Martin Zuber* and Renaud Sirdey

Efficient homomorphic evaluation of k-NN classifiers

Abstract: We design and implement an efficient, secure, homomorphic k-Nearest Neighbours determination algorithm, to be used for regression or classification over private data. Our algorithm runs in quadratic complexity with regard to the size of the database but is the only one in the literature to make the secure determination completely non-interactively. We show that our secure algorithm is both efficient and accurate when applied to classification problems requiring a small set of model vectors, and still scales to larger sets of model vectors with high accuracy yet at greater (sequential) computational costs.

Keywords: k-Nearest Neighbours, Fully Homomorphic Encryption, Machine Learning, Secure Cloud Computing

DOI 10.2478/popets-2021-0020

Received 2020-08-31; revised 2020-12-15; accepted 2020-12-16.

1 Introduction

The k-Nearest Neighbours (k-NN) method is a machine learning tool that can be used either for classification (attributing a tag/class to an unknown feature) or regression (estimating the relationship between a certain amount of variables). The principle that drives a k-NN classifier is that closeness of two vectors in a Euclidean space is a good measure of class-similarity. This is of course only true for certain kinds of classification problems. For many such classification problems, k-NN classifiers have been found to be very efficient (compared with more recent deep learning methods for instance) and very accurate (though less so than said deep learning methods).

In this paper, we design a secure k-NN algorithm that can be used for both classification and regression of either a private input against a public database of neighbors or a public input against a private such database. We do so by means of homomorphic encryption techniques fine-tuned to this specific problem. As such, we thus propose an algorithm that finds the k nearest neighbours of a given vector and which is amenable to practical homomorphic evaluation timings. More precisely, the specific problem we aim to solve is known as the *k-NN determination problem* or the *k-NN problem* for short. It is the following. We are given a database of vectors that we will call *model vectors* in this paper. We are given a single vector we call the *source vector*. We aim to find the k closest vectors to the source vector among the model vectors using a given norm.

At the end of the paper, we evaluate our k-NN determination algorithm by using it to classify in a real-world context. We then compare with existing work on the topic.

In the following, we first expose a general overview of our contribution with respect to prior work on the subject. We then present the existing work on which we are building our contribution. Then comes the description of our novel k-NN algorithm. Finally, we detail experimental results obtained applying our algorithm on a real-world classification problem.

1.1 Prior work

As we present prior work done on establishing privacy-preserving solutions for k-NN computations, we need to distinguish three kinds of cases depending on the nature of the database used for the k-NN computation. All three of them aim to protect the confidentiality of the database.

- The database is owned by a single entity and placed on a remote cloud for outsourced computation using encryption.
- The database is made up of several databases from separate entities and placed on a remote cloud for outsourced computation using encryption.
- The database is made up of several databases from separate entities that keep their own database and use a distributed computation process to obtain the

***Corresponding Author: Martin Zuber:** CEA, LIST, E-mail: martin.zuber@cea.fr
Renaud Sirdey: CEA, LIST, E-mail: renaud.sirdey@cea.fr

final result between them and without a central server.

The third case - the distributed case - is not the topic of this paper. It differs fundamentally from the first two cases in that it assumes that the database owners have computational resources at hand, and are only concerned about privacy. That is a legitimate case-study. However it is not our focus. We refer to [1–10] for papers which tackle this latter setup.

The first and second cases are fairly similar in their use of encryption and the outsourced nature of the computation. We will call this the *outsourced k-NN* problem. Our goal is to achieve confidentiality for the database with respect to the cloud server. Remark that achieving database confidentiality with respect to the querier involves some subtleties since, although the querier does not have access to the database, she or he can extract some information about it based on the results of its queries which she can choose freely. Additionally, [11] presents an impossibility result that states that a scheme achieving private multi-client computing cannot be semantically CPA secure if it is purely a cryptographic scheme. This is a fundamental difference with the schemes that use distributed computing techniques. The interest of using encryption is in the comparatively much lower interactivity that such a scheme will incur.

Solutions in the literature vary but they all design a communication protocol between the different parties that we represent in Figure 1. There can be one or several *data owners*. The *data host* is the one making the k-NN computation. The *querier* is the entity requesting a k-NN computation over its source vector. In some cases, there needs to be a key distribution done by an entity we call Cryptographic Service Provider (CSP) here. Depending on the solution, there sometimes is a Trusted Computation Assistant (TCA) with access to the secret encryption key and interacting with the data host during the computation phase. We separate clearly all of these entities in Figure 1 in order to present a general framework for all existing schemes. However in any solution, some of those entities could be merged with others and assume several functions at once. Depending on our privacy requirements, we could for instance choose to have the data owner be the CSP. Several papers ([12–17]) address the outsourced k-NN problem and propose various solutions. We quickly review them below.

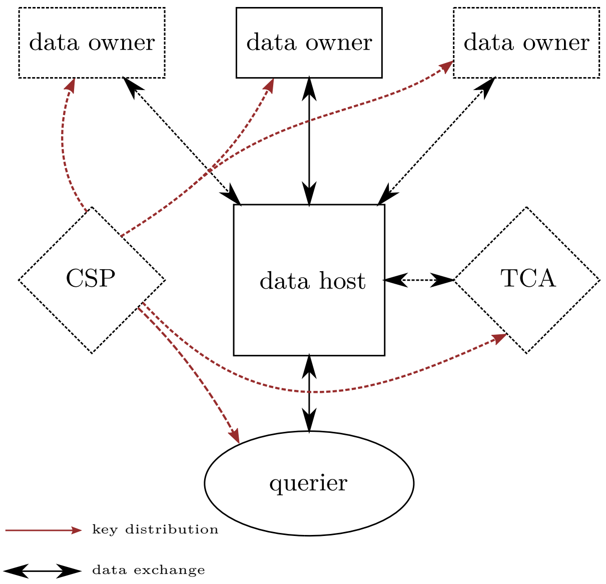


Fig. 1. A general design for protocols in the literature that tackle the *outsourced* problem. The data owner, data host and querier are always present and are drawn in full lines while other, "optional", entities are drawn with dashed lines. TCA: Trusted Computation Assistant. CSP: Cryptographic Service Provider.

In [12], the authors use an homomorphically additive encryption scheme. They use a TCA - which in this case is also the CSP - to distribute public encryption keys to the data owners and the querier while keeping the corresponding secret key. The encrypted distances for the k-NN computations are calculated by the data host. A kernel computation phase (k-NN optimization method) is made by the CSP, but it is worth noting that the distances are kept secret from it: they are masked by the data host before being sent to the CSP. The comparison and classification phases occur with interactions between the CSP and the DH. They use a garbled circuit (introduced in [18]) for this.

In [13], the authors build a system in which there is no data host, and the data is given to the querier in encrypted form. Then the queriers interact with a TCA (which has a decryption algorithm) to extract the result. One of the drawbacks of this scheme is the fact that the querier (called client in [13]) learns some incomplete plaintext information on the data from the TCA. In [15], the authors use this information to design an attack in the CPA model, in which the client can eventually obtain the actual data values in plaintext in the case of an honest TCA with no collusion. Because of this attack, we will not include this scheme into our comparison table (Table 1).

In [16], the authors propose a secure multiparty k-NN computation. The data owner outsources its encrypted database to the DH and its secret key to a TCA. Then the querier interacts with both of them to complete the k-NN determination in linear time.

In [14], the authors use what they call an "asymmetric scalar-product-preserving encryption". This self-explanatory type of encryption allows them to do away with a TCA. The queries and the database are encrypted using two different encryption algorithms but with the same key. Then the data host can compute the comparisons between every distance. The k encrypted data points that are closest to the query can then be sent back to the querier for decryption. This means the data host has access to the scalar products $p \cdot q$ for every data point p and every query q sent by the querier. Therefore, in a CPA attack model, with the data host as an attacker, it can choose the queries q and [15] shows that it can recover the data points through a set of linear equations (as many as there are queries). However, the data host can only mount a CPA attack with the help of the querier (it needs an encryption key which is secret). This means that this scheme is not resistant to collusion between the data host and the querier. This is actually the case of all other encryption-based schemes we encountered in the literature. It is also the case for our scheme.

One exception is that of the scheme in [15]. They only circumvent this problem by providing the data host with encryptions of portions of the database. The final result will not be an actual nearest neighbour but rather the nearest partition. This scheme is structurally different from all other schemes we mention here because it solves another problem in order to circumvent this structural security flaw. This is why we will not be comparing our scheme to it.

In [17], the authors design a secure k-NN scheme based on both additive homomorphic encryption and garbled circuits (therefore requiring interactions between the data host and the querier). They provide a secure implementation of two k-NN algorithms: a linear scan algorithm which has a linear complexity; and a clustering-based algorithm which has a sublinear complexity but does not aim to output the exact k nearest neighbours.

Table 1 compares the schemes we presented from the literature with our own scheme with respect to the two most important attributes for a secure k-NN determination scheme: its non-interactivity and its complexity. The schemes mentioned here diverge in a lot of ways. Most notably, in the way that they implement (or not) specific optimizations to the k-NN algorithm (kernels for instance) or whether they implement a secure majority-class voting at the end. We chose to focus on the basic k-NN algorithm. As a general rule we can see that our scheme, compared with pre-existing ones, trades non-interactivity with a higher complexity. This makes our scheme asymptotically slower (though still practical for real-world problems as is shown in Section 5.4) but requires no interaction for the k-NN computation. All other schemes require some sort of interaction.

One exception is [14] which is both linear in complexity and works offline. However that scheme provides a lower level of security: the use of their scalar-product-preserving encryption means the data host will have access to the distance values in the clear. We propose a scheme which guarantees (to the level of the security of the underlying encryption scheme) that no information will leak during the k-NN computation.

	this work	**[12]**	**[16]**	**[14]**	**[17]**
Non-Interactive	✓	✗	✗	✓	✗
Complexity	x^2	x	x	x	x

Table 1. This table shows a comparison of the two most important attributes for a secure k-NN scheme: its non-interactivity and its complexity. The complexity of the corresponding scheme is linear when x is used, and quadratic when x^2 is used. As mentioned, [17] introduces two secure k-NN algorithms and one of them is sublinear. However this clustering-based algorithm does not output the exact result.

