1].$

The authors of the paper show that there exists a unique polynomial $f_n(x)$ meeting all the three needed requirements, whereas $g_n(x)$ is not unique, but the optimal choice is the one that minimizes γ.

Polynomial approximation is one of the possible ways to address the problem, but there are other alternatives. Indeed, the approach described in [51] is completely different. In this paper, the authors take into considerations the differences between all pairs of possible distances, calculated by (4.1), which can be written as follows:

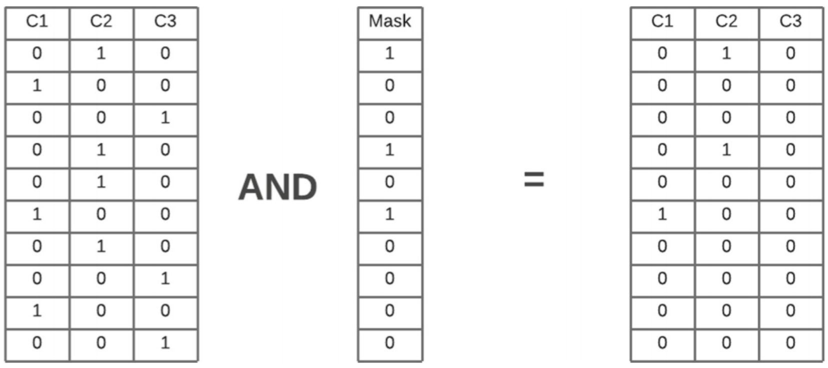
$$d_i - d_{i'} = \sum_{j=1}^{p}(x_{ij}^2 - x_{i'j}^2) - 2\sum_{j=1}^{p} q_j(x_{i'j} - x_{ij}).$$

In the encrypted domain, once we compute this quantity by composing the necessary homomorphic operations, it is possible to define

$$\delta_{ij} = \begin{cases} 1 & \text{if comp}(d_j, d_i) = 1 \\ 0 & \text{otherwise} \end{cases}.$$

We can build a matrix $\delta \in \mathbb{R}^{n\times n}$ containing all these values; by summing its rows, we get the vector

$$(\delta_{1\cdot}, \dots, \delta_{n\cdot}), \ \text{with } \delta_{i\cdot} = \sum_{j=1}^{n} \delta_{ij} \in [0, n-1].$$



C1	C2	C3
0	1	0
1	0	0
0	0	1
0	1	0
0	1	0
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1

AND

Mask
1
0
0
1
0
1
0
0
0
0

=

C1	C2	C3
0	1	0
0	0	0
0	0	0
0	1	0
0	0	0
1	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Figure 4.2. Encrypted majority vote [52]

The distance for which every comparison to another distance yields a 0 will correspond to $\delta_{i\cdot} = 0$, i.e. the greatest distance. Respectively, the smallest distance has an associated $\delta_{i\cdot}$ equal to $n - 1$. Thus, this vector will give us the ordered distances, which will allow us to choose the K nearest neighbors for the new point $\mathbf{q}$.
Unfortunately, the method has a limitation: in an homomorphic setting we cannot add an unlimited amount of ciphertexts together, because the computational cost of the bootstrapping procedure can become unbearable in high dimensions.
It has to be mentioned that this idea has been implemented on the TFHE scheme, so in order to make it suitable for that, every measure has to be in a certain range. For example, [52] exploited this procedure, rescaling the distances in $[-\frac{1}{2}, \frac{1}{2}]$.

Finally, majority vote can cause some troubles because it is performed by comparisons, thus it needs conditional operations to be carried out. Unluckily, there is no straightforward homomorphic-friendly method to deal with "if" conditions, but there are some ways to get around the obstacle.

The simplest way is to leave the last part of inference to the client. This means that the user receives from the server the labels of the K points in the neighboorhood of $\mathbf{q}$, he decrypts this information and then chooses the most common class. This method is the one chosen by [49], which also uses the above-descripted polynomial approximation to sort distances. This approach results to be the fastest, but of course it sacrifices privacy, because the client becomes aware of which points compose the neighborhood of $\mathbf{q}$. For this reason, [52] proceedes by making the server carry out the majority voting process in an encrypted manner. Initially, the client encodes the labels using one-hot encoding. This means that each sample is associated to a vector with l entries, where l is the number of possible labels. The labels are ordered in some way, and each vector will have 1 in the position corresponding to the label of that sample, 0 everywhere else.
After that, the SP does its computations to establish the K closest points to $\mathbf{q}$, and then defines a vector which serves as a mask to indicate which samples compose the neighborhood. With both the mask and the one-hot encoded label matrix available, an AND operation can be performed between the mask and each column of the label matrix,

Reference	Scheme	Problem	Solution
[49]	CKKS	Distances computation	Homomorphic Euclidean distance
[50]	CKKS	Distances comparison	Polynomial approximation
[51]	TFHE	Distances sorting	Delta matrix
[52]	TFHE	Private majority vote	Encrypted procedure

Table 4.1. Summary of Homomorphic KNN papers

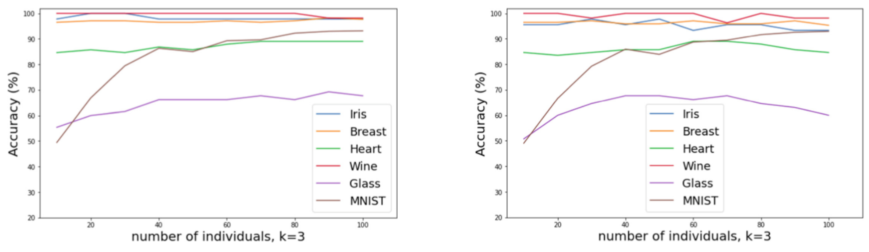


Figure 4.3. Accuracy of unencrypted vs encrypted KNN [52]

as illustrated in Figure 4.2. This results in a matrix A, where A_{ij} equals 1 if individual i is among the k-nearest neighbors and belongs to class j. By summing the columns of this matrix, the frequency of each class can be determined. The process allows for the return of only a vector with these frequencies, ensuring no information leakage about which points belong to the neighboorhood of $\mathbf{q}$.

To conlcude, Table 4.1 summarizes all the presented issues with the relative solution.

Performances

Regarding the performance of the method, these studies enlight that the number of samples heavily affects the execution time. In particular, [49] shows that increasing the size of the dataset, the time consumed in the distance sorting phase becomes very high and dominant. More in detail, 9 minutes are required with 8 samples and 11 hours with 17 samples (see Table 4.2 for more details). On the other hand, the parameter K does not seem to have a considerable influence. For what concern accuracy, [52] tests the algorithm on different known datasets, such as Iris [65], Breast Cancer [66], Heart [67], Wine [68], Glass [69], and MNIST [70], finding only a small performance drop, as can be spotted in Figure 4.3 (probably because these data are quite simple and very well preprocessed). A big difference in memory usage has also to be reported: each ciphertext encrypting one value of the dataset in [49] occupies approximately 51 MB. Considering a scenario with 16 samples, the total size would be 10.3 GB; in contrast, the original dataset in .csv format is only 649 bytes.

