International Journal of Information Security (2023) 22:1605–1619
https://doi.org/10.1007/s10207-023-00708-9

REGULAR CONTRIBUTION

Fast but approximate homomorphic *k*-means based on masking technique

Lorenzo Rovida[1]

Published online: 17 June 2023

Abstract
Nowadays, computing on encrypted data seems to be more practical than a few years ago, thanks to the emergence of new Homomorphic Encryption schemes. In this paper, an algorithm based on Homomorphic Encryption for Arithmetic of Approximate Numbers (Cheon et al., in: Takagi, Peyrin (eds) Advances in cryptology—ASIACRYPT 2017, Springer, Cham, pp 409–437, 2017) (HEAAN, or also CKKS) scheme, that is able to perform a secure *k*-means algorithm which processes encrypted data, has been studied and presented. The performance of the classifier running on encrypted data has been evaluated using a standard *k*-means algorithm that works on plain data as a supervised structure, since the results are obtained by approximated computations. The main point of this paper is to take existent theoretical techniques (for example approximations of $\mathrm{sgn}(x)$), to use them and to observe if they are valid in practical applications. The output of the algorithm is a set of *k* encrypted masks that can be applied to the original dataset in order to obtain different clusters. The setting is a standard client–server one. The workload is heavily server-centric, as the client only has to execute a light masking algorithm at the end of each iteration, which, excluding the decryption, is faster than a plain *k*-means iteration; the main disadvantage concerns the accuracy of the results. Experiments show that the algorithm can be executed fairly quickly: the execution time of the training phase is on the order of seconds, while classification is on the order of tenths of a second.

Keywords Homomorphic Encryption · Secure machine learning · Cryptography · Clustering

1 Introduction

Many applications of data mining techniques are used to exploit information hidden in data. This happens frequently with users' private data, especially today, where computers, smartphones, IoT devices, etc. accompany users in many aspects of life. This brings many advantages, although the main disadvantage of these techniques is that external services (such as MLaaS) are able to freely access their private data, and this is necessary for them in order to perform computations. Some techniques [24, 26] can be applied in order to prevent information leak, i.e. data transformation, even though some correlations and dependencies between attributes may be somehow accessible after a careful data analysis.

The rise of new Homomorphic Encryption (HE) schemes and standards [6], though, is opening up new possibilities in the field of information security. By using these schemes, servers do need to decrypt data in order to provide services. In 2009, Gentry [16] presented a Fully Homomorphic Encryption (FHE) scheme, even though its main issue was its impracticability. Moving forward, newer schemes and approaches are trying to address this problem, which prevents these techniques from being used in real applications. One of the main topics of this paper is the theme of *practical applicability*. A lot of work has been carried out, and since then, different cryptosystems based on the hard Learning with Errors [30] problem were introduced.

Writing algorithms based on HE schemes is a new and exciting challenge, since everything must be rewritten in terms of additions and multiplications: the only operations allowed. Furthermore, each scheme has its own way of encoding data. There are many works that show how HE can be used to build simple but secure machine learning models [7, 12, 36], including deep neural networks [1, 22, 29]. In this work, Homomorphic Encryption for Arithmetic of Approxi-

✉ Lorenzo Rovida
l.rovida1@campus.unimib.it

[1] Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano – Bicocca, Viale Sarca 336, 20126 Milan, Italy

mate Numbers [8] (HEAAN, also called CKKS), a levelled[1] fully homomorphic scheme, has been used. In particular, the approach presented in this paper is based on SEAL.jl[2], a Julia package that wraps the original Microsoft SEAL [33], available for C++ language. The main challenges in writing algorithms based on CKKS scheme are:

- finding the best way to encode data: it is a matter of fitting the data to the algorithm and to the schema itself, which encodes arrays of complex numbers into polynomials. One aspect to bear in mind is that operations are slot-wise; therefore, a single operation affects all the values in the encrypted array;
- sticking to the limit: since CKKS is a so-called levelled scheme, it is necessary to define in advance how many operations are to be performed. A HE-based algorithm always executes the same number of operations regardless of the input content. It is a good practice to maximise the number of available operations. Choosing the right parameters is, of course, another challenge that is strictly related to the design of the algorithm;
- finding the right approximation for each function evaluation: since only additions and multiplications are allowed, the evaluation of a non-polynomial function is impossible without approximations. Moreover, the number of operations is limited. This paper is based on some previous work [9, 10] that presents some efficient and elegant polynomial approximations of the signum function $\mathrm{sgn}(x)$ on different degrees and precisions;
- comparing data: as data are encrypted, it is not possible to use comparison operators. Considering that clustering (and classification, in general) is highly dependent on comparisons, it is essential to find a good way to solve this problem. The main idea is to evaluate $a > b$ by calculating $\mathrm{sgn}(a - b)$.

The purpose of this study is to present an efficient algorithm that is able to perform k-means training and classification based on Euclidean distance on a set of encrypted points. The training phase takes place through some interactions between server and client. Basically, the server is able to execute a single iteration and return the result. Then, the client processes the result and may request another iteration by sending a new set of centroids. The data handling phase, executed by the client, is faster than a standard k-means execution since, as shown later, each iteration requires a simple masking operation for each cluster; thus, it has a complexity of $\mathcal{O}(k \cdot n)$ where k is the number of clusters; n is the number of points (notice that the k procedures can be parallelised).

[1] Only evaluations of arbitrary circuits of bounded (pre-determined) depth are supported.

[2] https://github.com/JuliaCrypto/SEAL.jl.

On the other hand, the transformation of an existing model into an encrypted one and the classification of an encrypted point are simpler and do not require multiple interactions.
Note that the security of this algorithm is based on the scheme and, moreover, on *how* results are used by the secret key owner. As stated in "Correct use of Microsoft SEAL"[3]: *decryptions of Microsoft SEAL ciphertexts should be treated as private information only available to the secret key owner, as sharing decryptions of ciphertexts may in some cases lead to leaking the secret key*. Sharing the results of this algorithm can be a reason for serious security problems, as shown in Li and Micciancio's attack [18] on CKKS.

1.1 Related works

Secure clustering algorithms have been studied in different works and applications. There are many approaches to clustering based on secure multi-party computation (MPC) [4, 21, 39], although MPC requires the participation of data owners for the initial secret sharing. The approach used in this paper is based on Homomorphic Encryption, which allows to make use of encrypted data, without decrypting it. In particular, the algorithm is built on a client–server model. Since the introduction of Ring-LWE (RLWE) [23], most of the HE-based programs seem to be based either on the Brakerski–Gentry–Vaikuntanathan (BGV) scheme [3], on the Brakerski–Fan–Vercauteren (BFV) scheme [2] on the Cheon–Kim–Kim–Song (CKKS) scheme or on the latest Fully Homomorphic Encryption over the Torus (TFHE) scheme [13]. Currently, one of the most suitable schemes for machine learning applications is CKKS that is able to encrypt array of complex values, and it performs computations using the Single Instruction stream, Multiple Data stream (SIMD) mechanism, making it really suitable for Machine Learning applications. Nevertheless, there are not many proposals of k-means based on CKKS.