1.2 Our contribution

In this paper, we propose a fully homomorphic k-NN algorithm. This algorithm works in two settings. If the database is encrypted and the query is clear, then the computation is handled by the querier. If the query is encrypted and the database is clear, then the computation is handled by the database owner. Achieving encryption for both the query and the database is within reach but not yet attainable due to a high noise propagation issue. These two "settings" allow us to design two protocols for secure k-NN determination where the computation itself is fully non-interactive, and where the security of both the query and the database is ensured. Figure 2 represents the two protocols. We differentiate the two protocols depending on who performs the computation.

The *DB owner computation* protocol has the query q be sent encrypted by the querier, to the data owner using a bracket notation for encrypted data. The data owner then applies our secure k-NN algorithm over an encrypted query and a clear database and sends the encrypted result r back to the querier.

The *query owner computation* protocol is represented here with several queriers to better illustrate the scenario we present in section 2. In this case the data owner sends its database to queriers and receives their respective results encrypted.

Remark: Information leakage. It should however be emphasized that, although these architectures offer provable security on the raw data being exchanged using encryption, the guarantees they offer on the *confidentiality* of the database (resp. the query) from threats coming the querier (resp. the data owner) are less formal as, in both cases, having access to the result of the classification leads to some (useful) leakage on the raw database (resp. query). This latter leakage is inherent to the service provided and, if not acceptable, can be mitigated by other countermeasures such as noising the output of the classifier (differential privacy gives us the theoretical toolbox to do so meaningfully). This countermeasure is hard to consider in serious use-cases such as medical ones, where patient health - and therefore classification accuracy - cannot be traded for database confidentiality. Another possible countermeasure would be throttling the request rate, which can be performed by a given entity.

We show that we can implement both of these protocols with quadratic complexity with respect to the size of the database but constant with respect to k. We show that they achieve close to optimal classification rate over a real-world classification problem. Over this classification problem our scheme is very efficient. We also test our algorithm on a benchmark database to give the reader a better idea of our algorithm's scaling ability.

2 Scenario and threat model

This section aims at pointing out the kind of classification use cases that can be meaningfully addressed by the ability to run a k-NN classifier in the homomorphic domain. We do so in an abstract fashion, independently of the underlying FHE techniques (which are duly detailed in the rest of the paper). We start by considering the concrete e-health scenario in which a doctor (*operator*) wishes to benefit from a diagnosis service provided by a laboratory (*server*) for one of its patient (*user*) on a given class of pathologies. The laboratory is able to provide this service thanks to a highly sensitive database of previous cases which it cannot share with third parties (due to either or both legal constraints or commercial value) and which it uses at the core of a k-NN classifier. In this context, a privacy-preserving diagnosis service can work as follows. The doctor is the owner of an FHE private key. The doctor encrypts the patient's data which it then sends to the laboratory. The lab evaluates the k-NN classification of the (encrypted) patient data in the homomorphic domain against its database (which is in clear form) and produces an (encrypted) classification. The latter (encrypted) classification is then sent to the doctor for decryption and interpretation. In terms of privacy, this kind of protocol has several desirable properties. The raw patient data remains private from the lab (as it sees and manipulates them only in encrypted form). Additionally, the laboratory's precious database of past cases remains on its own server and its privacy is also preserved from the doctor and the patient. This scenario corresponds to the *DB owner computation* case illustrated in Figure 2.

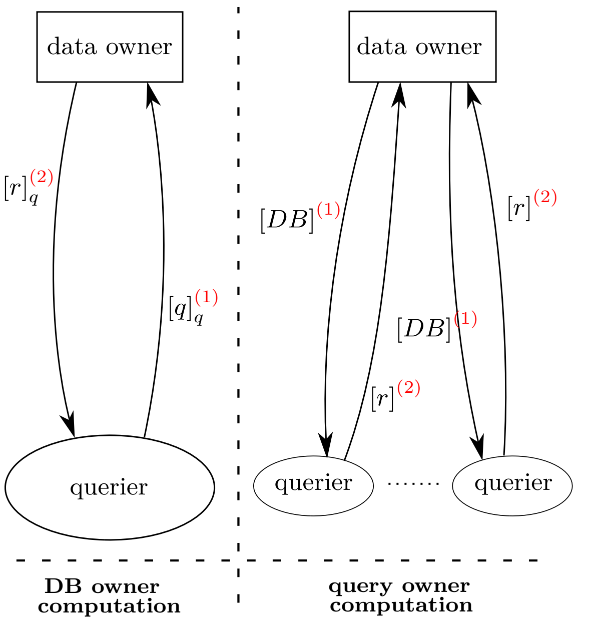


Fig. 2. Two examples of protocols for a secure k-NN computation that our scheme allows for. We use simple brackets ($[\cdot]$) to indicate that data is encrypted by the data owner and brackets with an index q ($[\cdot]_q$) to indicate that data is encrypted by the querier. The order in which the data is exchanged is indicated in red: $q^{(2)}$ means q is the second data exchange to happen in the protocol.

Beyond scenarios in which a private (encrypted) request is run over a cleartext database, the case where a cleartext request is run over a private (encrypted) database is also of practical relevance. To stay in the medical field, we can consider the case of a pharmaceutical company (*operator*) which needs to run an epidemiological study over a set of cases stored by one or more hospitals (*server*s). For normative and legal reasons, as is usual in the medical field, the data owned by a hospital cannot leave its information system. Yet, the pharmaceutical company does not wish to reveal the set of model vectors (the database) it will use to perform its study by means of a k-NN classifier (although this does not prevent it to disclose the nature of the study when applicable legislation requires it). In this context, the firm can then send its database of reference vectors encrypted under its own FHE public key and the hospital can then homomorphically evaluate the k-NN over its set of patient data (the queries), sending the (encrypted) classifications back to the firm for further analysis once processing complete (note that, in this second scenario of batch processing of a possibly large database under mild latency constraints, the non-interactivity of our approach is highly desirable to avoid unnecessary resource-wasting synchronization during the processing). In this context, the raw hospital queries are not disclosed to the pharmaceutical company and stay on premise. Additionally, the sensitive data from the firm is not disclosed to the hospital. The above illustrative example also applies in many operator/server situations (e.g. cyberattack threat monitoring with private criteria in cleartext traffic, recognition of private faces in cleartext video streams [19], ...). This scenario corresponds to the *query owner computation* case illustrated in Figure 2.

Lastly, note that our solution does not allow to address the case where both the request and the database are encrypted. We believe that adapting our solutions to allow for the encryptions of both of them will be possible soon. It would require an increased precision in the underlying FHE library we used for implementation and would come at, as of yet unknown, performance costs.

3 Preliminaries

3.1 Notations

We use $\mathbb{B}$ to denote the set $\{0, 1\}$. The cardinal of a finite set A is written $|A|$. We denote signed integers by $\mathbb{Z}$, the reals by $\mathbb{R}$ and use $\mathbb{T}$ to denote the real torus mod 1. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ will denote the integers mod q with a given integer q. We write $\mathbb{T}_N[X]$ the quotient $\mathbb{R}[X]/(X^N + 1)$ mod 1. We write $\|\cdot\|_p$ to denote the ℓ_p norm of vectors over reals or integers.

3.2 The TFHE encryption scheme

The TFHE scheme [20, 21] implemented in the TFHE library [22] is based on both the LWE and ring-LWE problems. The LWE problem was introduced by Regev in [23] and the ring-LWE problem in [24]. Importantly, both work by introducing an error into the ciphertext. When the ciphertext is added or multiplied to other ciphertexts, that error will grow, inducing what we call here an *error propagation*. The larger the noise, the lower the probability of correct decryption. We will not go into the details of the LWE problem or the TFHE scheme and refer the reader to the original papers. We will only present here the information necessary to understand the algorithm that we build and implement. Ciphertexts in TFHE can either be in $\mathbb{T}^{n+1}$ (where n is an integer parameter) or in $\mathbb{T}_N[X]^{k+1}$ (where k and N are integer parameters). The former encrypts a scalar value in $\mathbb{T}$ and the latter a polynomial message in $\mathbb{T}_N[X]$. The secret key is taken from $\mathbb{B}^n$ and $\mathbb{B}_N[X]^k$ respectively.

In the notation that we use, we differentiate ciphertexts based on whether they encrypt a polynomial or a scalar value. Therefore a ciphertext of a polynomial $\mu[X]$ will be written as $[\mu[X]]^{(r)}$ and one for a scalar μ as $[\mu]$.

3.3 Problem definition

The k-NN problem aims to find the k Nearest Neighbours of a given vector (the source vector) among a number of vectors (the model vectors).

The problem that we aim to solve is to have the data host perform a k-NN determination over an encrypted set of reference vectors, non-interactively. It is given encryptions of d real model vectors $c^{(i)} \in \mathbb{R}^\gamma, i \in \{1, d\}$ of dimension γ by the data owner. The data owner also gives it two bootstrapping keys which are used in TFHE computations and cannot be used to retrieve encrypted data without a decryption key. The querier sends it its query: one clear source vector $m \in \mathbb{R}^\gamma$. Again, it could very well be the source vector that is hidden under a layer of encryption and the model vectors that are clear. See Figure 2 for a presentation of the two protocols. It is made evident in section 4.5.2 that the distance computation can be done in the same way in either case and the rest of the computation is identical.

The data host's goal is to obtain d encrypted binary values - one for every model vector - with a 1 for every model vector that is among the k closest vectors to the source and a 0 for every model vector that is not. The results of the k-NN computation need to have a correct decryption with overwhelming probability.

3.4 FHE operations

In this section we present the different operations in TFHE and their noise propagations. Since they are based on the LWE problem, the TFHE encryption scheme relies on a noise to be introduced in the ciphertext. We assume that it is a Gaussian noise unless stated otherwise and refer to it through its standard deviation that we denote σ. With every homomorphic operation (except the bootstrap operation) the noise grows. This is a fundamental issue in FHE in general and in this paper in particular. We add to the notation of the ciphertext a possible mention of an encryption key s and a noise α as such: $[*]_{s,\alpha}$.

– Internal Addition/Subtraction : $[*] \times [*] \to [*]$
  Given two ciphertexts $[\mu_1]$ and $[\mu_2]$, we can add them and obtain a ciphertext $[\mu_1 + \mu_2]$. The exact same operation can be applied to polynomial ciphertexts.