Samples	Encryption	Distances	Comparisons	Sorting	Total time
8	0 : 00 : 49	0 : 02 : 06	0 : 14 : 57	0 : 08 : 58	0 : 26 : 50
9	0 : 00 : 55	0 : 02 : 39	0 : 22 : 27	0 : 17 : 38	0 : 43 : 39
10	0 : 01 : 01	0 : 03 : 10	0 : 28 : 12	0 : 27 : 24	0 : 59 : 48
11	0 : 01 : 10	0 : 03 : 44	0 : 40 : 10	0 : 46 : 54	1 : 31 : 59
12	0 : 01 : 14	0 : 04 : 20	0 : 50 : 45	1 : 12 : 35	2 : 08 : 56
13	0 : 01 : 18	0 : 04 : 49	1 : 02 : 23	1 : 51 : 31	3 : 00 : 01
14	0 : 01 : 28	0 : 05 : 43	1 : 19 : 31	2 : 52 : 25	4 : 19 : 07
15	0 : 01 : 39	0 : 05 : 51	1 : 38 : 49	4 : 16 : 46	6 : 03 : 05
16	0 : 01 : 42	0 : 06 : 41	1 : 59 : 44	6 : 19 : 01	8 : 27 : 08
17	0 : 01 : 49	0 : 07 : 58	2 : 33 : 37	10 : 51 : 28	13 : 34 : 52

Table 4.2. Execution time for KNN homomorphic inference featuring polynomial approximation for comparison and sorting (time is measured in HH:MM:SS) [49]

4.3 Homomorphic SVM

This ML algorithm does not include any particular difficulties in the inference phase, except for the evaluation of the sign function. In all the analyzed studies [53][54][55][56][57] this part is left to the user, which receives $\mathsf{Enc}(g_\mathcal{T}(\mathbf{x}))$ from the server, and computes the sign of this expression in the plaintext domain. Essentially, the only operation the SP has to do is the evaluation of the objective function, by using the optimal parameters coming from the unencrypted training phase.
At this stage, an important question is how to encrypt the feature vector. The most intuitive solution is to encrypt each component of the feature vector in a separate ciphertext element, but this would impose very large computation requirements. For instance, if the client needs to encrypt and communicate m ciphertexts to the SP, the latter necessitates m homomorphic multiplications and $m-1$ homomorphic additions to compute the dot product in the decision function.

Luckily, the CKKS scheme offers a smarter technique, thanks to Smart and Vercauteren [54]. Their method allows one to encode multiple messages in one plaintext element that can be encrypted to generate only one packed ciphertext element. More concretely, an array of up to $t = n/2$ complex numbers, where n is a parameter of the scheme, can be encoded as one plaintext element. One may view the plaintext or ciphertext elements as a container with a fixed number of slots. In each slot, one input message (a numeric value) can be stored. Homomorphic addition or multiplication of two packed ciphertexts, say $a = \mathsf{Enc}(u_1, \dots, u_t)$, and $b = \mathsf{Enc}(v_1, \dots, v_t)$ results in component-wise homomorphic addition or multiplication. In order to work, the size of feature vectors must not be more than t.

This method comes into play while computing $g_\mathcal{T}(\mathbf{x})$ in the encrypted domain: the basic procedure is to pack each SV $\mathbf{x}_i$ in one plaintext element. Afterwards, $\tilde{\mathbf{x}_i} = \mathsf{Enc}(\mathbf{x}_i)$ and $\tilde{\mathbf{x}} = \mathsf{Enc}(\mathbf{x})$ are multiplied element-wise, and finally the components of the resulting

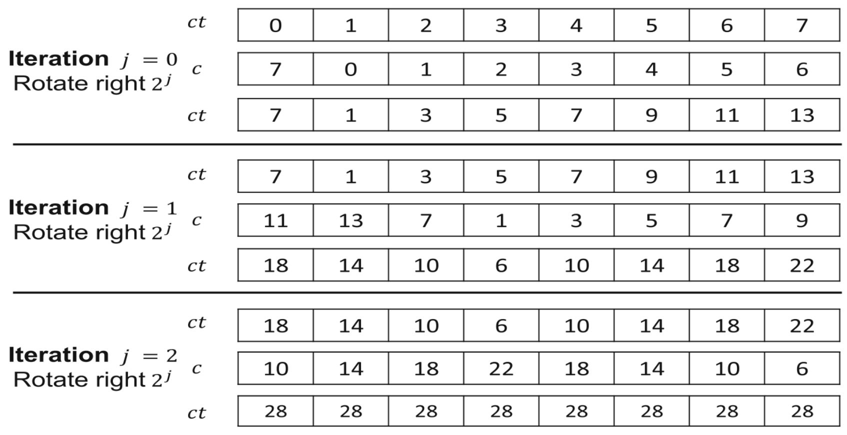


Figure 4.4. Sum of the components of a vector [53]

vector are summed up to get the dot product in (3.3). These last procedure is called *total sum*, and it is well explained by Figure 4.4: at the beginning, the ciphered vector ct is summed to a vector c, which contains the same elements as ct, right-shifted of 1 position. The same process is then repeated, but the vector c is obtained performing a shift of 2^j positions, where j identifies the iteration number. At the end, we will get a vector whose components are all equal to the desired sum.

In fact, [53] also developed an ultra-packed plaintext/ciphertext strategy for LSVM, capable of packing multiple support vectors in one plaintext element. Firstly, they compute $\eta = 2^{\lceil \log_2 p \rceil}$ and create a vector that contains $N = \frac{t}{\eta}$ support vectors. In this vector, support vector $\mathbf{x}_i$ is stored at index $i\eta$ for $i = 0, \dots, N-1$, and any unpopulated component is filled with zeros. The client is also required to create a vector of linearly packed η clones of $\mathbf{x}$ similar to the way $\mathbf{x}_i$ is packed. The vector will be encrypted and sent to the server for homomorphic evaluation of LSVM prediction. By doing so, it is possible to compute η addends of the sum in (3.3) at a time, speeding up the whole process. This process is carried out by employing the *partial sum* algorithm, as shown in Figure 4.5. In the image, each feature vector $\mathbf{x}_i$ belongs to $\mathbb{R}^3$, resulting in $\eta = 4$, thus every vector is padded by adding a zero at the end. Then, the vectors are concatenated to form ct, and after these preliminary operations, c is built and summed to ct analogously as in *total sum*. At the end, we will have the sum of the components of vector $\mathbf{x}_i$ in position $(i\eta - 1)$.