There are k-means works based on HE that are constructed over the Pailler's cryptosystem [28], such as [5, 27].

There were proposals for really fast systems based on Liu's cryptosystem [19], such as [20], until this cryptosystem was proved to be insecure [35].

There are few proposals for what concerns actual client–server systems built over RLWE-based schemes; [32] is based on BFV scheme and is able to run each iteration in less than 10 min for 10-dimensional datasets of 100 points. To reduce the burden of the client, the comparison operations (which are initially performed by the client) are completed by an additional entity, a trusted server.

A proposal based on TFHE is given in [17], and it does not require for the client to perform any computation during the whole training process because of its fast bootstrapping that

[3] https://github.com/microsoft/SEAL/blob/main/SECURITY.md.

allows for the number of operations to be almost unlimited. Even though this is the preferred behaviour, this approach is still far from practical since it takes days to run the algorithm.

Perhaps the closest work is given in [11], where it is shown that running a mean-shift clustering algorithm is becoming practical, using CKKS scheme.

Lastly, a general overview is provided in [37], even though every k-means work is based on different architectures and schemes (MPC, trusted servers, updatable distance matrix, and so on).

This work is based on the Ring-LWE problem, uses the CKKS scheme with Euclidean distance calculation, and proposes a traditional client–server model , where the server is assumed to be honest but curious and the client workload is considered to be really light. To the best of our knowledge, there is not any k-means implementation based on this setting, as all the other works use trusted servers or ask the client to perform some computations, especially when assigning each point to a cluster. Implementations based on other schemes, presented before, are limited in terms of execution times. This works focuses on executing the algorithm really fast. This costs in terms of precision, in fact the results are not 100% accurate with respect to a plain k-means execution. A quick comparison is carried out in Table 1.

Our contribution is to present an initial implementation of k-means and a different approach to homomorphic computing, proposing a method that, using existing theoretical results (especially [9]), is able to run each iteration very fast by renouncing precision in computations , and also to exclude other (even trusted) servers from the dialogue.

1.2 Work organisation

The paper is organised as follows: in Sect. 2, the design of the main algorithm is explained and justified. Section 3 presents the analysis of several scenarios; in particular, errors and computational times relative to some experiments conducted on various UCI Machine Learning [14] datasets are shown and discussed. Finally, in Sect. 4, all the obtained results are considered and conclusions are drawn.

2 Algorithm design

Within the context of this research, thoroughly understanding the inner workings of CKKS is not essential (although it might be useful to better understand some implementation choices). Let N be the polynomial modulus degree in which data is encoded and encrypted. For a fixed value of scale Δ (which affects precision in calculations), this value determines two important things:

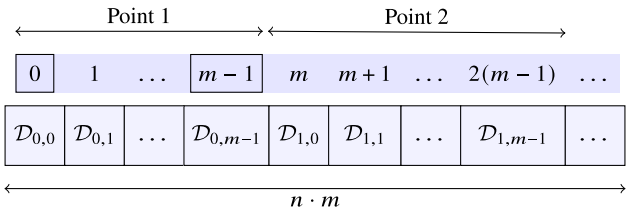


Fig. 1 Reshaping $\mathcal{D}$ into an array of size $n \cdot m$

- the depth of the circuit: it determines the upper bound of the number of multiplications that can be performed on a single ciphertext;
- the size of the array to be encoded and encrypted: CKKS takes as input an array $z \in \mathbb{C}^{N/2}$.

Let $\mathcal{D}$ be a numerical dataset composed of m columns and n rows. The number of dimensions m must not be greater than the number of available *slots* per ciphertext ($m \leq N/2$). As a pre-processing step, all columns of the dataset $\mathcal{D}$ must be transformed into $[0, \sqrt{m}/m]$. This is done to "set up" the points; it is shown later, after Fig. 7, why this is necessary. Given a dataset $\mathcal{D}$, finding an optimal way to *shape* it before encoding and encrypting it into a polynomial is the first objective. At the moment, it is still unknown how many multiplications are necessary for the execution of the algorithm, suppose 2. This means that for a fixed level of precision (suppose 40 bits, as is done in Microsoft SEAL examples), by setting, for instance, $N = 2^{13}$, the slots available for any ciphertext will be $2^{13-1} = 4096$. By encrypting each row of $\mathcal{D}$ into a ciphertext, if m is less than 4096 (most likely true), SEAL will automatically add a padding of zeros, resulting in not-optimised calculations. The approach proposed in this section maximises the use of available slots by reshaping $\mathcal{D}$ into a single array of length $n \cdot m$ and, then, by encoding and encrypting the array into $\lceil nm/s \rceil$ ciphertexts, where s is equal to the number of available slots in each ciphertext; therefore, $s = N/2$ (Fig. 1). This is done in order to take advantage of the SIMD property of CKKS in order to speed up calculations.

Definition 1 Given the number of slots available in a ciphertext $s = N/2$ and the number of total values in a dataset $n \cdot m$ (assuming that $s \geq m$), the efficiency $\mathcal{E}$ for any encoding process is defined as the ratio of the number of slots used to the total number of slots.

$$\mathcal{E} = \frac{s \cdot \lfloor \frac{nm}{s} \rfloor + nm \bmod s}{s \cdot \lceil \frac{nm}{s} \rceil} \leq 1 \tag{1}$$

Table 1 Comparing different k-means algorithms based on HE. D stands for days, H for hours, M for minutes, S for seconds

Reference	Scheme	Encoding	Client workload	Runtime
[34]	BFV	Integer	Heavy	H
[32]	BGV	Integer	Trustable server	M
[17]	TFHE	Binary	No workload	D
[5]	Pailler	Integer	Heavy	H
Our proposal	CKKS	Real	Light	S

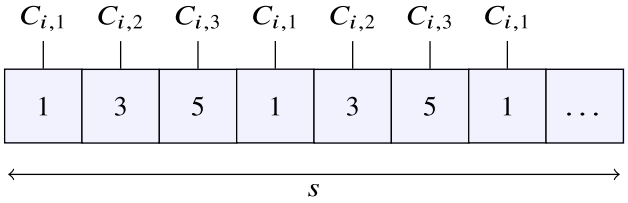


Fig. 2 Encoding the i-th centroid C_i into a an array of length $s - s \bmod m$

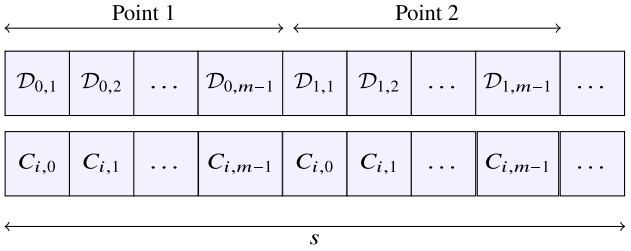


Fig. 3 Aligning C_i below the encoded dataset

The best and worst cases are defined as follows.