– External Multiplication : $* \times [*] \to [*]$
  Given an integer scalar a and a ciphertext $[\mu]$, we can multiply them and obtain a ciphertext $[a \times \mu]$. The exact same operation can be applied to polynomial ciphertexts with polynomials μ and a. This actually just corresponds to several iterations of internal additions or subtractions.

– Extraction : $[*]^{(r)} \to [*]$
  From a ring ciphertext of a polynomial $\mu[X] = \sum_{i=0}^{N-1} \mu_i^i$, it is possible to extract a scalar ciphertext of a single coefficient of μ_p at a position $p \in \{0, N-1\}$. We can do this at no cost to the noise of the ciphertext. We can extract similarly the corresponding scalar secret key from the initial ring secret key. This means the owner of the initial ring secret key can also decrypt the extracted ciphertext.

– Sign Bootstrapping[1] : $[*]_{s,\alpha} \to [*]_{s',\alpha_b}$
  From two keys s and s', we can create an object $\mathsf{BK}_{s \to s'}$ (BK for short) called the bootstrapping key, with a precision depending on parameters ℓ and B_g. Given an integer b, a ciphertext $[\mu]_{s,\alpha}$ of a scalar value μ encrypted using the key s with noise α, and this bootstrapping key BK, we can obtain a ciphertext $[\mu_0]_{s',\alpha_b}$ where $\mu_0 = 1/b$ if $\mu \in [0, \frac{1}{2}]$ and $\mu_0 = 0$ if $\mu \in [\frac{1}{2}, 1]$. Very importantly, α_b is fixed by the parameters of the bootstrapping key BK and does not depend on the initial standard deviation. We call it *sign bootstrap* because the function that it applies (it could apply other functions) can be considered a sign computation. Indeed, if a value in $[0, \frac{1}{2}]$ (upper half of the torus in Figure 3) is considered positive and a value in $[-\frac{1}{2}, 0]$ (lower half of the torus) is considered negative, the operation outputs 0 for a negative input and $\frac{1}{b}$ for a positive input.
  This operation therefore allows us to both apply a sign function to the input ciphertext and reduce its noise down to α_b. Figure 3 is a representation of this operation. The application of the function is not infinitely precise. The figure illustrates that there is a range of inputs (the red zones) for which the operation does not necessarily output the correct value: it will output a random value. This is not a problem when the parameters are chosen appropriately.

4 Our k-NN algorithm

In this section, we will present our general algorithm solving the k-NN problem over encrypted inputs. However, for simplicity sake, we will first present it with all inputs and outputs as clear values. We show in section 4.5 how and why this algorithm can be used efficiently in a fully homomorphic setting. All of the operations that we use here are therefore clear-value equivalents of the homomorphic operations presented in section 3.4. First of all, we assume that the integer k (as in k-NN) is set.

4.1 Distance computation

What interest us are not actually the distances between the model vectors and the source vectors themselves but

[1] This bootstrapping is only a slight variation on the bootstrapping procedure introduced in [20], we just add a public rotation to the bootstrap operation used in [25].

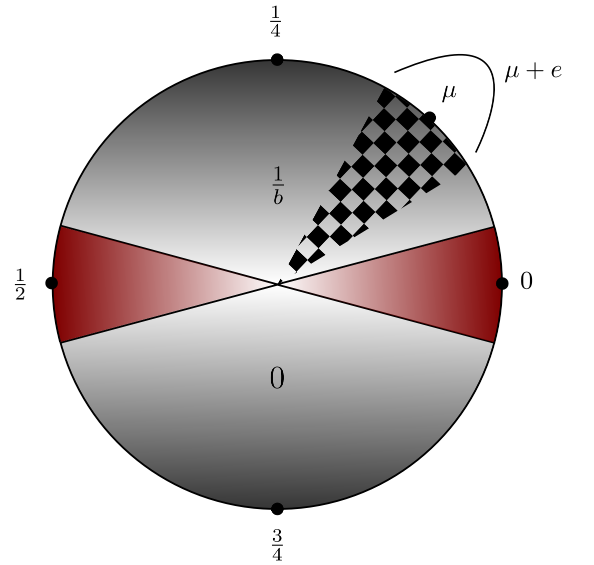


Fig. 3. The bootstrapping operation represented on the torus. As indicated in the figure, any value in the upper half of the torus will yield an encrypted output of $\frac{1}{b}$ with b a given base; and a value in the lower half an encrypted output of 0. There is a range (red zones) around 0 and around $\frac{1}{2}$ where the bootstrapping operation will return a random value.

rather a comparison of distances. In fact we are going to compute the difference of the squares of distances. This means that if $\mathbb{d}_i$ is the distance between vector i and the source for a given $i \in \{1, d\}$, then we want to compute $\mathbb{d}_i^2 - \mathbb{d}_j^2$ for every $i, j \in \{1, d\}$.

4.2 Delta values

For all i, j in $\{1, d\}$, we compute the sign of the difference of the two squared distances $\mathbb{d}_i^2, \mathbb{d}_j^2$:

$$\begin{aligned} \delta_{i,j} &= 1 \quad \text{if} \quad \mathbb{d}_i^2 < \mathbb{d}_j^2 \\ &= 0 \quad \text{otherwise} \end{aligned}$$

These δ values will help us select the smallest distances. They form a matrix:

$$\begin{pmatrix} 0 & \delta_{1,2} & \cdots & \delta_{1,d} \\ \delta_{2,1} & 0 & \cdots & \delta_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{d,1} & \delta_{d,2} & \cdots & 0 \end{pmatrix}$$

4.3 A first partial solution

From here, we present a solution which works in theory, but is not applicable in practice for any number d of vectors. However, this idea is the basis for the final k-NN algorithm that we propose, the one we present in section 4.4. The idea is to sum all of the rows from the δ matrix together as in [19]. We obtain:

$$\begin{pmatrix} \Delta_0 & \Delta_1 & \cdots & \Delta_{d-1} \end{pmatrix} \quad \text{where} \quad \Delta_i = \sum_{j=0}^{d-1} \delta_{j,i}$$

All of the Δ_i are between 0 and $d-1$. If i is the index of the greatest distance, then $\Delta_i = 0$. It corresponds to the distance for which every comparison to another distance yields a 0. Respectively, the smallest distance has an associated Δ equal to $d-1$. We want to select the k greatest Δ values. For this, we can apply $\texttt{Sign}_k$, an operation that applies the following transformation:

$$\begin{aligned} \texttt{Sign}_l : \mathbb{N} &\to \mathbb{B} \\ x &\mapsto 0 \quad \text{if } x \leq d-1 \\ &\quad\;\; 1 \quad \text{if } x > d-l \end{aligned}$$

Therefore, we would obtain an encryption of 1 only for the indexes of the k closest vectors to the source, and an encryption of 0 for every other index.

Just as in [19], this solution is limited by the fact that, in an homomorphic setting, we cannot add an unlimited amount of ciphertexts together for a given set of parameters. This is due to the fact that, after some point, the noise distribution in the output ciphertext becomes too great for accurate decryption.

One can in theory change the parameters to allow for any number of sums if that number is known beforehand. This is in the case of a Levelled Homomorphic Encryption (LHE). But in practice, this is not true for any number of inputs d, since we are limited by the precision of the homomorphic library used in the implementation. To put it in other words, if one were to design a levelled homomorphic solution for the k-NN problem, one would be limited in the number of inputs it could parse. Since we want to design a fully homomorphic k-NN algorithm, we have to assume that the set of parameters are set independently of the number d of input vectors. Given that set of parameters (see section 5.1 for our choices), we call m the maximum number of ciphertexts that we can add together before applying a bootstrapping operation.

This m value changes in theory depending on the noise of the ciphertexts to add. If a ciphertext is fresh (a direct encryption of a plain value), then usually its noise will be lower than if a ciphertext has been created by summing two other ciphertexts together. We choose parameters in section 5.1 that induce the following property: δ ciphertexts and ciphertexts that are outputs of a bootstrapping operation have the same noise distribution. This means that there exists a single value m so that we can add m δ ciphertexts and/or bootstrapped ciphertexts together and not more.

At this point in [19], the authors just apply a bootstrapping operation to the sum of m δ values to reset that value to either 0 or 1 and then keep adding new δ values. This works to find the single closest vector because the bootstrapping only selects the vector with minimum distance every time. This would not work here because we need to select more than one vector and therefore keep more information at every bootstrapping step. Our solution is presented in section 4.4.

4.4 The scoring algorithm solution

From this point on, we set an index i for one of the model vectors, and aim to produce a 0 value if that vector is not among the k nearest neighbours to the source and a value of 1 if it is. We consider that $k < m$. We will see in section 5.3.1 that this will always be the case in practice.

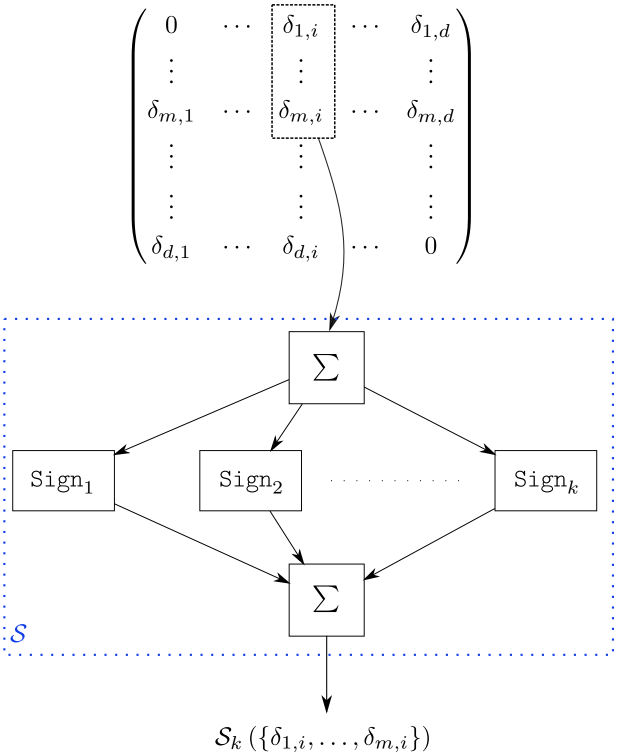
We introduce a scoring algorithm to solve the k-NN problem over an arbitrary amount of input vectors. Figure 4 presents the algorithm applied in the first phase of our k-NN computation: over the first m values in column i. The algorithm takes m inputs and has one output. Its building blocks are the summing operation and Sign operations which applying the following transformation:

$$\begin{aligned} \texttt{Sign}_l : \mathbb{N} &\to \mathbb{B} \\ x &\mapsto 0 \quad \text{if } x \le m-l \\ &\quad\;\; 1 \quad \text{if } x > m-l \end{aligned}$$

We can formalize our scoring operation as the following function $\mathcal{S}_{k,m} : \mathbb{N}^n \to \{0,\dots,k\}$:

$$\begin{aligned} \mathcal{S}_{k,m} : (x_1,\dots,x_n) \mapsto &\max\left(0\ ,\ k-m+\sum_{l=1}^{n} x_l\right) \\ &\varnothing \quad \text{if } \sum_{l=1}^{n} x_l > m \end{aligned}$$

For the output to be in $\{0,\dots,k\}$, we need to have $\sum_{l=1}^{n} x_l \le m$. This corresponds to the condition that lead us to design this algorithm.