Some studies have also explored the possibility of performing training in the ciphertext domain. For example, [55] decided to use SVM for the so-called *fair learning* task. This basically consists in establishing the influence of some sensitive features on classification, determining the *fairness* of the model. [56] defines how this characteristic can be measured: each sample has an associated sensitive feature $z \in \{0, 1\}$ (e.g. gender). In this case, fairness is expressed as the absence of *disparate impact*.

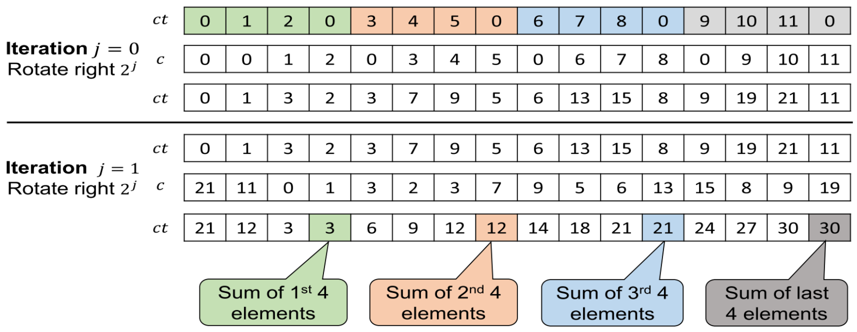

Iteration $j = 0$
Rotate right 2^j
ct 0 1 2 0 3 4 5 0 6 7 8 0 9 10 11 0
c 0 0 1 2 0 3 4 5 0 6 7 8 0 9 10 11
ct 0 1 3 2 3 7 9 5 6 13 15 8 9 19 21 11
Iteration $j = 1$
Rotate right 2^j
ct 0 1 3 2 3 7 9 5 6 13 15 8 9 19 21 11
c 21 11 0 1 3 2 3 7 9 5 6 13 15 8 9 19
ct 21 12 3 3 6 9 12 12 14 18 21 21 24 27 30 30
Sum of 1st 4 elements
Sum of 2nd 4 elements
Sum of 3rd 4 elements
Sum of last 4 elements

Figure 4.5. Sum of the components of multiple vectors ($p = 3, \eta = 4$) [53]

Definition 4.3.1 (Disparate impact) *A binary classifier does not suffer from disparate impact if the probability that a classifier assigns a user to the positive class, $\hat{y} = 1$, is the same for both values of the sensitive feature z, i.e.,*

$$\mathbb{P}[\hat{y} = 1|z = 0] = \mathbb{P}[\hat{y} = 1|z = 1].$$

The framework in [57] enables to measure the disparate impact and to train a fair LSSVM classifer by forcing a covariance-based constraint. Namely, the covariance measure for a sensitive feature z and a decision function $f(\mathbf{x})$ for a dataset τ is:

$$\text{Cov}_\tau(z, f(\mathbf{x})) = \frac{1}{|\tau|} \sum_{(z,\mathbf{x})\in\tau} (z - \overline{z}) f(\mathbf{x}).$$

where $\overline{z}$ denotes the average of z. The fairness constraint is simply $|\text{Cov}_\tau(z, f(\mathbf{x}))| \leq c$. In this context, the LSSVM minimization problem (3.2) has to be modified. From now on, we will be writing $\mathbf{x}$ to not overcomplicate the notation, but we remind that all this procedure has to be carried out in the encrypted domain, so it is actually run on $\mathsf{Enc}(\mathbf{x})$. Given the column vectors $\mathbf{z}_j \in \mathbb{R}^n$ for $j = 1, \dots, s$, where s is the number of sensitive features, we can define the matrix $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_j] \in \mathbb{R}^{n\times s}$, and solve:

$$\begin{aligned} \min_{\theta,\mathbf{e},\mathbf{d}} \ & \frac{1}{2}\sum_{k=1}^{p} \theta_k^2 + \frac{C_1}{2}\sum_{i=1}^{n} e_i^2 + \frac{C_2}{2}\sum_{j=1}^{s}(nd_j)^2 \\ \text{s.t. } & y_i \theta^T \phi(\mathbf{x}_i) = 1 - e_i, \quad i = 1, \dots, n \\ & d_j = \frac{1}{n}\sum_{i=1}^{n}(z_{ij} - \overline{z}_j)\theta^T\phi(\mathbf{x}_i), \quad j = 1, \dots, s, \end{aligned}$$

with $\overline{z}_j = \sum_{i=1}^{n} z_{ij}$ and $C_1, C_2 > 0$.
We can notice some differences with respect to (3.2): the third sum in the objective function and the new constraint are needed to regularize the magnitude of disparate

Reference	Scheme	Problem	Solution
[53]	CKKS	Ciphertext management	Ultra packing technique
[54]	CKKS	Encode multiple messages	Standard packing technique
[55]	CKKS	Fair learning	Reformulation of SVM problem
[56]	CKKS	Fairness formalization	Disparate impact
[57]	CKKS	Fairness management	Covariance measure

Table 4.3. Summary of Homomorphic SVM papers

impact, in order to satisfy the covariance constraint. This optimization problem is solved by employing the Lagrangian function with multipliers $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}^s$:

$$\mathcal{L}(\theta, \mathbf{e}, \mathbf{d}; \alpha, \beta) = \frac{1}{2}||\theta||^2 + \frac{C_1}{2}||\mathbf{e}||^2 + \frac{C_2}{2}||n\mathbf{d}||^2 + \\ + \sum_{i=1}^{n} \alpha_i (y_i - \theta^T \Phi(\mathbf{x}_i) - e_i) + \sum_{j=1}^{s} \beta_j (\tilde{\mathbf{z}}_j^T \mathbf{\Phi} \theta - n d_j),$$

where $\mathbf{\Phi} = [\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_n)]^T$ and $\tilde{\mathbf{z}}_j = \mathbf{z}_j - \bar{z}_j \mathbb{1}_n$. Using the optimality conditions of Theorem 3.2.1 we get the linear system

$$M \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \mathbf{K} + \frac{1}{C_1}\mathbf{I}_n & -\mathbf{K}\tilde{\mathbf{Z}} \\ -\tilde{\mathbf{Z}}^T\mathbf{K} & \tilde{\mathbf{Z}}^T\mathbf{K}\tilde{\mathbf{Z}} + \frac{1}{C_2}\mathbf{I}_s \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}, \tag{4.2}$$

with $\tilde{\mathbf{Z}} = [\tilde{\mathbf{z}}_1, \dots, \tilde{\mathbf{z}}_s]$ and $\mathbf{K}_{ij} = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) \in \mathbb{R}^{n \times n}$. At this point, we have to solve this linear system in the homomorphic domain. Lagrange multipliers theory ensures existence and uniqueness of the solution, thus a possible way to proceed might be to invert the matrix M utilizing the gradient descend method [71], but the computational effort would be too high. Since we are in the encrypted domain, we want to simplify the system as much as possible to solve it in a reasonable time. Indeed, matrices multiplications require a huge number of sums and multiplications to be performed if the involved factors are large, sensibly increasing the overall computational overhead. For this reason, the authors of this paper explored some techniques to reduce the number of equations in (4.2).