- best case: $nm = 0 \bmod s$. Since $nm = 0 \bmod s \implies s \mid nm \implies \frac{nm}{s} \in \mathbb{Z} \implies \lfloor \frac{nm}{s} \rfloor = \lceil \frac{nm}{s} \rceil$, meaning that $\mathcal{E} = 1$;
- worst case: $nm = 1 \bmod s$. In this case, the last ciphertext will only contain a single value and $s - 1$ zeros. The worst possible efficiency value is given by the fact that the difference $\lceil \frac{nm}{s} \rceil - \lfloor \frac{nm}{s} \rfloor$ is the highest possible.

If $s \nmid m$, the coordinates of points will occupy $s - (s \bmod m)$ slots in each ciphertext (the remaining $s \bmod m$ slots will contain zeroes). This may be seen as a waste of space, but by "splitting" a point into two different ciphertexts it would be impossible to execute Algorithm 1 that is shown later.

After encoding $\mathcal{D}$, the set of centroids C (that are chosen by the client, in case of first iteration they will probably be chosen randomly) must be encoded cleverly. A way of encoding that optimises time and computational complexity is the following: given the i-th centroid C_i, repeat its coordinates in an array of length $s - (s \bmod m)$. It is possible to define each position j of the final ciphertext c_i as follows:

$$c_i[j] = C_i[j \bmod m] \tag{2}$$

For instance, given a centroid $C_i = \{1, 3, 5\}$ with $m = |C_i| = 3$, a representation of the pre-processed centroid is given in Fig. 2.

As for the ciphertext that contains the dataset, if $s \nmid m$, the last $s \bmod m$ slots will contain zeroes.

In the next subsections, the reasons why points and centroids have been encoded this way will become clearer. Foremost, it is worth taking a look at the main steps of the k-means algorithm:

1. calculate the distances between each point p in $\mathcal{D}$ and each centroid c_i, with $i \in \{1, 2, \ldots, k\}$;
2. assign each point p to the nearest centroid by finding the minimum distance;
3. compute new centroids using the newly formed clusters.

Each of these three phases has been analysed, and it will be shown how each step has been rewritten.

2.1 Distance calculation

The idea is to create an array of size k (k is equal to the number of clusters) whose i-th element contains a ciphertext representing the distance between each point of $\mathcal{D}$ and the i-th centroid. Now it is shown how to calculate an entry of this array, assuming that $\mathcal{D}$ is encoded in a single ciphertext for simplicity, although the procedure is extensible to any dimension of $\mathcal{D}$ (by repeating the process $\lceil nm/s \rceil$ times: one for each ciphertext).

Remark 1 Operations between ciphertexts are *slot-wise*. For example, given $c_1 = [1, 2]$ and $c_2 = [3, 4]$, it holds that $c_1 + c_2 = [4, 6]$, and $c_1 \cdot c_2 = [3, 8]$.

By aligning each centroid C_i below the ciphertext that contains $\mathcal{D}$ (encoded as in Fig. 1), a situation like the one presented in Fig. 3 is obtained.

It is possible to subtract these two ciphertexts and square the result, obtaining the values presented in Fig. 4 (notice that one multiplication, that is squaring, has been performed).

Next, it is possible to take advantage of a very useful operation available in CKKS, i.e. rotation. It gives the possibility of rotating (or shifting to the left) all the values contained in a ciphertext by x positions. It is possible to rotate rot times

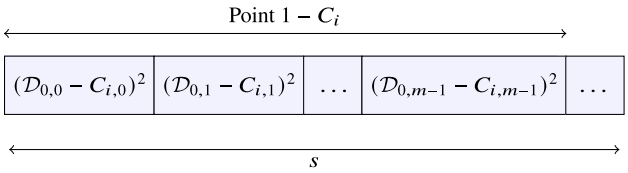


Fig. 4 Squared distance between each coordinate of points from dataset $\mathcal{D}$ and C_i

(Eq. 3) a ciphertext like the one presented in Fig. 4 to obtain the sum of the m distances between coordinates for each point (Algorithm 1).

$$rot = \lfloor \log_2(m) \rfloor + (m - 2^{\lfloor \log(m) \rfloor}) \tag{3}$$

Algorithm 1 Sum of the first m slots

Input: Ciphertext c, number of dimensions m
Output: Ciphertext c containing the sum of the first m slots

```
i ← 1
j ← 1
result ← c
while i ≤ ⌊log2(m)⌋ do
    r ← rotate(c, i)                ▷ Rotates values to the left
    result += r
    i ← 2i
end while
while 2^i + j ≤ m do
    r ← rotate(c, 2^i + j)
    result += r
    j ← j + 1
end while
return result
```

The best case is when m is a power of 2 (the second term of Eq. 3 becomes 0), whereas the worst is when m is a power of 2 minus 1. A visual simulation of the algorithm is given in Fig. 5.

After the execution, positions $x : x = 0 \bmod m$ will contain the sum of the m squared differences between the coordinates of the x-th point and the i-th centroid, that is, the squared Euclidean distance! For instance, position 0 will contain:

$$(a_{0,1} - c_{i,1})^2 + (a_{0,2} - c_{i,2})^2 + ... + (a_{0,m} - c_{i,m})^2$$

while position m will contain:

$$(a_{1,1} - c_{i,1})^2 + (a_{1,2} - c_{i,2})^2 + ... + (a_{1,m} - c_{i,m})^2$$

And so forth. By repeating this procedure for each centroid C_i, the final output will be an "array" that can ideally be represented as shown in Fig. 6 (each row is split into columns

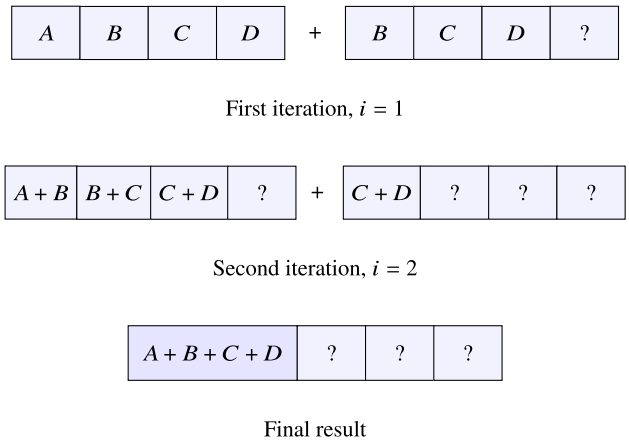


Fig. 5 Simulation of Algorithm 1 on $m = 4$

	p_1	–	p_2	–
c_1:	$\text{dist}(p_1, C_1)$	–	$\text{dist}(p_2, C_1)$	–
c_2:	$\text{dist}(p_1, C_2)$	–	$\text{dist}(p_2, C_2)$	–
	⋮	–	⋮	–
c_k:	$\text{dist}(p_1, C_k)$	–	$\text{dist}(p_2, C_k)$	–

Fig. 6 A distances matrix

representing the slots of the i-th ciphertext). Empty columns (–) represent sets of different slots in positions $p : p \neq 0 \bmod m$ that contain partial computations, which are useless in this context.