$0 \;\cdots\; \delta_{1,i} \;\cdots\; \delta_{1,d}$
$\delta_{m,1} \;\cdots\; \delta_{m,i} \;\cdots\; \delta_{m,d}$
$\delta_{d,1} \;\cdots\; \delta_{d,i} \;\cdots\; 0$
Σ
Sign$_1$
Sign$_2$
Sign$_k$
Σ
$\mathcal{S}$
$\mathcal{S}_k\left(\{\delta_{1,i},\dots,\delta_{m,i}\}\right)$

Fig. 4. A figure presenting our scoring algorithm. Here it is presented parsing through the first m values in one column - the first phase of the k-NN computation. The values $\delta_{1,i},\dots,\delta_{m,i}$ are summed and then their sum is parsed by the Sign operations. The k outputs are summed again at the end. The operation - the dotted blue box called $\mathcal{S}$ - outputs a score between 0 and k we write $\mathcal{S}_{k,m}\left(\{\delta_{1,i},\dots,\delta_{m,i}\}\right)$.

When applied to binary inputs ($\mathbb{B}^m$), this operation counts the number of 0s and returns either $(k - \#0)$ or 0. An example, for $m = 7$ and $k = 3$:

$$\begin{aligned} \mathcal{S}_{3,7}(0,1,1,1,0,0,1) &= \max(0, k-3) = 0 \\ \mathcal{S}_{3,7}(1,0,1,1,1,1,0) &= \max(0, k-2) = 1 \\ \mathcal{S}_{3,7}(1,1,0,1,1,1,1) &= \max(0, k-1) = 2 \end{aligned}$$

In our k-NN computation, we apply it on the δ values from the matrix, which are binary values. And if $\delta_{j,i} = 0$, that means that vector j is closer than vector i to the source. Therefore the number of 0s in our i^{th} column counts the number of vectors that are closest to the source than vector i. This is why we call it a scoring operation. The higher the score given by our operation, the closer vector i is to the source. The operation, in an FHE setting, is still limited to inputs of sum lower than m. The output is in the range $\{0,\dots,k\}$. Therefore, since $k < m$, after applying the scoring operation once, there is "room" to add $m-k$ more values to the output and applying the operation again. Even if the output were 0, in an FHE setting, that information is not known, therefore we have to plan everything as if

the output value were its maximum value: k.
Let's formalize this with a proposition.

Proposition 1. *Let $x_1, \ldots, x_{2m-k} \in \mathbb{B}$.*
Let A denote:

$$\mathcal{S}_{k,m}\left(x_1, \ldots, x_{m-k},\, \mathcal{S}_{k,m}\left(x_{m-k+1}, \ldots, x_{2m-k}\right)\right)$$

Let B denote:

$$\mathcal{S}_{k,2m-k}\left(x_1, \ldots, x_{2m-k}\right)$$

in an FHE setting, we cannot compute B directly because it would require us to sum more than m binary values. However we can compute A and:

$$A = B$$

A proof of this proposition is provided in appendix A.

Therefore, by iterating enough times the scoring operation, we can obtain the score of a set of any size. This means we can compute

$$\mathcal{S}_{k,d}\left(\delta_{1,i}, \ldots, \delta_{i-1,i}, \delta_{i+1,i}, \ldots, \delta_d\right) \tag{1}$$

Which is equal to 0 if vector i is not among the k nearest neighbours and a non-zero value if it is one of the k closest vectors to the source. Note that we removed $\delta_{i,i}$ from the computation as its value is trivial and known to be 0. In fact, (1) is not exactly the value that we are looking for. We want a constant output value of 1 for the k closest vectors. This means the last scoring operation is replaced with a sum and the $\texttt{Sign}_k$ operation. The full algorithm is represented in Figure 5. Furthermore, if - for the last sum - there are less than $m-k$ values left in matrix column, then for the last $\texttt{Sign}_k$ operation to output the correct value, we need to pad the sum with an appropriate value.

4.4.1 An example execution

An example of our k-NN algorithm applied in the case of $d = 7$ model vectors is given in Figure 6. We are looking for the $k = 2$ closest vectors to the source vector. We can only add $m = 4$ δ values at a time. The model vectors are represented in a drawing on top in their Euclidean space as small stars, and the source vector is the larger star. As we can see from the figure, the two closest vectors are vectors 2 and 7. We do not go into the details of the distance computation and the distance comparisons that go into creating the associated δ matrix. We show how to apply the algorithm on the 5th vector. From the drawing, we can see that it is the 3rd closest vector and therefore we should obtain a result of 0. The matrix diagonal is "removed" in the sense that we do not take it into account. We take the first $m = 4$ δ values for the 5th column and make them go through the scoring algorithm $\mathcal{S}$. Since there is one 0 among those first values, the output of the scoring algorithm at this point is $k - 1 = 1$. The output of the scoring algorithm is at most of size $k = 2$. Since we do not know its value (in an FHE setting), we can only add $m - k = 2$ more δ values to it (we cannot have a sum go above the value m).
Therefore, the rest of the algorithm consists of adding 2 fresh values to the output of the last scoring operation and running a new scoring operation. Here we are left with only two values, therefore we add them with the output of the first $\mathcal{S}$ box and compute the score. We find an overall score of 0, which means that the 5th vector is not among the 2 closest. We do not know anything else about it.
If the matrix was of size 6, then we would add a 7th line of 1s to the matrix as padding. Since the algorithm scores according to the number of 0s, this would not affect the overall score but it would allow us to have the right number of inputs for the scoring algorithm. In the context of an FHE computation, this padding would be added as plaintext values and therefore induce no noise.

4.5 Going FHE

From the start of section 4, we have so far presented a k-NN algorithm that works over clear values. To apply our algorithm in an FHE setting we need to use the equivalent homomorphic operations on encrypted inputs.

4.5.1 Encoding and encryption

First of all, we will be using a torus-based homomorphic encryption scheme: TFHE. This means that our values are not actually real values but rather torus values in $[0, 1]$. Every real value i is therefore actually a torus value $\frac{i}{b}$ with $b > i$ a given integer, called the *base*. When a real value i is encrypted as $\left[\frac{i}{b}\right]$, it can be rescaled to $\left[\frac{i}{b'}\right]$ with b' a different base by applying an external scalar multiplication by $\frac{b'}{b}$. However, we designed a scheme that only needs to define three bases:

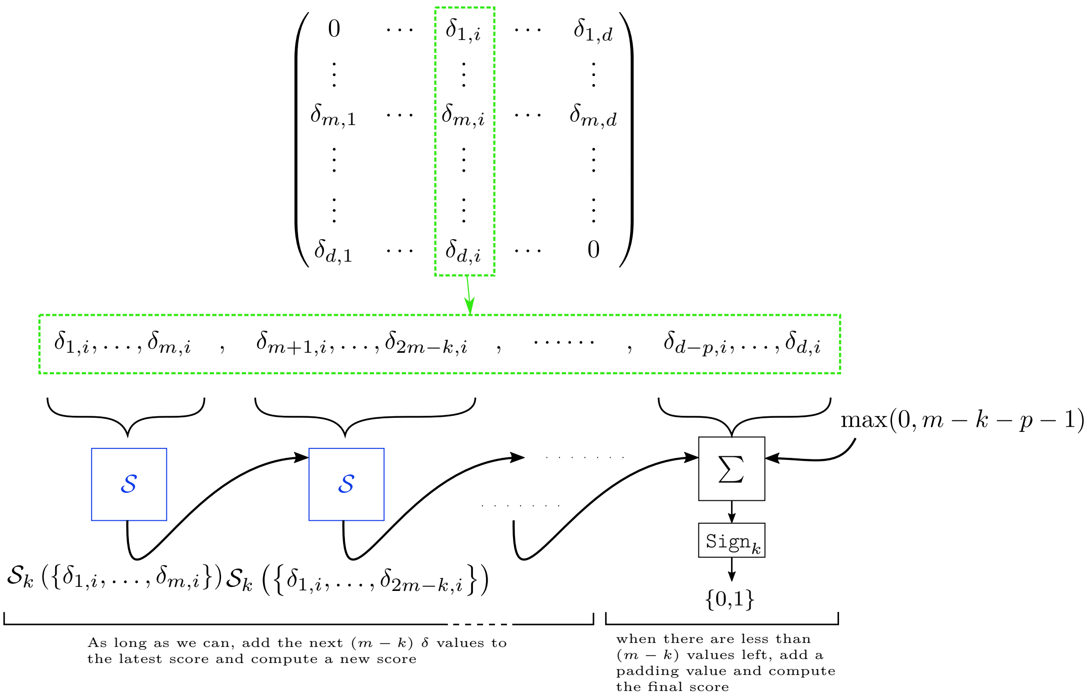

$\delta_{1,i}, \dots, \delta_{m,i}$, $\delta_{m+1,i}, \dots, \delta_{2m-k,i}$, ... , $\delta_{d-p,i}, \dots, \delta_{d,i}$
$\mathcal{S}$
$\mathcal{S}$
Σ
$\max(0, m-k-p-1)$
$\mathtt{Sign}_k$
{0,1}
$\mathcal{S}_k(\{\delta_{1,i}, \dots, \delta_{m,i}\})$
$\mathcal{S}_k(\{\delta_{1,i}, \dots, \delta_{2m-k,i}\})$
As long as we can, add the next $(m-k)$ δ values to the latest score and compute a new score
when there are less than $(m-k)$ values left, add a padding value and compute the final score

Fig. 5. A figure representing the final score computation for a given vector i. The blue $\mathcal{S}$ boxes correspond to the scoring algorithm as represented in the dotted blue box in Figure 4. It is important to note that for simplicity sake, we did not make it clear in the figure that we remove $\delta_{i,i}$, the diagonal value, from the computation. It is however important to do it.

one as a common encoding base for all the model vectors and the source vector: ν; one as a common encoding base for all the δ values: b_δ; one as the encoding base for the final output values: b_{f}.