One of the possibilities is the employment of the Schur complement [72], which is essentially a reformulation of the inverse matrix. In the process, inverse matrices are computed by using Goldschmidt's division algorithm [73] if $s = 1$, Newton-Schulz iterative algorithm [74] if $s > 1$.

Although this algorithm significantly reduces the number of matrix multiplications, the computational cost can still pose a challenge when n is very large. To address scalability, the authors introduce a low-rank approximation technique for the $n \times n$ matrices. Using eigendecomposition, we express $\mathbf{K} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, where $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n] \in \mathbb{R}^{n \times n}$ is an orthogonal matrix composed of the eigenvectors of $\mathbf{K}$, and $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$.
The key is approximating $\mathbf{K}$ using the its q largest eigenvalues, selecting the corresponding p eigenvectors. This allows to replace the multiplications between $n \times n$ matrices and the centered sensitive attribute matrix $\tilde{\mathbf{Z}} \in \mathbb{R}^{n \times s}$ with $\mathbf{V}_{(p)}^T \tilde{\mathbf{Z}} \in \mathbb{R}^{p \times s}$, where

Threads	Latency	Speed up
1	29.89	23.01
2	15.30	44.95
4	8.14	84.49
8	4.33	158.83
16	2.33	295.17
27	1.64	424.01
32	1.51	419.36
52	1.25	550.20

Table 4.4. Performance analysis of Homomorphic SVM with ultra-packing in terms of number of threads versus average prediction latency [53]

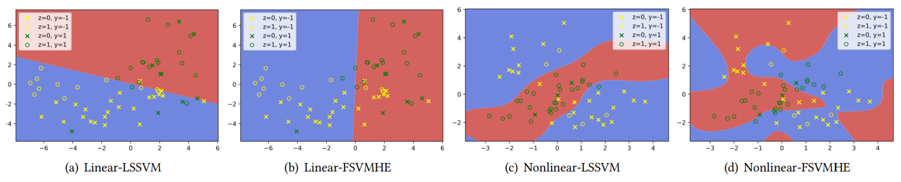


Figure 4.6. SVM output for different datasets [55]

$\mathbf{V}_{(p)} = [\mathbf{v}_1, \dots, \mathbf{v}_p] \in \mathbb{R}^{n \times p}$. This approximation can significantly improve the algorithm's efficiency when $p \ll n$, with a minimal impact on accuracy.
Table 4.3 gives an overview of all the aspects we covered in this paragraph.

Performances

[53] did some tests to measure the impact of their ultra-packing technique on the method, trying also to introduce parallelization to achieve an even better result. The standard method needs 688 seconds to execute, but parallelization improves the latency by almost one order of magnitude when the number of threads is 8 or above. Plus, when applying the aforementioned technique with the maximum number of threads, the total time is only 1.25 seconds, making the method suitable for real-time predictions (more details reported in Table 4.4). In addition, the authors declare a zero loss in the prediction accuracy with respect to the unencrypted inference when the system is tested over all the examples in the testing dataset.
[55] implemented the method with linear kernel and RBF kernel with for two datasets, Linear and Nonlinear. In Figure 4.6 there is a plot of the classification results with fairness constraint (FSVMHE) and without it (LSSVM). In the image, the color of a point represents its label, the shape depends on the sensitive feature z. Hence, an accurate classifier should separate yellow points from dark ones, whereas a good fairness is achieved if the number of crosses and circles on the same side is approximately the same.
We can notice that LSSVM classifes more $z = 0$ samples as negative. For the first

dataset, to make the model fair, FSVMHE changes the decision boundary to classify more $z = 0$ samples as positive. Similarly for Nonlinear dataset, FSVMHE creates a more complex decision boundary to prevent more $z = 0$ samples from being classifed as negative. Furthermore, the experimental results verifed that FSVMHE could attain good inference performance even if the low-rank approximation was applied during training.

4.4 Homomorphic Decision Tree and Random Forest

The tree-based algorithms share the problem of traversing the binary tree in a homomorphic way. This issue was addressed, for example, by [58] employing the TFHE scheme, and by [59] utilizing the CKKS scheme.
The first solution is based on quantization, which rescales the values x_{ij} of every column of the training matrix $\mathbf{x}_j = (x_{1j}, \ldots, x_{nj})$, containing the j-th feature of all n training samples. Given the number of bits b that will be used to represent the quantized values, this is done by the formula

$$q(x_{ij}) = \left\lfloor \frac{x_{ij}}{\Delta} + \mu \right\rceil ,$$

where

$$\Delta = \frac{\max(\mathbf{x}_j) - \min(\mathbf{x}_j)}{2^b - 1}$$

is the step size and μ is the center of the quantized distribution of values. Because of the definition of the step size, we have that $q(x) \in [-2^{b-1}, 2^{b-1} - 1]$ if $\mu = 0$. This would be the case of *symmetric quantization*, but for the application we are analyzing *asymmetric quantization* is preferred, thus μ is chosen such that $q(\min(\mathbf{x}_j)) = 0$. The latter is better for its greater precision, and since a tree-based model does not perform linear combinations of the inputs, every single feature can be quantized independently of each other. This allows to have a scale and a zero point (the minimum assumed by the unquantized values) for each feature, which is a great advantage when the input dimensions have different orders of magnitude and follow different distributions. Proceeding like this, the input space is fully quantized, i.e. made of integers only. This means that the tree-based model can be trained on this new input space, resulting in quantized decision rules. In practice, when a splitting rule $\mathbb{1}\{x_{ij} < k\}$ is chosen, k is rounded to the closest integer and the inequality is converted into a lower or equal comparison, with the purpose to have an integer-only problem. Finally, each terminal leaf value is properly quantized.

When dealing with splitting rules, we face conditional operations, which are not directly feasible in FHE. To work around this limitation, the lookup table (LUT) operation, which is currently a unique feature of TFHE, is a valid solution. For example, consider a two-dimensional integer input space where each data point $\mathbf{x}_i$ belongs to the set $[0,2^b) \times [0,2^b)$, where b is the number of bits used to encode the features of $\mathbf{x}_i$. Let the first feature of $\mathbf{x}_i$ be represented as x_{i1} and the second feature as x_{i2}, and assume $b = 3$. The simple condition $x_{i2} > 3$ can be expressed by a function:

$$f(\mathbf{x}_i) = \begin{cases} 0 & \text{if } x_{i2} > 3 \\ 1 & \text{otherwise.} \end{cases}$$

Algorithm 2 Tensorial tree scoring

Input: Input features to internal nodes X, tensors A, B, C, D, E
Output: Prediction matrix T

```
  $P \leftarrow X \cdot A$
  $Q \leftarrow P < B$
  $R \leftarrow Q \cdot C$
  $S \leftarrow R == D$
  $T \leftarrow S \cdot E$
```

Such a function includes an "if", which is not FHE-compatible, but if we take the array T = [1,1,1,1,0,0,0,0] as a table lookup, it yields $f(\mathbf{x}_i) = T[x_{i2}]$. In particular, the TFHE scheme allows to transform an input ciphertext (encrypting some value $\mathbf{x}$) into a ciphertext encrypting $f(\mathbf{x})$, by using LUT and bootstrapping. This whole process is called *programmable bootstrapping* [75] (the same introduced in section 2.1).