Remark 2 As someone said, bits are not coloured; the results of Algorithm 1 are only found in positions $j \cdot m$ (where $j \in \{0, 1, \ldots, n-1\}$). Other positions will contain partial calculations that should not be taken into account.

The next step would be to find the square root of these values in order to obtain the final Euclidean distance. However, this is not the case. In fact, the information required is not the *actual* values of the Euclidean distances, but the order relationships between them. In other words, the aim is not about knowing the values of d_1 and d_2, but to know whether $d_1 > d_2$.

Proposition 1 *Given $a, b \in \mathbb{Z}$, assuming that $a \geq b > 0$, it holds that:*

$$a > b \longleftrightarrow \sqrt{a} > \sqrt{b} \tag{4}$$

Proof Let a, b be two of the just calculated distances (that are positive, since are sum of squares) and assume that $a \geq b$. Therefore, $a - b \geq 0$, meaning that $(\sqrt{a}+\sqrt{b})(\sqrt{a}-\sqrt{b}) \geq 0$. Now, it is possible to divide both sides by $(\sqrt{a}+\sqrt{b})$ and obtain $\sqrt{a} - \sqrt{b} \geq 0$, or, equivalently $\sqrt{a} \geq \sqrt{b}$. □

Proposition 1 justifies the choice of using squared distances during the comparison phase, as they provide the same results as comparing their square roots (with lighter computations).

At this point, the objective is to find the minimum distance in each column of the distance matrix; it is necessary to assign each point to a cluster.

2.2 Finding the minimum

The aim is to find the minimum value in each column p_i (Fig. 6), and it must be done without using comparison operators. As mentioned in Introduction, the key is to use $\text{sgn}(x)$ function. The idea is to rely on a function like the one presented in Eq. 5 which, indeed, may be seen as $(\text{sgn}(b-a)+1)/2$.

$$f(a,b) = \begin{cases} 1 \text{ if } a < b \\ \frac{1}{2} \text{ if } a = b \\ 0 \text{ if } a > b \end{cases} \tag{5}$$

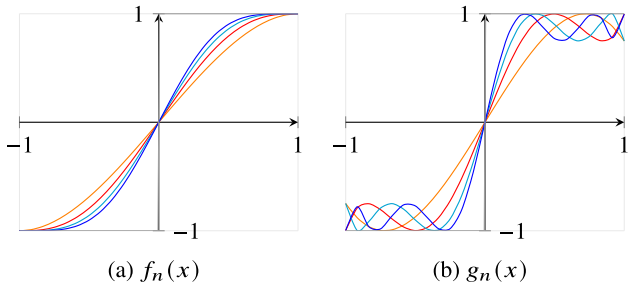
Subsequently, it would be possible to use that function on each couple $(\text{dist}(p_i, c_j),\ \text{dist}(p_i, c_z))$ in order to generate k masks that could be used to obtain the clusters. For instance, by masking $\mathcal{D}$ with the first mask, the resulting subset of $\mathcal{D}$ will only contain points that are closer to the first centroid than the others. Equation 5 returns 1 with these input points, because the distance to the first centroid is the smallest, 0 with the other points: it effectively behaves like a mask. It is clear, however, that this function is not polynomial; thus, it is not possible to reproduce it flawlessly through HE computations.

It is possible, nevertheless, to use approximations. This topic is well discussed in [9, 10]; hence, in order to build an approximate version of Eq. 5, Cheon et al. work has been used.

In the case of $k = 2$, in order to assign a point to a cluster, calculating a ciphertext that has been called, for the sake of clarity, C_1 vs C_2, is enough. This ciphertext is nothing more than the application of a function, like the one presented in Eq. 5 (but approximated), on a and b, where a is the first row of the distances matrix (Fig. 6) and b is the second one. In other words, for each point, the procedure is to check whether the distance to the first centroid is smaller than the distance to the second one.

Notice that, since operations are slot-wise, by comparing two ciphertexts, all the distances stored into the first ciphertext, with the ones stored into the second ciphertext, are compared. The slots in position $j \cdot m$, with $j \in 1, 2, \ldots, k$ (Remark 2), in the resulting ciphertext will contain ≈ 1 if the j-th point is assigned to the first cluster, ≈ 0 if it is assigned to the second one.

Definition 2 Given two ciphertexts c_1 and c_2 containing two sequences of distances in positions $j \cdot m$ (where $j = \{0, 1, \ldots, n-1\}$, with n being the number of rows of the original dataset), a comparison function is defined as:



(a) $f_n(x)$ (b) $g_n(x)$

Fig. 7 Families of polynomial approximations of sign function, recalled from [9]

$$f_{comp}(c_1, c_2) = \frac{\text{sgn}'(c_1 - c_2) + 1}{2} \tag{6}$$

where the $\text{sgn}'(x)$ function is a polynomial approximation of $\text{sgn}(x)$. The result of this operation is **approximately** a binary mask. As usual, the actual results are found in positions $j \cdot m$.

Example 1 Given $c_1 = [4, 1, 8]$ and $c_2 = [2, 10, 2]$:

$$f_{comp}(c_1, c_2) \approx [0, 1, 0]$$

The next objective is to define $\text{sgn}'(x)$: this is a polynomial approximation of the function $\text{sgn}(x)$. I will recall two families of functions from [9] (Fig. 7).

These approximations are the reason why, at the beginning of Sect. 2, all the values in each column of $\mathcal{D}$ have been transformed into $[0, \sqrt{m}/m]$. By doing that, the maximum squared Euclidean distance d between two points is always in $[0, 1]$.[4] This happens because the diagonal (that is, the maximum Euclidean distance between two points in space) of a m-dimensional hypercube is equal to $\ell\sqrt{m}$. Since in this case the distances are squared, the maximum possible value of distance between two points is equal to $(\ell\sqrt{m})^2$. In order for $\text{sgn}'(x)$ to work, without loss of generality, for any couple of points, it must hold that Eq. 7 is less than or equal than 1 (and greater than or equal to 0, but it is trivial).

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 + \ldots \tag{7}$$

In particular, it reaches its maximum value when calculating the distance between two diagonally opposite points. Considering the i-th column of the dataset, max_i is defined as the maximum value in that column, while min_i as the minimum. The range $max_i - min_i = \ell_i$ is the i-th side of the hypercube. By transforming all the values in each

[4] As a consequence, the maximum difference between two distances will always be in $[-1, 1]$.