Therefore, to take an example, a δ value that we presented in section 4.4 as being equal to either 0 or 1 will actually be equal to either 0 or $\frac{1}{b_\delta}$. And a sum of m such values will be in $\{0, \frac{1}{b_\delta}, \dots, \frac{m}{b_\delta}\}$.

4.5.2 Squared distance difference computation

Every squared distance is computed between an encrypted and a non-encrypted vector as explained in section 3.3. Again we assume that the source vector is plain and the model vectors are encrypted. The opposite yields a strictly equivalent way to compute the distances and from then on, everything is encrypted and therefore nothing changes. We now give specific names to our vectors and present the distance computation in detail.

We are given d real model vectors: $c^{(i)} \in \mathbb{R}^\gamma, i \in \{1, \dots, d\}$ of dimension γ. However, we encode these vectors as torus vectors in order to have them be encrypted in the THFE encryption scheme. Therefore we define a base $\nu > \max_i \left(\left\|c^{(i)}\right\|_\infty\right)$ and rescale every vector by ν to obtain torus vectors.

We call $\mu \in \mathbb{R}^\gamma$ the source vector. First of all, it needs to be rescaled using the same value used for the model vectors: ν. Secondly, when going FHE, we have to run the k-NN computation on an integer source vector and not a real one. However μ is a real data vector and therefore needs to be rounded to an integer vector after a scaling is performed to preserve the precision. Therefore, with an additional rescaling factor $\tau \in \mathbb{N}$, we actually have to transform every μ_i into $\lfloor \frac{\tau \times \mu_i}{\nu} \rceil \in \mathbb{N}$. We then encode this rounded and rescaled source vector as a polynomial:

$$M = \sum_{l=0}^{\gamma-1} \left\lfloor \frac{\tau \times \mu_{\gamma-l}}{\nu} \right\rceil \cdot X^l$$

However, introducing this τ factor by itself would change the distance computation result. Therefore, we need to rescale the model vectors by τ as well. Encoding the twice-rescaled model vectors a polynomials gives us: $\forall i \in \{1, d\}$,

$$C^{(i)} = \sum_{l=0}^{\gamma-1} \frac{c_{l+1}^{(i)}}{\tau\nu} \cdot X^l$$

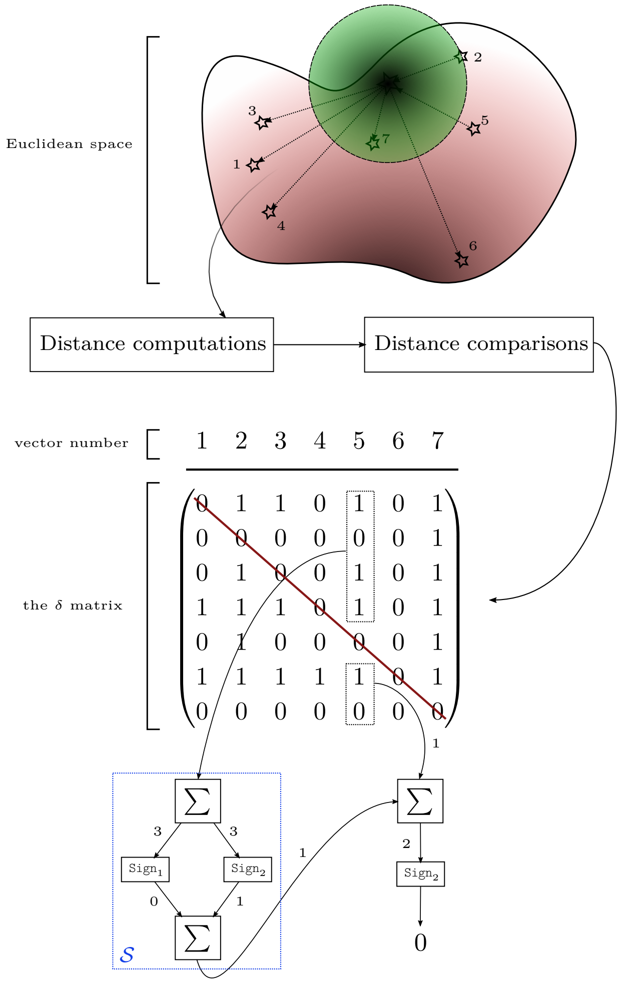


Fig. 6. An example of our k-NN algorithm applied in the case of $d = 7$ model vectors. We are looking for the $k = 2$ closest vectors to the source vector. We can only add $m = 4$ δ values at a time.

We are actually given an encryption $\left[C^{(i)}\right]^{(r)}$ for every i. And we want to obtain encryptions of every differences of every distances between vectors $c^{(i)}$ and μ. In other words, if we write $\mathbb{d}_i$ the distance from μ to $c^{(i)}$ for every i, then we want to obtain $\left[\frac{1}{\nu^2}(\mathbb{d}_i^2 - \mathbb{d}_j^2)\right]$ for every $i \neq j$.

In order to simplify the computation, we assume that the party that encrypted the $\left[C^{(i)}\right]^{(r)}$ ciphertexts also pre-computed and encrypted:

$$A_i = \left(\sum_{l=1}^{\gamma} \left(\frac{c_l^{(i)}}{\nu}\right)^2\right) \cdot X^{\gamma-1}$$

for every i.

At this point, given M, $\left[C^{(i)}\right]^{(r)}$ and $[A_i]^{(r)}$ we can compute:

$$2M \cdot \left(\left[C^{(j)}\right]^{(r)} - \left[C^{(i)}\right]^{(r)}\right) + [A_i]^{(r)} - [A_j]^{(r)} \quad (2)$$

Proposition 2. *For τ a high enough power of* 10*, if we extract the $(\gamma - 1)^{th}$ coefficient from* (2)*, we obtain* $\left[\frac{1}{\nu^2}(\mathbb{d}_i^2 - \mathbb{d}_j^2)\right]$.

A proof of this proposition is provided in appendix A. The value of τ will determine the extent to which the result is approximated. In section 5.3.1, we can see that experimentally, allowing some approximation does not impact the overall k-NN result significantly.

This homomorphic computation can only work if the difference does not go above $\frac{1}{2}$ or below $-\frac{1}{2}$. Indeed, we will consider a value between 0 and $\frac{1}{2}$ to be positive and one between $-\frac{1}{2}$ and 0 to be negative. Therefore overflowing in the difference computation will cause us to return a wrong sign value. This requires choosing a base value ν appropriate for the given data range. We explain our choices for the different base values in section 5.1.

4.5.3 Delta computation

At this point, we want to obtain the sign of the differences $\left[\mathbb{d}_i^2 - \mathbb{d}_j^2\right]$ for every $i \neq j$ (it is the same as that of $\left[\frac{1}{\nu^2}(\mathbb{d}_i^2 - \mathbb{d}_j^2)\right]$). We do this using the bootstrapping operation we presented in 3.4 and used in several works in the literature. To illustrate it we show a torus representation of this sign bootstrapping operation in Figure 3. The output of the sign boostrapping operation applied to $\left[\frac{1}{\nu^2}(\mathbb{d}_i^2 - \mathbb{d}_j^2)\right]$ is $[\delta_{i,j}]$. As mentioned in section 4.5.1, the output values will either be 0 or $\frac{1}{b_\delta}$ with b_δ a given base. We do not actually compute all of the δ values as $\delta_{i,j} = 1 - \delta_{j,i}$ for every i, j. Therefore, there are only $\frac{1}{2} \cdot (d^2 - d)$ bootstrapping operations.

4.5.4 The scoring operation

Figure 7 is a representation of the scoring operation applied to torus variables. To represent any variable, we present it in the figure as every value it can take in the torus: 0 and $\frac{1}{b_\delta}$ for δ variables for instances. The scoring operation starts with a sum of δ variables. The sum of ciphertexts is a standard operation. As shown in Figure 7, the resulting ciphertext can hold one of $m + 1$ different values from 0 to $\frac{m}{b_\delta}$. We want the greatest possible value to be $\frac{1}{2}$ and therefore we need to set $b_\delta = 2m$.

From there intuitively, we apply k rotations to obtain k ciphertexts on which we run the sign bootstrapping operation. In figure 3, the sign bootstrapping is represented as giving an output of 0 for negative values. For this bootstrapping operation, we still apply a sign operation but invert the outputs so that our bootstrapping outputs $\frac{1}{b_\delta}$ for "negative" values. The values are not actually seen as negative here but rather as *overflowing* from the rotation we apply. We can see in Figure 7 that only the greatest possible value will yield an output of $\frac{1}{b_\delta}$ over a sign bootstrapping of the first rotated ciphertext. Only the l greatest possible values will yield an output of $\frac{1}{b_\delta}$ over a sign bootstrapping of the l^{th} rotated ciphertext.

This figure therefore represents the homomorphic equivalent of the `Sign` operations. However, though this is exactly how these `Sign` operations work intuitively, their implementations are slightly more subtle. The homomorphic operations $\texttt{Sign}_1, \ldots, \texttt{Sign}_k$ are all computed together at the same time on the same input, and can be seen as one single operation with one input and k outputs. We can do this by applying a sign bootstrapping over the first rotated ciphertext and then extracting k different outputs where the usual sign bootstrapping only extracts one. It therefore corresponds to a slight variation on the original bootstrapping operation and allows us to compute our k-NN determination at virtually a constant cost with regard to k.