After this procedure, tree traversal still needs to be done. Unfortunately, it is not directly possible in FHE, as control-flow operations are not supported. Selecting which branch to run from the encrypted data is thus impossible. To overcome this, every branch has to be computed simultaneously by converting the tree traversal into tensor operations. This is done by Algorithm 2 employing the integer matrices A, B, C, D and E, which are derived from the training process. More in detail:

- A encodes the relationships between input features and internal nodes.
- B represents internal node values.
- C captures the relationship between internal nodes and the left or right sub-trees.
- D tracks the count of left child nodes from leaf to root in the decision tree.
- E maps the leaf nodes to produce the final prediction.

This algorithm consists of five main steps:

1. *Input Path Tensor Creation*: the input tensor X is multiplied by a tensor A, resulting in the input path tensor P.
2. *Comparison with Internal Node Values*: the resulting tensor P is compared to tensor B. This comparison assigns a binary value to each element of P, indicating whether the corresponding internal node is satisfied, producing the boolean tensor Q.
3. *Output Path Tensor Creation*: the tensor Q is then multiplied by tensor C, producing the output path tensor R.
4. *Path Comparison*: the tensor R, representing the output path, is compared with tensor D. This comparison generates tensor S, indicating the matching paths.
5. *Prediction Generation*: finally, the matching paths tensor S is multiplied by tensor E that maps the leaf nodes to produce the final prediction.

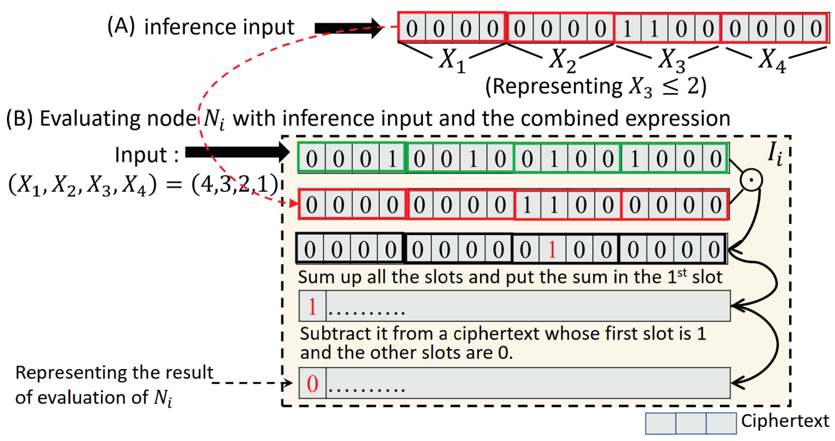

(A) inference input
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
X_1 X_2 X_3 X_4
(Representing $X_3 \leq 2$)
(B) Evaluating node N_i with inference input and the combined expression
Input :
$(X_1, X_2, X_3, X_4) = (4,3,2,1)$
0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0
I_i
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
Sum up all the slots and put the sum in the 1st slot
1 ....
Subtract it from a ciphertext whose first slot is 1 and the other slots are 0.
Representing the result of evaluation of N_i
0 ....
Ciphertext

Figure 4.7. Evaluation of a node during inference, with $n_{max} = 4, p = 4, t = 3$ [59]

On the other hand, [59] developed a completely different solution: they tried to use the CKKS scheme to perform encrypted inference on integer data, supposing that each feature $j = 1, \ldots, p$ can assume only values in $\{1, \ldots, n_j\}$, for some $n_j \in \mathbb{N}$. Let us define $n_{max} = \max\{n_1, \ldots, n_p\}$ and let $y = \{0, \ldots, c-1\}$ be the set of possible labels. All variables are represented using one-hot encoding, and the length of the vector representation for each variable is n_{max}. This allows for easy expansion by adding zeros at the end of features with $n_j < n_{max}$. Such a representation expresses a sample as a vector with $p \cdot n_{max}$ elements. To enhance efficiency, data is processed column-wise: in plaintext, a feature j of a new sample $\mathbf{x} = (x_1, \ldots, x_{n_j})$ would be represented as $(x_j^1, \ldots, x_j^{n_{max}})$, where x_j^h stands for the h-th bits across the one-hot encoded value of feature j. Once encrypted, x_j^h is stored in w ciphertexts, represented as $c_j^{h,1}, \ldots, c_j^{h,w}$. It is worth underlining that the application of one-hot encoding implies that there is no decision rule of the form $x_{ij} \in S$ for a categorical feature j. This is because with this encoding, feature j is replaced by n_j features, say $j^1, \ldots, j^{n_j}$, each one representing one of the possible values of the initial feature. All these new features will assume only 0 and 1 values, so the only meaningful decision rule for each one of them will be $x_l^h < 1 \ \forall l = j^1, \ldots, j^{n_j}$, which discriminates between those values.

For efficient inference, given the new sample $\mathbf{x}$, each condition $s(\mathbf{x})$ for each node N_i is processed further to be represented in $n_{max} \cdot p$ slots, as shown in Figure 4.7. This representation enables checking whether the inference input being processed at N_i satisfies the condition, following the process outlined in the lower part of Figure 4.7, which is denoted with (B). The input used for inference is also represented with one-hot encoding across $n_{max} \cdot p$ slots. By performing the operations shown, the model value is multiplied by the input and then the sum of all slot values is shifted into the first slot. This step allows to move the inference result into the first slot, regardless of its original position. By subtracting this value from 1, we obtain the desired result as described in the image, returning 0 if the split condition is met, and 1 otherwise during the path evaluation in a

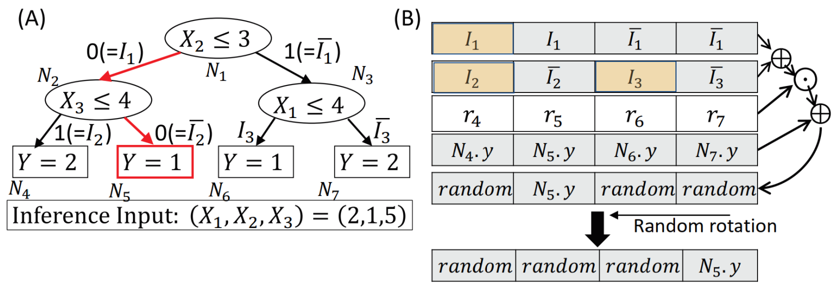


Figure 4.8. Path evaluation and label assignment [59]

decision tree.