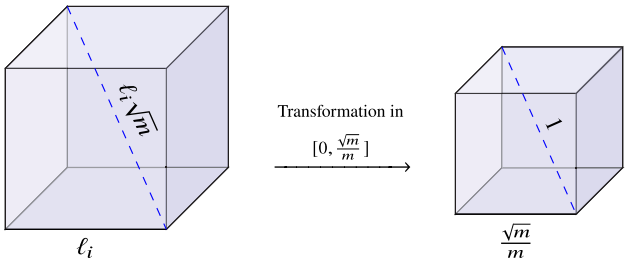


Fig. 8 Column transformation makes the magnitude of the diagonal of a hypercube equal to the unit; this happens because $\sqrt{m}/m \cdot \sqrt{m} = 1$. Notice that in this case $m = 3$

dimension from $[min_i, max_i]$ to $[0, \sqrt{m}/m]$, the maximum possible value of Eq. 7 becomes:

$$\left(0 - \frac{\sqrt{m}}{m}\right)^2 + \left(0 - \frac{\sqrt{m}}{m}\right)^2 + \left(0 - \frac{\sqrt{m}}{m}\right)^2 + \ldots$$
$$= \sum_{i=1}^{m} \left(0 - \frac{\sqrt{m}}{m}\right)^2 = m\left(\frac{1}{m}\right) = 1 \tag{8}$$

The transformation is really easy, just normalise in $[0, 1]$, then multiply by $\sqrt{m}/m$. For each x in the i-th column, x becomes:

$$x = \frac{x - min_i}{max_i - min_i} \cdot \frac{\sqrt{m}}{m} \tag{9}$$

Equation 9 is very important, because it highlights how max_i, min_i and m define the transformation. The larger m, the smaller the new points. The bigger the difference between max_i and min_i (i.e. the range ℓ_i), the smaller the new points. The smaller the new points, the less accurate the calculations (because the accuracy of $\text{sgn}(x)$ approximations is lower with tiny values, it is simple to notice it in Fig. 7). In other words, the accuracy of the algorithm depends on the range of each column of $\mathcal{D}$ (first term) and on the number of dimensions (second term). By enlarging the range of $\text{sgn}'(x)$ from $[-1, 1]$ to some wider interval, it would be possible to have more accurate calculations, as the values in the dataset could be transformed into larger intervals.

Now, the idea is to use two distances d_1, d_2 as input for a comparison function (Definition 2). A function such as those presented in Fig. 7 (which has previously been referred to as $\text{sgn}'(x)$) is called on the difference $d_1 - d_2$: if this value is negative, $\text{sgn}'(x)$ returns ≈ -1, which means that Eq. 6 returns ≈ 0. On the other hand, if that difference is positive, the result of Eq. 6 is ≈ 1.

In the case of $k = 2$, assigning a point to a cluster is relatively easy, as it is sufficient to calculate C_1 vs C_2, but what happens with $k > 2$? For a point p to be assigned to the i-th cluster, the distance between p and C_i should be the smallest of all distances. It is possible to solve this problem by using a comparison matrix (Fig. 9).

C_1 vs C_1	C_1 vs C_2	...	C_1 vs C_k
C_2 vs C_1	C_2 vs C_2	...	C_2 vs C_k
⋮	...	⋱	⋮
C_k vs C_1	C_k vs C_2	...	C_k vs C_k

Fig. 9 A comparison matrix

1	C_1 vs C_2	...	C_1 vs C_k
1 − (C_1 vs C_2)	1	...	C_2 vs C_k
⋮	...	⋱	⋮
1 - (C_1 vs C_k)	1 - (C_2 vs C_k)	...	1

Fig. 10 Optimised comparison matrix

Each cell in this matrix is defined as follows:

$$\text{distmat}_{ij} = C_i \text{ vs } C_j = f_{comp}(c_i, c_j) = \frac{\text{sgn}'(c_i - c_j) + 1}{2} \tag{10}$$

where each ciphertext c_i represents the i-th row of the distance matrix (Fig. 6). Using the comparison matrix, it is possible to obtain the mask associated with each cluster. By multiplying all the ciphertexts in the first row, for instance, it is possible to get the first mask. The $(j \cdot m)$-th slot of a mask (with $j \in \{0, 1, 2, \ldots, (n-1)\}$) will contain:

- ≈ 1: if p_j is closer to the relative centroid than *all* the other centroids.
- ≈ 0: if there is *at least* one other centroid whose distance from p_j is smaller.

Proposition 2 *C_i vs C_j and C_j vs C_i, for any $i \neq j$, are strongly correlated; in fact, it holds that:*

$$C_i \text{ vs } C_j = 1 - (C_j \text{ vs } C_i) \tag{11}$$

Using Proposition 2, the number of comparisons to be calculated in the comparison matrix is reducible from $k(k-1)$ to $k(k-1)/2$. This is half the number of off-diagonal coefficients in a $k \times k$ matrix. Some optimisations can be applied to the matrix using these considerations: rewriting the lower triangular part as a function of the upper one and writing ones on the main diagonal (comparing the same centroid does not make sense).

By checking Table 1, it is easy to deduce that the only part that needs to be calculated is the upper triangular part.

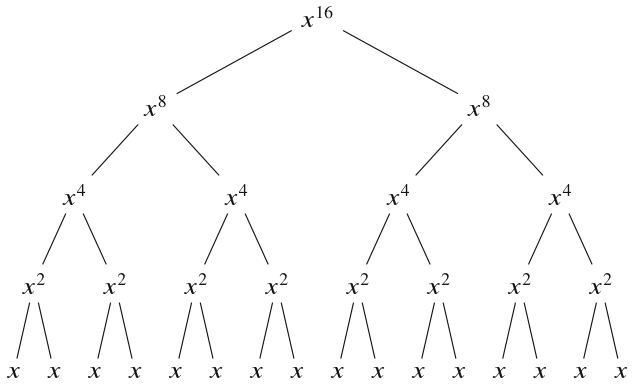


Fig. 11 Products tree for x^{16}

Proposition 3 *Time and space complexity for the comparison phase is equal to* $\mathcal{O}\left(\frac{k(k-1)}{2}\right)$.

Considering that triangular part, since the result of each cell does not depend on any other one, it is possible to exploit multithreading. Every C_i vs C_j has been saved into a ThreadSafeDict[5] in order to parallelise computations and reduce time. This procedure, however, has a flaw: the number of necessary multiplications increases with k. This is an open issue; for the moment, it is possible to reduce this relationship from linear to logarithmic by evaluating x^n as a tree. Normally x^n needs $n-1$ multiplications to be evaluated, but notice that:

$$x^n = (x^{\frac{n}{2}} \cdot x^{\frac{n}{2}}) = (x^{\frac{n}{4}} \cdot x^{\frac{n}{4}}) \cdot (x^{\frac{n}{4}} \cdot x^{\frac{n}{4}}) = \dots \tag{12}$$

In general, the number of multiplications can be decreased from $n-1$ to $\lceil \log(n) \rceil$. This reasoning is well represented in Fig. 11.