Algorithm 1 presents the modified bootstrapping algorithm that, given an encrypted input, computes the $\texttt{Sign}_1, \ldots, \texttt{Sign}_k$ operations on that input homomorphically. The notations that we use are those found in the original presentation of the bootstrapping algorithm in [20] (Algorithm 3: Bootstrapping procedure). We do this to stress where our algorithm deviates from the original version. As done in [20], we use the $\mathsf{LWE}(\mu)$ notation to present a `TLWE` encryption of μ. Lines 1 to 7 describe the start of the standard bootstrapping procedure. In that original procedure, the last step is an extraction of the first coefficient of the ACC ciphertext: $\mathsf{SampleExtract}(\mathsf{ACC},\ 0)$ (see Section 3.4 for a brief presentation of the extraction operation). In our variation, we extract k different coefficients to virtually no additional performance cost. This algorithm is provided to ensure the reproducibility of our results and a reader wishing to understand all of the notations should refer to [20].

Algorithm 1: Homomorphic `Sign` operations

```
Input: A TLWE sample $(a, b) = [\mu]$ of an unknown
       message $\mu \in \mathbb{T}$, a bootstrapping key $\mathsf{BK}$, a
       bootstrapping base $b_\delta$, a number $k$ of
       desired outputs. The symbol $\boxdot$ represents
       the internal multiplication operation.
Output: $k$ $\mathsf{LWE}$ samples $(u_j)_{1 \leq j \leq k}$ where $u_j \in$
        $\mathsf{LWE}\left(\frac{1}{b_\delta} \text{ if } \mu + \frac{(j-1)}{b_\delta} \in \left]0, \frac{1}{2}\right[ \ ; \ -\frac{1}{b_\delta} \text{ else}\right)$
Let $\overline{b} = \lfloor 2Nb \rceil$                                              1
for $i = 1$ to $n$ do                                                  2
    Let $\overline{a}_i = \lfloor 2Na_i \rceil$                                           3
Let testv $= \left(1 + X + \cdots + X^{N-1}\right) \times X^{-\frac{2N}{4}} \cdot \frac{1}{b_\delta}$            4
$\mathsf{ACC} \leftarrow \left(X^{\overline{b}} \cdot (0, \text{testv})\right)$                                      5
for $i = 1$ to $n$ do                                                  6
    $\mathsf{ACC} \leftarrow \left[h + (X^{-\overline{a}_i} - 1) \cdot \mathsf{BK}_i\right] \boxdot \mathsf{ACC}$                   7
for $j = 1$ to $k$ do                                                  8
    $u_j = \mathsf{SampleExtract}\left(\mathsf{ACC},\ \lfloor \frac{4jN}{b_\delta} \rceil\right)$                      9
```

Remark: (parallelization)
In section 3.3, we mention that the data host is given two bootstrapping keys to compute the homomorphic operations the k-NN determination requires. This is because the bootstrapping operation does three things: it outputs a ciphertext with a fixed noise (therefore reducing the input noise if it was higher); it applies a pre-programmed function (here the sign function or its opposite); it switches the encryption key with another one. This key switching means we cannot re-apply the same bootstrapping operation twice in a row. Therefore Fig. 5 does not represent the real order in which the δ values are used as inputs for the $\mathcal{S}$-boxes.

We can actually design an FHE implementation in the *linear* order presented Fig. 5. It would require the use of a key-switching operation after every $\mathcal{S}$-box. This additional operation would decrease the efficiency of the scheme. It would at the same time decrease the bandwidth costs by eliminating the need for a second bootstrapping key (a key-switching key is much lighter than a bootstrapping key).

We presented a *linear* order of computation in Fig. 5 for simplicity sake. However, the real precedence relation between the $\mathcal{S}$-boxes offers more degrees of freedom and we were able to implement our scheme with better efficiency by finding of an ordering which removed the need for a key-switching. In essence, we divide our

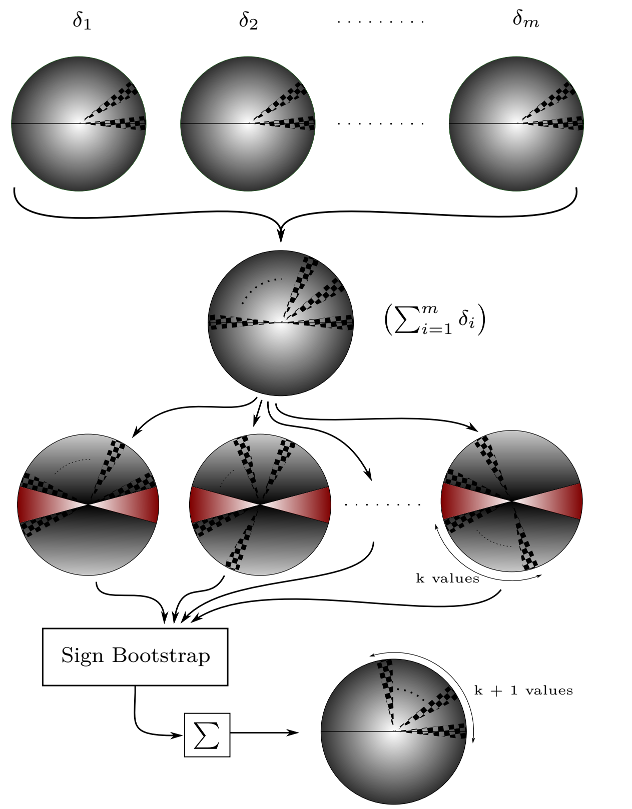


Fig. 7. A figure showing how the scoring operation intuitively works in the torus. The circles, as in other figures in this article, represent the torus. The squared areas in each circle represent every possible value that the given variable can take. For instance, every δ variable can have either a value of 0 or of $\frac{1}{b_\delta}$. After summing m of them, we obtain a variable that can take one of $m+1$ values.

δ values into chunks of size m and apply $\mathcal{S}$-boxes on each chunk. Then we do the same on the output values of the $\mathcal{S}$-boxes. Additionally, this partial ordering allows for parallelization at run-time, although the timings presented in the paper do not exploit this. Two bootstrapping keys are then needed for the implementation of the scheme.

5 Experimental results

We fully implemented the homomorphic k-NN scheme described above, and integrated it with a real-world classification problem. In this section we provide the details on our implementation, k-NN training and efficiency and accuracy of our FHE k-NN classification.

5.1 FHE implementation

The TFHE encryption scheme is implemented in the TFHE library[2]. That is what we use to implement our homomorphic algorithm. We use it in its 64-bit torus representation and its dedicated AVX assembly version for FFT computations. The choice of parameters depends both on questions of *security* and *accuracy*. The parameters needed to replicate our results are presented in Table 2. As in [25], we base the security of our scheme on the `lwe-estimator`[3] script. The estimator is based on the work presented in [26]. It allows us to find the smallest initial noise for our ciphertexts, that still ensures security and gives us the most leeway in terms of noise propagation. As for accuracy, there are two issues that we need to solve to ensure our scheme outputs an accurate result.

One issue is the precision of the bootstrap operation. As shown in Figure 3, the bootstrap operation, given a set of parameters, has a red zone around 0 and $\frac{1}{2}$ where input values yield an uncertain output. We can arrange for the values of $\frac{1}{\nu^2}(\mathbb{d}_i^2 - \mathbb{d}_j^2)$ to never reach the zone around $\frac{1}{2}$ by increasing ν. However, increasing ν too much will mean having values too small for the bootstrap operation to work accurately. We find the appropriate value for ν through a scilab script, simulating FHE operations over the dataset on which the homomorphic k-NN is implemented (see Section 5.2.1).
An erroneous output by the bootstrapping operation, when it happens, does not necessarily impact the accuracy of the overall result. If none of the two distances in question is actually one of the k smallest, then the mistake changes nothing to the end-result. In the case where one of the k smallest distances is overlooked and another one is chosen in its stead, we can assume the new distance is close to the one that was overlooked. Therefore in a majority class voting setting (where we determine the class of the source depending on the majority class of its k nearest neighbours) the output determination has a good chance of still being correct. This qualitative analysis does not prove the accuracy of our scheme. Rather it helps explain its theoretical resilience to mistakes. We show in section 5.3.1 that, in practice, our scheme is highly accurate in a real-world setting.

[2] https://tfhe.github.io/tfhe/
[3] https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py

The other issue pertaining to accuracy is the correct final decryption. The initial ciphertexts are purposefully noisy and, the noise increases with every non-bootstrapping operation. This *noise propagation*, if too great, can lead to an incorrect decryption. The parameters have to be carefully chosen so that it does not. Again, this is done using a scilab simulation.

λ	N	σ
110	**1024**	**1e−9**

N_b	σ_b	B_g	ℓ
1024	**1*e*-9**	**64**	**6**

m	ν	τ	b_δ	$b_\mathfrak{f}$
65	**3**	**100**	$\mathbf{4m-4}$	**4**

Table 2. The parameter tables for our implementation. The top-left table presents the overall security (λ), and the parameters for the initial encryption: σ is the Gaussian noise parameter and N the size of polynomials. In the TFHE polynomial encryption scheme (TRLWE), there is a parameter k independent from the one we use to refer to the k-NN determination problem here. That parameter was set to 1 in every case. The top-right table presents the parameters needed to create the two bootstrapping keys we are using. For details on the use of B_g and l, see [20]. The bottom table presents parameters specific to our k-NN scheme as presented in Sections 4 and 4.5. m is the maximum number of sums of fresh δ ciphertexts we can do before a bootstrapping operation. τ is used to rescale the data before encryption. ν, b_δ and $b_\mathfrak{f}$ are introduced in Sect. 4.5.1 and are encoding bases used respectively for the initial data, the output of the intermediate scoring operations and the output of the final scoring operation. b_δ and $b_\mathfrak{f}$ are parameters of the scoring operation presented in Algorithm 1.

5.2 Experimental setup

5.2.1 The datasets

Breast Cancer dataset. As a first example of a secure k-NN application, we use a dataset[4] previously used with the purpose of testing machine learning algorithms. Specifically, it is a dataset containing 569 instances (vectors) of 30 attributes (meaning the vectors are of dimension $\gamma = 30$). Each of those instances is extracted from a digitized image of a "fine needle aspirate of a breast mass" from a single patient with a tumor in her breast. Every instance is obtained from a different patient. The data is labeled with a class: the tumor can be malignant or benign. This dataset was retrieved from an online open archive of machine learning datasets [27]. Our goal is to build a secure k-NN classifier that can determine with high accuracy whether a given instance is one of a malignant or a benign tumor by comparing it to an encrypted set of labelled model vectors.