At this point, let us denote the evaluation process and its result at node N_i as I_i, and consider the complement of the bit in I_i as $\overline{I}_i$. These individual I_i and $\overline{I}_i$ are referred to as blocks. Figure 4.8 illustrates how the tree shown evaluates the red path using the proposed method. The left part (A) shows the conditions relative to the nodes of the tree, the right part (B) explains how the correct branches are selected. The yellow-colored blocks represent the nodes where actual computations are performed to obtain evaluation results, while the remaining results are generated through rotations and addition operations. In this figure, blocks $I_1, \overline{I}_2$, and I_3 are computed as 0, while the rest are set to 1 because $x_2 \leq 3, x_3 > 4$ and $x_1 \leq 4$. Adding the top two ciphertexts of the blocks in (B) results in only the block corresponding to the leaf nodes along the red path in (A) having a value of 0, while the others contain non-zero values. Next, this result is multiplied by a randomly sampled plaintext polynomial and added to the ciphertext that contains the label of the target variable at the positions corresponding to each leaf node. This ensures that only the label of the final leaf node along the red path is stored in the corresponding slot of the ciphertext, while the other slots are filled with random values. To further obfuscate the positions of the leaf nodes containing the result, a random rotation is applied, producing the final inference result. In the proposed inference method, the client sends their input in a single ciphertext, where the one-hot encoded values of the independent variables are stored in the first $n_{max} \cdot p$ slots, without revealing any information about the decision tree to the client.

Since many decision trees constitute a random forest, this classifier is simply constructed by carrying out the aforementioned procedures. After that, the server sends to the client the classification result of each tree, and the client decrypts this data, performing majority vote in the plaintext domain. Table 4.5 serves as a recap of the aforementioned descripted methods.

Performances

[59] compared their work with a previous one, developed by Akavia et al. [76], who employ a low-degree polynomial approximation for the evaluation of decision rules. The results clearly demonstrate that the proposed method outperforms [76] across all datasets,

Reference	Scheme	Problem	Solution
[58]	TFHE	Tree traversal	Tensorial operations, quantization
[59]	CKKS	Homomorphic inference	Binary vectorial operations
[76]	CKKS	Decision rules management	Polynomial approximation

Table 4.5. Summary of Homomorphic DT papers

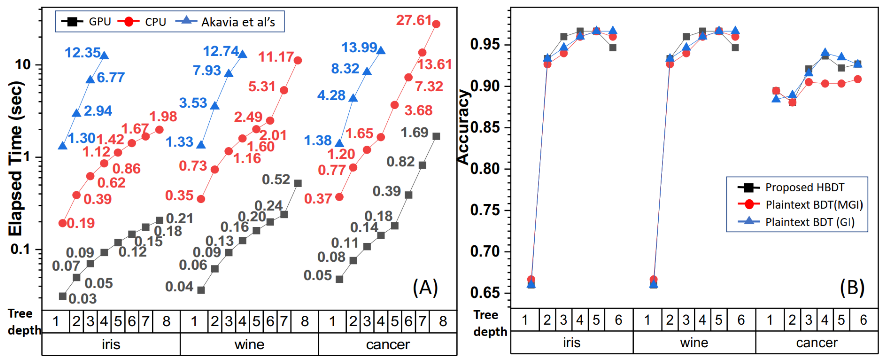


Figure 4.9. Inference: (A) execution time and (B) accuracy comparison [59]

Trees Depth	Encrypted accuracy	Unencrypted accuracy
1	0.800	0.807
2	0.920	0.933
3	0.960	0.967
4	0.960	0.973

Table 4.6. Trees depth vs accuracy of encrypted and unencrypted RF [59]

achieving speeds at least 3.7 times faster. For trees with a depth of 4 (16 leaf nodes), the proposed method delivers nearly eight times faster inference times. Figure 4.9 presents a comparison of inference execution times, showing that the CPU implementation is at least 3.7 times faster than Akavia et al.'s method. Additionally, GPU-assisted inference times ranged from 0.03 seconds (depth 1, Iris data) to 1.69 seconds (depth 8, Cancer data). Notably, with GPU acceleration, inference performance was found to be less than 200 times slower compared to plaintext inference on a single CPU core for trees with depths up to 4 in the same environment.
Figure 4.9 compares also the accuracy of the inference results between the encrypted model and models trained on plaintext data, using two different measures to define the splitting rules (Gini Impurity (GI) and Modified Gini Impurity (MGI), which is its more efficient version). As shown in the graph, there is little difference in accuracy between the model trained with the proposed method and the plaintext-based model trained with GI,

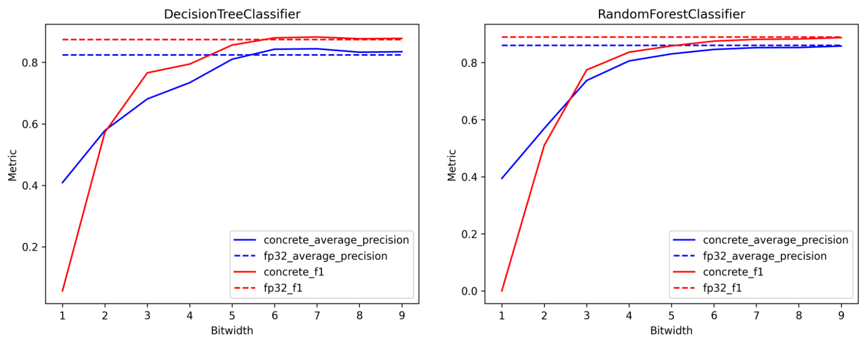


Figure 4.10. Experiment reporting the f1-score and average precision with varying precision on the spambase dataset [58]

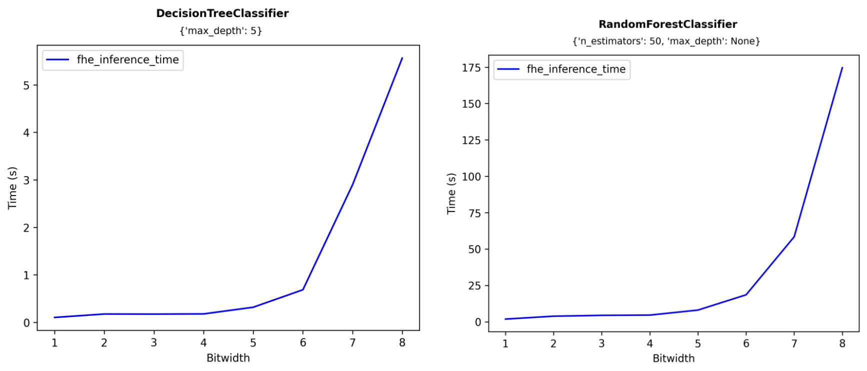


Figure 4.11. FHE inference time for different bit widths [58]

demonstrating comparable accuracy.
Finally, Table 4.6 shows some tests performed on the Iris dataset, highlighting the differences bewteen the encrypted and unencrypted RF. The experiments have been carried out by setting the number of trees $B = 64$ in advance. A small accuracy drop can be spotted in the encrypted model, and the gap slightly increases while the trees depth grows.