Of course, the total number of products to be executed is the same, but the final mask will be the result of a total of $\lceil \log(n) \rceil$ operations. As a consequence, large values of k will inevitably reduce the accuracy of comparisons, since it is necessary to reduce the degree of $\text{sgn}'(c_i, c_j)$ because the evaluation of a deeper tree requires more multiplications (remember that the number of multiplications is fixed). An alternative method could be to add up all the masks and then to ask the user to find the largest values for each slot. This method would solve the problem of the increasing number of multiplications and allow $\text{sgn}'(x)$ to be a better approximation. One of the points of this research, though, is to entrust most of the computations to the server, and this solution would violate this rule; therefore, it will be ignored.

The output of this second phase is a set of masks that the client can apply to $\mathcal{D}$ in order to obtain different clusters.

[5] https://github.com/wherrera10/ThreadSafeDicts.jl.

2.3 New centroids calculation

It is easy for the client to calculate new centroids, for each received mask msk_i:

1. compute $msk_i \cdot \mathcal{D}$ to obtain the i-th cluster (remembering to consider only positions $p : p = 0 \bmod m$, as discussed before);
2. add up all the points in it and divide by the number of points, that is, the number of ones in msk_i in positions $p : p = 0 \bmod m$, in order to get the new centroids.

It has been studied later, in the next section, whether for the client is more convenient to execute a plain k-means iteration from scratch or to run this algorithm.

3 Experimentation

In Sect. 2, several ideas behind the homomorphic k-means have been defined. Algorithm 2 presents a simple overview of the execution flow.

Algorithm 2 Homomorphic k-means server iteration

Input: Dataset $\mathcal{D}$, set of k centroids
Output: Set of k masks
1: Calculate distances matrix
2: Calculate comparison matrix
3: Evaluate k products trees
4: **return** masks

Algorithm 3 Homomorphic k-means client iteration

Input: Set of k masks
Output: Set of k centroids
1: Decrypt k masks
2: **for each** mask $\in$ masks **do**
3: $c_{mask} \leftarrow$ mask $\cdot \mathcal{D}$
4: $c_{mask} \leftarrow c_{mask} / \#points$
5: **end for**
6: **return** masks

The next subsections define three different scenarios: a fully homomorphic training, a conversion of an existing plain model to an encrypted one, and the classification of an encrypted point.

3.1 Homomorphic training

Training a model is done by running `niter` times Algorithm 2 over the server, and Algorithm 3 on the client. In particular, traffic between server and client can be represented as shown in Fig. 12.

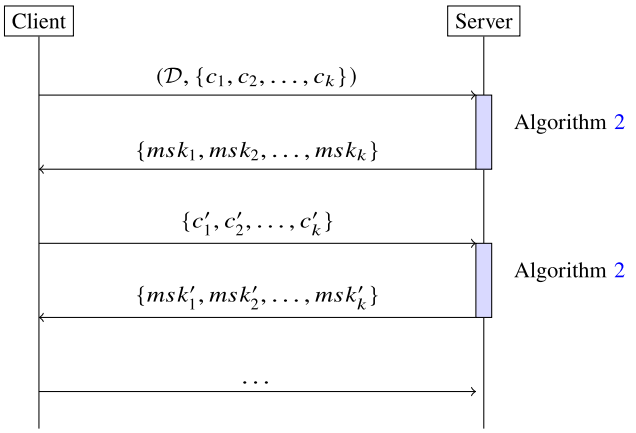


Fig. 12 Traffic between server and client during homomorphic training

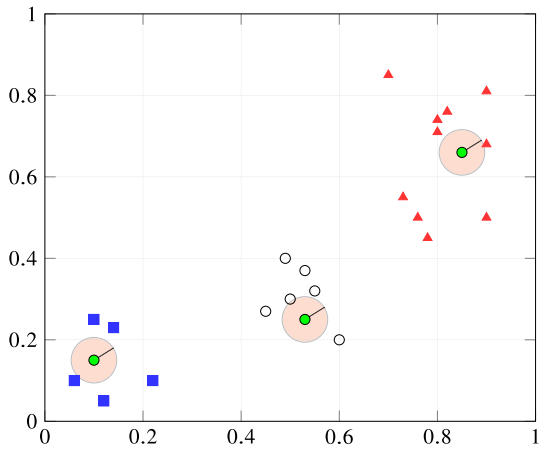


Fig. 13 Centroids error interpretation; in this case, the error (or the radius of red circles) is equal to 0.05 (5%). The green points are plain centroids $\in C$, whereas the encrypted ones lie inside the red areas

Notice that the dataset is sent only once, at the beginning of the dialogue (the server can store it). Subsequent iterations will only require a set of centroids that will most likely result in really light communications. Notice that since masks are *almost* binary (they are nothing but ciphertexts, that decrypted and decoded are arrays of complex numbers), their values heavily depend on the precision of $\mathrm{sgn}'(x)$; therefore, there will be differences between centroids calculated via homomorphic computations and the ones returned from a plain iteration.

These differences change according to different factors (dimensionality, number of clusters, precision of $\mathrm{sgn}'(x)$), as it has been explained after introducing Eq. 9. Figure 13 presents a visual interpretation of errors made by a homomorphic k-means algorithm in respect to a standard one.

Two versions of homomorphic k-means have been implemented: a high speed version and a high-precision one. The next objective is to determine whether these two versions are able to emulate efficiently a plain k-means algorithm. The main difference between the two versions is the value of N, that is, the polynomial modulus degree in which data are encoded. This, in turn, determines the precision of $\mathrm{sgn}'(x)$, because larger values of N allow the use of higher degree approximations, whereas smaller values do not. To find the optimal composition of the functions $f_n(x)$ and $g_n(x)$ (from [9]) that better approximate $\mathrm{sgn}(x)$ for a fixed number of multiplications x, different compositions have been tested, and from those that had a maximum degree of $2x$ (with x that depends on the algorithm version, high precision is capable of evaluating functions of a higher degree than the high-speed version) it has been chosen the one that minimised:

- the value of $|1 - \int_0^1 \mathrm{sgn}'(x)\,dx|$;
- the values of $1 - \mathrm{sgn}'(x)$ in different $x \in \{0.001, 0.01, 0.1\}$.

Table 2 Parameter sets for both versions

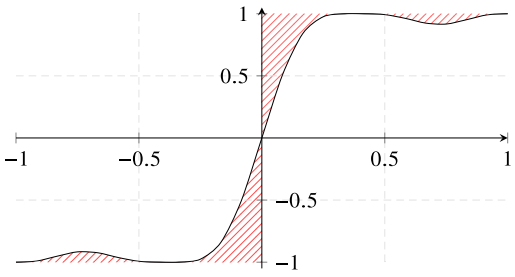
	High speed	High precision
N	2^{14}	2^{15}
Scale (Δ)	2^{27}	2^{40}
Intermediate primes bits	27	40
Circuit depth	10	19
$\mathrm{sgn}(x)$ degree	12 (Fig. 14a)	20 (Fig. 14b)
$\lvert 1 - \int_0^1 \mathrm{sgn}'(x)\,dx \rvert$	≈ 0.1161	≈ 0.0374

For both versions, the number of security bits is always 128, as specified by HE standards [6]. More details are given in Table 2. The two functions $\mathrm{sgn}'(x)$ used are presented in Fig. 14.