MNIST dataset. We present a second application of our algorithm: on the widely used MNIST dataset[5]. This dataset is used to test the classification of handwritten digits (0 to 9). In fact, we do not use the raw MNIST dataset (which consists of 60,000 training images and 10,000 testing images all of them with 28x28 pixels) but we use a pre-processed subset of the data provided by the scikit-learn[6] library. This subset includes 1,797 images of size 8x8.

5.2.2 The classification process

In order to showcase the efficiency and accuracy of our scheme, we design our own k-NN classification algorithm over the dataset. Since, as discussed in Sect. 2, we are interested only in the homomorphic evaluation of *already trained* k-NN classifiers, we train our algorithm in the clear domain. In this case, *training* means selecting a subset of the learning data that will be used as reference vectors (model vectors) in a k-NN computation as well as selecting the most appropriate k value. The classification is then made homomorphically and consists of running our k-NN algorithm over the encrypted reference vectors selected in the training phase, comparing them to a cleartext instance. The results are then decrypted and a majority voting takes place: the instance is attributed the class shared by a majority of the k model vectors selected as its closest.

With respect to this, our goal is not to design the best clear-domain classification algorithm for the given problem but rather to build a “good enough” clear-domain algorithm and show that its encrypted-domain counterpart works just as well or almost as well and efficiently so. In particular, with respect to efficiency, as the computing time of any algorithm running in the encrypted domain over FHE *cannot by construction depend on its input data*, we could even have used a random selection of reference vectors and obtained the very same performance results. To allow for our results to be fully reproducible, we present our training and classification process in detail in Appendix B.

[4] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

[5] http://yann.lecun.com/exdb/mnist/

[6] https://scikit-learn.org/stable/

5.3 Performance results

5.3.1 Classification rate

Breast Cancer dataset. For a measure of our algorithm's accuracy, we can compare with the results of previous work (see [28] for instance) which regularly present classification rates between 94% and 97% depending on the method used. We train our k-NN classifier in the clear and obtain a classification rate of 95.2% with $k = 3$ and $d = 10$ model vectors.

We implemented the FHE classification algorithm and ran it on the encrypted model vectors selected during the training phase. The overall classification rate was 95.0%. This means "going FHE" only slightly reduces the accuracy of our k-NN classifier. Therefore, rounding the classified instances to integer vectors and keeping real model vectors does not affect the overall accuracy that much. Furthermore, our algorithm works well here not because all of the instances are quite far from one another but rather in spite of the opposite. With the given parameters, we determined that around 4% of the differences of squared distances are lower than the precision threshold of our sign bootstrapping operation. However, as mentioned in Section 5.1, a k-NN classifier can be quite resilient to such computational mistakes.

In all of our testing, we found values for k that are low (7 or below) not because it was easier for us computationally (we mentioned that it changes almost nothing to the efficiency of the algorithm) but rather because k values are always very low in a k-NN classifier. [29] reviewed an important amount of k-NN algorithms at the time and determined that in 75% of the cases, the $k = 1$ classifier was the best one.

Interestingly, in the FHE case, we found that $k = 3$ gives significantly better results than $k = 1$. When $k = 1$, the difference in classification rate between a clear classification and an FHE one is 3% on average. The difference when $k = 3$ never exceeds 0.4%. This is due to the fact that, as mentioned in Sect. 5.1, a majority-class voting scenario ($k > 1$) attenuates the impact of a single error significantly.

MNIST dataset. An application of our algorithm on the well-known MNIST database allows us to provide benchmarks for the efficiency and accuracy of our algorithm. To that end, we test our algorithm with 5 different model sizes. Each size is chosen as the smallest model size able to achieve a given benchmark classification rate on the MNIST database. Table 3 shows the values that were chosen for model sizes (d) and their respective benchmark clear classification rates. It shows the classification rates achieved with each model through the FHE classifier. We use $k = 3$ for every classification.

d	40	175	228	269	457
clear rate (%)	80	95	96	97	98
fhe rate (%)	79.5	94.8	95.4	96.4	97.3

Table 3. k-NN classification rates over the MNIST database. This table gives the MNIST classification rates both in the clear case and in the fhe case, depending on the size of the model (d).

Overall the accuracy loss is relatively low. Importantly, it does not depend on the size of the model: this is due to the fact that we built a fully homomorphic scheme and not a levelled scheme. Since the parameters are the same regardless of the size of the problem, on average, the loss of accuracy is constant. The relatively high number of model vectors needed to obtain a similar classification rate to that of the breast cancer database result can be explained (in part) by the higher number of classes. The MNIST database requires a classification among 10 different classes (there are 2 classes for the breast cancer database), this increases the amount of model vectors needed for an accurate classification.
s

5.4 Timing results

As mentioned previously, our algorithm has a quadratic complexity with regard to the number of training instances (model vectors). This is because we are computing $\frac{d^2-d}{2}$ sign bootstrapping operations to obtain the δ values. After that, we need to run the scoring algorithm on each column of the δ matrix, with roughly $d \times \frac{d}{m-k}$ $\mathcal{S}$-box evaluations in total. Of course, this is a less efficient k-NN algorithm compared with existing work in the clear as, some optimizations in the literature allow *approximate* k-NN classifiers to run in sublinear time. However our work is the only one to achieve a fully non-interactive secure k-NN classifier, and the quadratic complexity is the price that we pay for that. This can be explained intuitively by the fact that any such secure non-interactive classifier cannot use conditional gates to reduce the complexity of the algorithm: if it did, it would leak a lot of information to the data host. We are running the operations on an Intel Core i7-6600U CPU. All of the times that we present here correspond to sequential computation times. Our scheme is highly parallelizable as mentioned.

Breast Cancer dataset. The real-world breast cancer detection problem we applied our algorithm on requires us to find the $k = 3$ closest vectors to a source vector among $d = 10$ model vectors. When applied to this problem, a single k-NN classification takes 4 seconds to finish *sequentially.* In the first application scenario presented in Sect. 2, a patient and/or her doctor are sending an encrypted query for remote classification. In our case this means they would have to wait around 4 seconds for an answer. Given that an average medical consultation takes around 15 minutes, the response time is appropriate. In the second scenario of an epidemiological study as presented in Sect. 2, a pharmaceutical company running this study over several remote databases of 500 patient vectors *each* would have to wait under 35 minutes (for *sequential* computations) to get the results of their study. It is an appropriate latency for such a use-case.

MNIST dataset. As for the MNIST database, Table 4 shows how much time a single classification takes depending on the size of the model. For instance, in order to classify with 95% accuracy, one classification would take just under 12 minutes of *sequential* computation-time to complete (recall, as already emphasized in Sect. 4.5.4, that our scheme is highly parallelizable and that these timings can be significantly decreased through multi-core execution). We recall that each model size is chosen as the smallest model size able to achieve a given benchmark classification rate on the MNIST database.

d	40	175	228	269	457
time (min)	**0.5**	**11.6**	**18.3**	**25.4**	**70.8**

Table 4. Sequential timings for a single MNIST k-NN homomorphic classification, depending on the size of the model (d). The time is given in minutes.

5.5 Bandwidth usage

The bandwidth usage depends on which scenario (among the two presented in Section 2) is used for the secure k-NN classification. In any case, two bootstrapping keys have to be sent (once at setup time), which - given the parameters used here - correspond to 200 MBytes. The encrypted result is sent in every case as 10 `TLWE` samples amounting to 40 KBytes. Let us present the amount of bandwidth used on top of that in both our scenarios.

Encrypted model vectors. In the k-NN scenario presented in Sect. 4.5, the model vectors are encrypted and the query is in the clear. This means that $d\ \left[C^{(i)}\right]^{(r)}$ ciphertexts and $d\ [A_i]^{(r)}$ ciphertexts have to be sent (with d the size of the model). For the breast-cancer problem ($d = 10$), the encrypted data to be transferred amounts to 160 KBytes. In the case of the MNIST database, the size of the data transfer is given in table 5 depending on the size of the model used.

d	40	175	228	269	457
bandwidth usage (MBytes)	**0.6**	**2.8**	**3.6**	**4.3**	**7.3**

Table 5. Amount of bandwidth usage (on top of the bootstrapping key) for the different model sizes used in our MNIST database application, in the case of an encrypted model.

Encrypted query. In the case where our scheme is used to send an encrypted query to be compared with a set of clear model vectors, then the communication overhead prior to the computation amounts to a single `TRLWE` sample (8 KBytes).

6 Conclusion

In this paper, we designed and implemented a novel secure k-NN algorithm requiring no interaction between involved entities during the computation phase. To the best of our knowledge, it is the first such algorithm. It uses exclusively special purpose fully homomorphic encryption building on the versatility of the TFHE encryption scheme. Although our algorithm is quadratic in complexity it achieves practical homomorphic sequential execution performances for small sizes of model vector sets. Furthermore, it may still achieve performances meeting the latency constraints of real-world scenarios requiring larger such sets by means of improvements achievable through basic multi-core parallelization.

Acknowledgments

This research was funded in part by a PhD grant from Commissariat à l'Energie Atomique, France.

References

[1] H. Rong, H. Wang, J. Liu, and M. Xian. Privacy-preserving k-nearest neighbor computation in multiple cloud environments. *IEEE Access*, 4:9589–9603, 2016.

[2] M. Burkhart and X. Dimitropoulos. Fast privacy-preserving top-k queries using secret sharing. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pages 1–7, Aug 2010.

[3] Feng Zhang, Gansen Zhao, and Tingyan Xing. Privacy-preserving distributed k-nearest neighbor mining on horizontally partitioned multi-party data. In Ronghuai Huang, Qiang Yang, Jian Pei, João Gama, Xiaofeng Meng, and Xue Li, editors, *Advanced Data Mining and Applications*, pages 755–762, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[4] Murat Kantarcıoğlu and Chris Clifton. Privately computing a distributed k-nn classifier. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, pages 279–290, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Li Xiong, Subramanyam Chitti, and Ling Liu. K nearest neighbor classification across multiple private databases. In *CIKM*, 2006.