Regarding the TFHE implementation, Figure 4.10 shows the F1 score and average precision on the spambase dataset [77] at varying precision levels. This behavior is expected, and the intuitive choice would be to select the highest bit width for better accuracy. However, in Fully Homomorphic Encryption (FHE), execution time is affected by changes in precision. To better understand the impact of the quantization parameter, [58] conducted an experiment shown in Figure 4.11, where they measured the FHE inference time for the two models at different quantization precisions.

Figure 4.10 and Figure 4.11 together provide a clear view of the trade-off between model accuracy and FHE inference time. A significant increase in FHE inference time is observed starting at 7 bits. On the other hand, 5- and 6-bit precision yield metrics that are very close to the unencrypted model (FP32), with less than a 2% drop in reported metrics for 6-bit precision.

Chapter 5

Conclusion

Our analysis clearly shows that Fully Homomorphic Encryption represents a significant advancement in secure Machine Learning. By enabling computations on encrypted data, FHE not only protects sensitive information from unauthorized access but also fosters collaboration among various users. Its potential to enhance model evaluation and fairness makes FHE a powerful tool in the development of robust, privacy-preserving ML solutions that can address the complex challenges of modern data-driven applications. As the technology matures and becomes more accessible, FHE is likely to play an increasingly important role in ensuring the security and ethical use of Machine Learning across a wide range of industries.

In particular, the client-server architecture we analyzed ensures data privacy, secure computation and scalability. This is because client's sensitive information remains encrypted at all times, significantly reducing the risk of data breaches. Then, the server can perform complex computations without needing access to the underlying data, and the architecture can be easily scaled, as multiple clients can interact with the same server infrastructure, facilitating parallel processing of encrypted data.

5.1 Results

From the discussion of the previous chapter, it is clear that some algorithms present more difficulties than others. For instance, KNN is quite simple if tested on unencrypted data, but it is way more complicated if we try to perform homomorphic inference, because there are three different phases to deal with (as explained in section 4.2).

On the other hand, the inference phase of SVM just consists in the evaluation of the sign of a function: for this reason, some studies also analyzed the possibility of homomorphically training the algorithm (refer to section 4.3).

Finally, DT and RF have the problem of tree traversal, which is pretty complex to handle in the encrypted domain, and because of that the inference phase has to be adapted to the homomorphic world via quantization or tensorial operations (see section 4.4).

Regarding the employed scheme, we can notice that both TFHE and CKKS are equally utilized in the homomorphic version of KNN and DT and RF, whereas SVM features only the latter.

In general, we can say that the homomorphic algorithms are slower than their unencrypted counterpart, but many improvements have been made to reduce the gap. For what concerns accuracy of the encrypted models, it is usually a bit lower, but generally comparable to the standard algorithms run on plaintext.

5.2 Future works

In this work we described the homomorphic implementations for some of the main ML algorithms, but the field can be further explored by looking at other methods. For example, Logistic Regression [78][79] has been integrated with FHE, as well as all sorts of Neural Networks [80][81]. Furthermore, we have chosen to study the *classification* setting, without covering *regression*, which is used to predict a continuous output variable instead of a discrete one. It features many algorithms which have been developed in the homomorphic domain, implementing the encrypted version of methods such as Linear Regression [82][83]. We have also decided to focus only on *supervised learning*, where ML models are trained on some known data whose label is known. However, ML also includes *unsupervised learning*, where we do not have the label ground truth to train the model. Some of the most commonly used unsupervised methods have been integrated with FHE, such as K-means [84][85] and Principal Component Analysis (PCA) [86][87].

Besides the type of ML method, for widespread adoption, further advancements in computational efficiency and optimized algorithms will be essential. Future work should focus on developing hybrid approaches that combine FHE with other privacy-preserving methods, such as differential privacy [88][89] or secure multi-party computation [90][91], to create more scalable and efficient solutions. By overcoming these obstacles, FHE has the potential to become an integral part of ML practices, supporting ethical data usage and fostering public trust in ML applications.

Bibliography

[1] The tls protocol version 1.3. https://datatracker.ietf.org/doc/html/rfc8446.

[2] Bitlocker. https://learn.microsoft.com/it-it/windows/security/operating-system-security/data-protection/bitlocker/.

[3] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[4] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[5] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 325–341. Springer, 2005.

[6] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[7] Bootstrapping scheme. https://zhuanlan.zhihu.com/p/260033204.

[8] Programmable bootstrapping. https://www.zama.ai/post/tfhe-deep-dive-part-4.

[9] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 595–618. Springer, 2009.

[10] Dan Boneh, Shai Halevi, Mike Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*, pages 108–125. Springer, 2008.

[11] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

[12] Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

[13] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[14] Peter Scholl and Nigel P Smart. Improved key generation for gentry's fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, pages 10–22. Springer, 2011.

[15] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 377–394. Springer, 2010.

[16] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 24–43. Springer, 2010.

[17] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International algorithmic number theory symposium*, pages 267–288. Springer, 1998.

[18] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[19] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[20] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[21] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, pages 27–47. Springer, 2011.

[22] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234, 2012.

[23] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.

[24] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. Shield: scalable homomorphic implementation of encrypted data-classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, 2015.

[25] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[26] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[27] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*, pages 360–384. Springer, 2018.

[28] Oussama Amine. Overview of open source libraries for fully homomorphic encryption (fhe). Technical report, 2019.

[29] SEAL. https://github.com/Microsoft/SEAL.

[30] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886. Springer, 2012.

[31] HElib. https://github.com/homenc/HElib.

[32] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71:57–81, 2014.

[33] PALISADE. https://palisade-crypto.org.

[34] OpenFHE. https://openfhe.org.

[35] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. Openfhe: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 53–63, 2022.

[36] FHEW. https://github.com/lducas/FHEW.

[37] TFHE. https://github.com/tfhe/tfhe.

[38] Zama. https://www.zama.ai/.

[39] TFHE-rs. https://github.com/zama-ai/tfhe-rs.

[40] Concrete. https://github.com/zama-ai/concrete.

[41] ConcreteML. https://github.com/zama-ai/concrete-ml.

[42] Scikit-learn. https://scikit-learn.org/stable/.

[43] Lattigo. http://github.com/ldsec/lattigo.

[44] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 64–70, 2020.

[45] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[46] Jie Li, Yamin Liu, and Shuang Wu. Pipa: Privacy-preserving password checkup via homomorphic encryption. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 242–251, 2021.