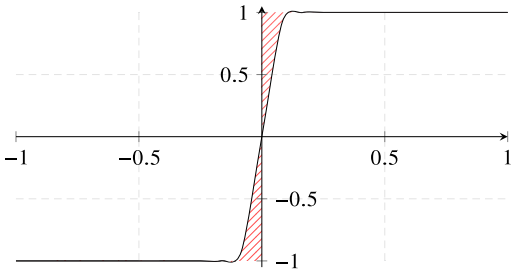
An example of how Euclidean distances between plain and encrypted centroids change, during each iteration, in both versions of homomorphic k-means, on Iris dataset [15] ($k = 4$, $n = 150$, and $m = 4$), along eight iterations, is presented in Fig. 15.

Each line represents the Euclidean distance between plain and encrypted centroids in two different scenarios: Fig. 15a shows the behaviour of the high-precision algorithm, whereas Fig. 15b shows the behaviour of the high-speed one. The maximum possible distance between two points, and therefore the biggest error, is always 1 (because of the transformation explained in Fig. 8). The bigger error in this experiment is $0.078 \approx 7.8\%$, while the least is $0.0002 \approx 0.02\%$.

It is easy to notice that the distances in high-speed variant are larger, and it is not surprising. These plots are useful to examine how, during each iteration, distances become closer, but they reach a point from which the algorithm does not improve, but it somehow stabilises. Table 3 presents the results relative to ten different combinations of initial centroids.

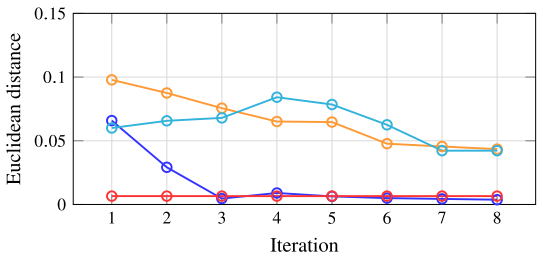


(a) High speed $(g_1(x) \circ g_1(x) \circ f_1(x))$

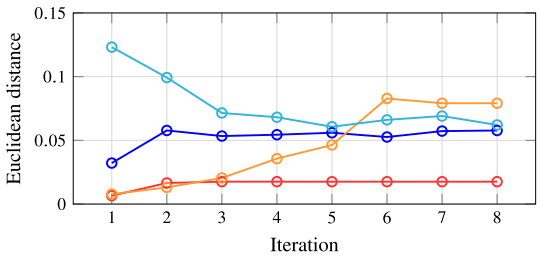


(b) High precision $(g_1(x) \circ g_1(x) \circ g_1(x) \circ g_1(x) \circ f_1(x))$

Fig. 14 Optimal $\text{sgn}(x)$ approximations for both versions, built using the definitions of $f_n(x)$ and $g_n(x)$, from [9]



(a) High precision version



(b) High speed version

Fig. 15 Euclidean distance between centroids calculated on plain Iris dataset and on encrypted Iris dataset during eight iterations

Table 3 Distances between encrypted centroids and plain ones on Iris dataset

	Mean	Mean (%)	Median	Variance (σ^2)
High speed	0.0721	7.21%	0.0711	0.0141
High precision	0.0295	2.95%	0.0284	0.0046

The data are for the last of ten iterations and on ten executions with different random centroids

Table 4 Single homomorphic k-means training iteration computational time on Iris dataset, calculated by BenchmarkTools.jl 1.3.1

	Minimum	Median	Average	Maximum
High speed	1.712 s	1.730 s	1.746 s	1.791 s
High precision	5.987 s	6.511 s	6.718 s	8.113 s

As for the high-precision algorithm, the mean percentage distance (or error) value is about 3%, which means that, on average, the resulting centroids after 10 iterations of k-means are a little biased. Now, the interpretation of these values most of the time depends on the context, but it is a good starting point. The high-speed version performs worse, as expected; its errors are too large (more than double) compared to the other version.

Of course, on the other hand, the high-speed algorithm has lighter computational time and space. The following values in Table 4 have been calculated on a standard desktop PC (AMD Ryzen 3600 with 32 GB of RAM) using SEAL.jl 0.4.0 on Julia 1.5.

The high-precision version is almost four times slower, but its errors are usually more than twice as small. Unfortunately, it is not possible to run the high-speed version several times to get better results, because it gets stuck in situations where comparing two distances is difficult. For instance, if the value of a distance x is smaller than, suppose 0.001, low-degree functions $\text{sgn}'(x)$ are not able to achieve a decent result; ergo, the computed value remains close to the middle (0.5), which means that the point will somehow be half-assigned to both clusters. This situation happens less in the high-precision version (less, but it may still happen).

Table 5 is calculated by comparing the standard k-means and the homomorphic high-precision k-means on some UCI Machine Learning repository [14] datasets. Surprisingly, the errors seem to be smaller when k is relatively big. This may happen as a side effect during the evaluation of the comparison matrix. The larger this matrix, the greater the possibilities of obtaining better results since the masks are produced by more comparisons; in other words, it may happen that the comparisons compensate for each other. On the other hand, with a smaller k, a single comparison seems to lead to more errors.

With regard to execution times with different values of k, the high-speed version is characterised, not surprisingly, by decidedly lower values, as confirmed in Fig. 16, that presents computational times on different values of k on Iris dataset.

These two plots make sense as the comparison matrix is characterised by a complexity of $\mathcal{O}(\frac{k(k-1)}{2})$ (Proposition 3).

3.2 Bandwidth requirements

An aspect not to be underestimated is bandwidth. Outsourcing of computations is useful, especially on devices with

Table 5 High-precision homomorphic k-means performance on various UCI Machine Learning dataset

Dataset name	k	n	m	Mean error (± std. dev.)	Mean iteration time (± std. dev.)
Iris	4	150	4	0.0293 ± 0.0028	$6.724\text{s} \pm 0.321\text{s}$
Wine	3	178	12	0.0631 ± 0.0031	$8.024\text{s} \pm 0.502\text{s}$
Breast Cancer (Diagnostic)	2	569	30	0.0685 ± 0.0041	$7.441\text{s} \pm 0.571\text{s}$
Absenteeism at work [25]	2	740	20	0.0910 ± 0.0011	$9.451\text{s} \pm 0.460\text{s}$
Seeds	3	210	6	0.0566 ± 0.0023	$6.944\text{s} \pm 0.290\text{s}$
Transfusion [38]	3	748	4	0.0633 ± 0.0021	$7.345\text{s} \pm 0.367\text{s}$
Banknote Authentication	2	1372	4	0.0633 ± 0.0021	$6.784\text{s} \pm 0.288\text{s}$
Tripadvisor Review [31]	5	979	10	0.0392 ± 0.0148	$21.621\text{s} \pm 0.572\text{s}$