[6] Li Xiong, Subramanyam Chitti, and Ling Liu. Preserving data privacy in outsourcing data aggregation services. *ACM Trans. Internet Techn.*, 7, 08 2007.

[7] Y. Qi and M. J. Atallah. Efficient privacy-preserving k-nearest neighbor search. In *2008 The 28th International Conference on Distributed Computing Systems*, pages 311–319, June 2008.

[8] M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. In *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, pages 541–545, Dec 2006.

[9] J. Zhan and S. Matwin. A crypto-based approach to privacy-preserving collaborative data mining. In *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, pages 546–550, Dec 2006.

[10] Jeongsu Park and Dong Lee. Privacy preserving k -nearest neighbor for medical diagnosis in e-health cloud. *Journal of Healthcare Engineering*, 2018:1–11, 10 2018.

[11] Marten Van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, HotSec'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[12] Frank Li, Richard Shin, and Vern Paxson. Exploring privacy preservation in outsourced k-nearest neighbors with multiple data owners. In *Proceedings of the 2015 ACM Workshop on Cloud Computing Security Workshop*, CCSW '15, page 53–64, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450338257. 10.1145/2808425.2808430. URL https://doi.org/10.1145/2808425.2808430.

[13] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *2011 IEEE 27th International Conference on Data Engineering*, pages 601–612, April 2011.

[14] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 139–152, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2.

[15] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 733–744, April 2013.

[16] B. K. Samanthula, Y. Elmehdwi, and W. Jiang. k-nearest neighbor classification over semantically secure encrypted relational data. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1261–1273, 2015.

[17] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. SANNS: scaling up secure approximate k-nearest neighbors search. *CoRR*, 2019.

[18] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86. IEEE Computer Society, 1986.

[19] Martin Zuber, Sergiu Carpov, and Renaud Sirdey. Towards real-time hidden speaker recognition by means of fully homomorphic encryption. 2019.

[20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53887-6.

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Improving tfhe: faster packed homomorphic operations and efficient circuit bootstrapping. *IACR Cryptology ePrint Archive*, page 430, 2017.

[22] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[23] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. ACM, 2005.

[24] V Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*. Springer, 2010.

[25] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Proceedings of CRYPTO 2018*. Springer, 2018.

[26] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169 – 203, 2015. URL https://www.degruyter.com/view/journals/jmc/9/3/article-p169.xml.

[27] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[28] David Bingham Skalak. *Prototype Selection for Composite Nearest Neighbor Classifiers*. PhD thesis, USA, 1997.

[29] C Feng, A Sutherland, R King, S Muggleton, and R Henery. Comparison of machine learning classifiers to statistics and neural networks. In *Proceedings of the Third International Workshop in Artificial Intelligence and Statistics*, 1993.

A Proofs

In this section, we provide proofs of propositions 1 and 2.

proof of Proposition 1. There are two cases to look at.

Case 1: $$\sum_{l=m-k+1}^{2m-k} x_l \leq m-k$$

then we have

$$\mathcal{S}_{k,m}\left(x_{m-k+1},\ldots,x_{2m-k}\right)=0$$

and since $x_1,\ldots,x_{2m-k}\in\mathbb{B}$, then $\sum_{l=1}^{m-k} x_l \leq m-k$. Therefore $A=0$. Also,

$$\begin{aligned}\sum_{l=1}^{2m-k} x_l = \sum_{l=1}^{m-k} x_l + \sum_{l=m-k+1}^{2m-k} x_l\\ &\leq m-k+m-k\\ &\leq 2m-2k\end{aligned}$$

And we have

$$B=\max\left(0\ ,\ 2k-2m+\sum_{l=1}^{2m-k} x_l\right)$$

Therefore $B=0=A$.

Case 2: $$\sum_{l=m-k+1}^{2m-k} x_l > m-k$$

Then

$$\mathcal{S}_{k,m}\left(x_{m-k+1},\ldots,x_{2m-k}\right)=k-m+\sum_{l=m-k+1}^{2m-k} x_l$$

And

$$\begin{aligned}A&=\mathcal{S}_{k,m}\left(x_1,\ldots,x_{m-k},\ k-m+\sum_{l=m-k+1}^{2m-k} x_l\right)\\&=\max\left(0,\ k-m+\sum_{l=1}^{m-k} x_l+k-m+\sum_{l=m-k+1}^{2m-k} x_l\right)\\&=\max\left(0,\ k-(2m-k)+\sum_{l=1}^{2m-k} x_l\right)\\&=\mathcal{S}_{k,2m-k}\left(x_1,\ldots,x_{2m-k}\right)\\&=B\end{aligned}$$

□

proof of Proposition 2. If we assume that the homomorphic operations work as expected (this depends only on an appropriate choice for the parameters) then we can simply show that the same computation, but over clear data, yields the right result.
We have:

$$2M\cdot C^{(i)}=2\left(\sum_{l=0}^{\gamma-1}\lfloor\frac{\tau\times\mu_{\gamma-l}}{\nu}\rceil\cdot X^l\right)\cdot\left(\sum_{l=0}^{\gamma-1}\frac{c_{l+1}^{(i)}}{\tau\nu}\cdot X^l\right)$$

Therefore its $(\gamma-1)^{\text{th}}$ coefficient is $2\times\sum_{l=1}^{\gamma}\lfloor\frac{\tau\times\mu_l}{\nu}\rceil\times\frac{c_l^{(i)}}{\tau\nu}$. If τ is high enough, then $\lfloor\frac{\tau\times\mu_l}{\nu}\rceil=\frac{\tau\times\mu_l}{\nu}$ and this means the $(\gamma-1)^{\text{th}}$ coefficient of

$$A_i+\left(\sum_{l=1}^{\gamma}\frac{\mu_l^2}{\nu^2}\right)X^{\gamma-1}-2M\cdot C^{(i)}$$

is

$$\begin{aligned}&\sum_{l=1}^{\gamma}\left(\frac{c_l^{(i)}}{\nu}\right)^2+\sum_{l=1}^{\gamma}\frac{\mu_l^2}{\nu^2}-2\times\sum_{l=1}^{\gamma}\frac{\mu_l c_l^{(i)}}{\nu^2}\\&=\frac{1}{\nu^2}\sum_{l=1}^{\gamma}\mu_l^2-2\mu_l c_l^{(i)}+\left(c_l^{(i)}\right)^2\\&=\frac{1}{\nu^2}\sum_{l=1}^{\gamma}\left(\mu_l-c_l^{(i)}\right)^2\\&=\frac{1}{\nu^2}\left\|\mu-c^{(i)}\right\|_2^2=\frac{1}{\nu^2}\mathbb{d}_i^2\end{aligned}$$

Therefore the $(\gamma-1)^{\text{th}}$ coefficient of

$$\begin{aligned}&A_i+\cancel{\left(\sum_{l=1}^{\gamma}\frac{\mu_l^2}{\nu^2}\right)X^{\gamma-1}}-2M\cdot C^{(i)}\\&-\left[A_j+\cancel{\left(\sum_{l=1}^{\gamma}\frac{\mu_l^2}{\nu^2}\right)X^{\gamma-1}}-2M\cdot C^{(j)}\right]\\&=A_i-A_j+2M\cdot\left(C^{(j)}-C^{(i)}\right)\end{aligned}$$

is $\frac{1}{\nu^2}(\mathbb{d}_i^2-\mathbb{d}_j^2)$. □

B Details on the classification process

In this section, we present our classification process in detail. This classification is made in the cleartext space.

Breast Cancer dataset. The data is first preprocessed by rescaling every attribute to a value between 0 and 1. More complex classification algorithms could choose to rescale some attributes differently to increase or decrease their weight in the distance computation. However, most of the previous work applying a k-NN algorithm applies this simple pre-processing (see [28]). We simply apply the same pre-processing. As seen section 4.5.2 our model vector and source vector values are both rescaled using the same factor ν and the source vector is multiplied by a factor τ and then rounded.

At this point a classification rate is obtained through a classification process which we took from the existing literature (see [28] for instance). A few things change in our process compared with the literature's usual rules of thumb. We present our reasoning below. Here is our classification process:

- 1. Select 459 instances at the start to be the testing set among the 569.
- 2. Among the 110 instances left, select randomly 10 instances to be the training set. The 100 other one are the validation set.
- 3. Classify the 100 instances from the validation set by finding their k closest neighbours among the training set instances with k an odd number. The majority class among the neighbours is selected.
- 4. Compute the classification rate (the number of correct classifications divided by 100).
- 5. Repeat steps 2 to 4 a certain amount of times and keep the training set that yields the maximum classification rate on the validation set.
- 6. Compute the classification rate of the testing set using the selected best training set. This is what we actually call the classification rate.

The use of a training, validation and testing set is a standard machine learning method. In this context, one aims to optimize one's classification rate on the validation data by changing every parameter under their control (here, the size and content of the training set and the value k). Then the classification rate of the testing set is computed once.

A rule of thumb in machine learning is to have the following ratio: 64% training set; 16% validation set; 20% testing set. This ratio is not the one we choose to apply: a very small (in absolute value) testing set could not have yielded a very trustworthy result. Additionally, a small validation set (again, in absolute value) limits the quality of the classifier: if it is too easy to obtain a 100% classification rate over the validation set, then there is less competition among the possible model vectors. In comparison, our ratio is the following: 2% training set; 18% validation set; 80% testing set. With this ratio, we achieve classification rates comparable to the state-of-the-art results on this database using the k-NN classification method. It allows us to reduce the time a single classification takes in the encrypted space, and, by increasing the size of the testing set, to make our results more stable. For the purpose of comparison, when running *exactly* the same training algorithm, but with the usual set ratios given above, we obtain a clear classification rate of 94.7% (with $d = 364$ model vectors). The clear classification rate we obtain with our modified ratios is 95.2% (with $d = 10$ model vectors).

For reproductibility purposes, we include here the identification numbers of all 10 vectors eventually selected as our model vectors: 864018, 857010, 855563, 848406, 857156, 862717, 8510653, 86208, 861598, 852781.

MNIST dataset. The database in this case is pre-processed and therefore we jump directly to the selection of the model vectors phase. We do this in exactly the same manner as with the breast cancer database. Importantly, we used several different sizes for our model to give a better understanding of our algorithm's scaling ability, both in terms of accuracy and in terms of efficiency.