[47] Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, and Radislav Vaisman. *Data Science and Machine Learning: Mathematical and Statistical Methods*. 2020.

[48] Random forest example. https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d.

[49] Albert Viladot Saló. Implementation of a privacy preserving knn algorithm based on fhe. 2023.

[50] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*, pages 221–256. Springer, 2020.

[51] Martin Zuber and Renaud Sirdey. Efficient homomorphic evaluation of k-nn classifiers. *Proceedings on Privacy Enhancing Technologies*, 2021.

[52] Yulliwas Ameur, Rezak Aziz, Vincent Audigier, and Samia Bouzefrane. Secure and non-interactive k-nn classifier using symmetric fully homomorphic encryption. In *International Conference on Privacy in Statistical Databases*, pages 142–154. Springer, 2022.

[53] Ahmad Al Badawi, Ling Chen, and Saru Vig. Fast homomorphic svm inference on encrypted data. *Neural Computing and Applications*, 34(18):15555–15573, 2022.

[54] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

[55] S Park, J Byun, and J Lee. Privacy-preserving fair learning of support vector machine with homomorphic encryption. association for computing machinery, 2022.

[56] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P Gummadi. Fairness constraints: A flexible approach for fair classification. *Journal of Machine Learning Research*, 20(75):1–42, 2019.

[57] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th international conference on world wide web*, pages 1171–1180, 2017.

[58] Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, Celia Kherfallah, Benoit Chevallier-Mames, and Arthur Meyre. Privacy-preserving tree-based inference with fully homomorphic encryption. *Cryptology ePrint Archive*, 2023.

[59] Hojune Shin, Jina Choi, Dain Lee, Kyoungok Kim, and Younho Lee. Fully homomorphic training and inference on binary decision tree and random forest. In *European Symposium on Research in Computer Security*, pages 217–237. Springer, 2024.

[60] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 238–252, 2021.

[61] Guozhu Xin, Yifan Zhao, and Jun Han. A multi-layer parallel hardware architecture for homomorphic computation in machine learning. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[62] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems*, pages 1295–1309, 2020.

[63] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. Towards case-optimized hybrid homomorphic encryption: Featuring the elisabeth stream cipher. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 32–67. Springer, 2022.

[64] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. 2023.

[65] Iris dataset. https://archive.ics.uci.edu/dataset/53/iris.

[66] Breast cancer dataset. https://archive.ics.uci.edu/dataset/14/breast+cancer.

[67] Heart dataset. https://archive.ics.uci.edu/dataset/45/heart+disease.

[68] Wine dataset. https://archive.ics.uci.edu/dataset/109/wine.

[69] Glass dataset. https://archive.ics.uci.edu/dataset/42/glass+identification.

[70] MNIST dataset. https://archive.ics.uci.edu/dataset/683/mnist+database+of+handwritten+digits.

[71] Saerom Park, Junyoung Byun, Joohee Lee, Jung Hee Cheon, and Jaewook Lee. He-friendly algorithm for privacy-preserving svm training. *IEEE Access*, 8:57414–57425, 2020.

[72] Schur complement. https://en.wikipedia.org/wiki/Schur_complement.

[73] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International conference on the theory and application of cryptology and information security*, pages 415–445. Springer, 2019.

[74] Adi Ben-Israel. An iterative method for computing the generalized inverse of an arbitrary matrix. *Mathematics of Computation*, pages 452–455, 1965.

[75] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 1–19. Springer, 2021.

[76] Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. *ACM Transactions on Privacy and Security*, 25(3):1–30, 2022.

[77] Spambase dataset. https://archive.ics.uci.edu/dataset/94/spambase.

[78] Xiaopeng Yu, Wei Zhao, Yunfan Huang, Juan Ren, and Dianhua Tang. Privacy-preserving outsourced logistic regression on encrypted data from homomorphic encryption. *Security and Communication Networks*, 2022(1):1321198, 2022.

[79] VVL Divakar Allavarpu, Vankamamidi S Naresh, and A Krishna Mohan. Privacy-preserving credit risk analysis based on homomorphic encryption aware logistic regression in the cloud. *Security and Privacy*, 7(3):e372, 2024.

[80] Lorenzo Rovida and Alberto Leporati. Encrypted image classification with low memory footprint using fully homomorphic encryption. *Cryptology ePrint Archive*, 2024.

[81] Adrien Benamira, Tristan Guérand, Thomas Peyrin, and Sayandeep Saha. Tt-tfhe: a torus fully homomorphic encryption-friendly neural network architecture. *arXiv preprint arXiv:2302.01584*, 2023.

[82] Guowei Qiu, Xiaolin Gui, and Yingliang Zhao. Privacy-preserving linear regression on distributed data by homomorphic encryption and data masking. *IEEE Access*, 8:107601–107613, 2020.

[83] Junyoung Byun, Saerom Park, Yujin Choi, and Jaewook Lee. Efficient homomorphic encryption framework for privacy-preserving regression. *Applied Intelligence*, 53(9):10114–10129, 2023.

[84] Ray-I Chang, Yen-Ting Chang, and Chia-Hui Wang. Outsourced k-means clustering for high-dimensional data analysis based on homomorphic encryption. *Journal of Information Science & Engineering*, 39(3), 2023.

[85] Peng Zhang, Teng Huang, Xiaoqiang Sun, Wei Zhao, Hongwei Liu, Shangqi Lai, and Joseph K Liu. Privacy-preserving and outsourced multi-party k-means clustering based on multi-key fully homomorphic encryption. *IEEE Transactions on Dependable and Secure Computing*, 20(3):2348–2359, 2022.

[86] Jung Hee Cheon, Hyeongmin Choe, Saebyul Jung, Duhyeong Kim, Dah Hoon Lee, and Jai Hyun Park. Arithmetic pca for encrypted data. *Cryptology ePrint Archive*, 2023.

[87] Samanvaya Panda. Principal component analysis using ckks homomorphic scheme. In *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5*, pages 52–70. Springer, 2021.

[88] Baiyu Li, Daniele Micciancio, Mark Schultz-Wu, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In *Annual International Cryptology Conference*, pages 560–589. Springer, 2022.

[89] Rezak Aziz, Soumya Banerjee, Samia Bouzefrane, and Thinh Le Vinh. Exploring homomorphic encryption and differential privacy techniques towards secure federated learning paradigm. *Future internet*, 15(9):310, 2023.

[90] Debasis Das. Secure cloud computing algorithm using homomorphic encryption and multi-party computation. In *2018 International Conference on Information Networking (ICOIN)*, pages 391–396. IEEE, 2018.

[91] Yongbo Jiang, Yuan Zhou, and Tao Feng. A blockchain-based secure multi-party computation scheme with multi-key fully homomorphic proxy re-encryption. *Information*, 13(10):481, 2022.