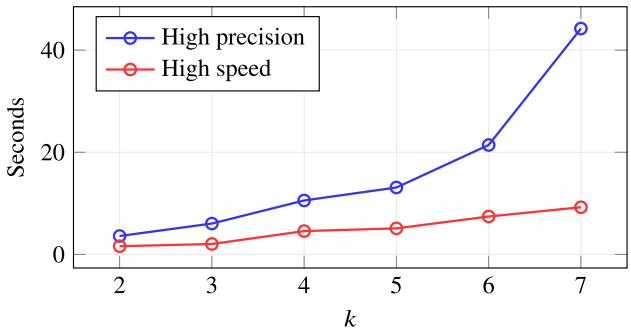


Fig. 16 Computational times for a single iteration of high-precision and high-speed algorithms on Iris dataset

Table 6 `EncryptionParameters` size when serialised

Compression mode	Size
None	537 bytes
ZLIB	115 bytes
Zstandard	130 bytes

limited resources; therefore, it is important to compare the amount of data sent and received between this algorithm and an unsafe request made by sending plain data (or encrypted using a block cipher, which does not necessarily increase file size, like AES) and to calculate the overhead created by CKKS. First of all, server and client must agree on keys and parameters. Microsoft SEAL supports the serialisation of the `EncryptionParameters` object using different compression modes. It contains the scheme type, the polynomial modulus degree, and the coefficients modulus.

ZLIB seems to be the lightest by a little, but Zstandard is preferred due to its speed. Next, the ciphertexts must be sent. Like shown at the beginning of Sect. 2, ciphertexts are composed by slots, which are filled by maximising the value of $\mathcal{E}$ (Definition 1). An experiment is carried out using Iris dataset; its results are visible in Table 7.

Relinearisation keys are created during the keys exchange phase. Ciphertexts can be created in a seeded state in secret-key mode, resulting in a good reduction in the data size. In this client–server scenario, though, ciphertexts are build in public-key mode. In practise, after a relatively long time in setting up the keys (that is performed only once), a high-precision ciphertext (based on $N = 2^{15}$, that contains 2^{14} values) takes about 5.57 MB. A plain `vector<double>` object would take almost 8 bytes$\cdot 2^{14} = 0.13$ MB, a overhead that makes data 40x times larger. Assuming that server and client have already exchanged keys, sending few amounts of MB each iteration seems practical, even though it depends on a lot of factors.

Table 7 Bandwidth when serialised, (pk) means that the ciphertext is created in public key mode, (sk) in secret key

N	Relin keys	Ciphertext size (pk)	Ciphertext size (sk)
2^{15}	57.44 MB	5.57 MB	2.78 MB
2^{14}	10.40 MB	1.65 MB	0.83 MB

3.3 Client workload

After showing the results and the workload on the server, it is important to show the client load. As stated in the Introduction, one of the objectives of the study is to verify whether all the theoretical studies are in some way feasible in practice, thus in an acceptable time frame, and furthermore, if it could be more convenient for the client to ask a server for a computation than to perform a plain iteration by itself. The algorithm executed by the client is shown in Algorithm 3. The first step is to compare execution times for the server-side algorithm and the client-side one, on the same dataset (which is Iris). In particular, the server takes $6.724\text{s} \pm 0.321\text{s}$, while the client takes $0.101\text{s} \pm 0.002\text{s}$. A very meaningful representation of the computation division is given in Fig. 17.

Next, client executions are split in two: decryption and data handling (Table 8).

It is easy to notice that the decryption times are more or less always the same; this is because they are decrypted

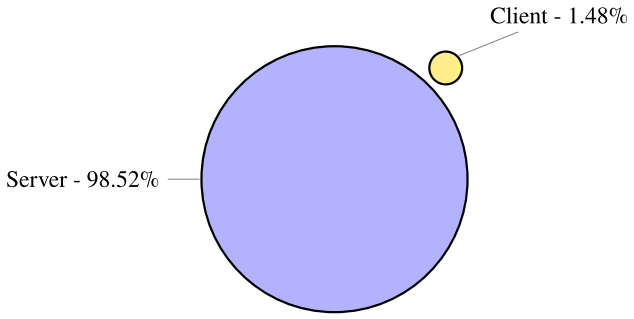


Fig. 17 Single iteration run times clouds chart on Iris dataset

Table 8 Run times of client execution

Dataset name	k	n	m	Decryption	Handling
Iris	4	150	4	0.104s	0.0002s
Absenteeism at work	2	740	20	0.101s	0.0008s
Wine	3	178	12	0.101s	0.0002s
Tripadvisor review	5	979	10	0.113s	0.0015s

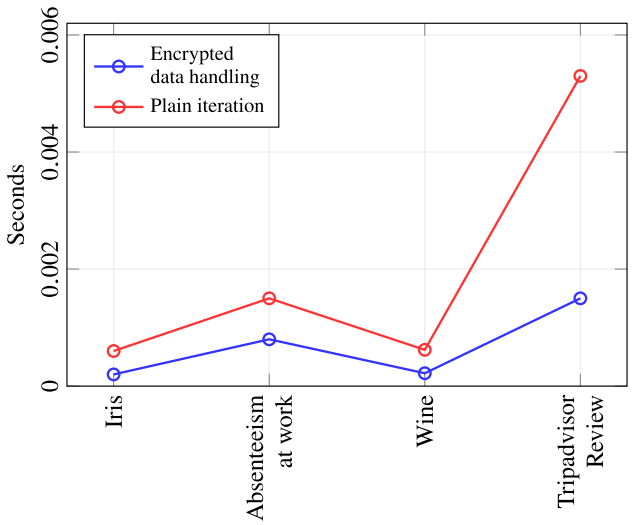


Fig. 18 Runtime of data handling client-side on various datasets compared with runtime of a plain k-means iteration

using a multi-threaded iteration; thus, this time can be seen as constant. Of course, this depends on the number of cores, but in general the complexity of this phase is $\mathcal{O}(\frac{k}{c})$, where c is the number of cores of the machine. It is different when observing the runtime for the handling data task since is strongly related to the number of points of the dataset. In Fig. 18, the handling runtime is compared with a plain k-means iteration time, on the same machine.

This difference grows when using larger datasets, since k-means complexity is higher than the HE data handling one. The bottleneck in this case is given by the decryption phase that takes ≈ 0.1 seconds.

3.4 Encrypting an existent model

It is very easy and straightforward to convert an existing and trained plain model into an encrypted one, starting from the idea that a trained model is nothing more than a set $S = (\mathcal{D}, C, k)$.

In particular, the only differences between a new model and a trained one are the positions of the k centroids. It is sufficient to encode all elements as presented at the beginning of Sect. 2, and it is done. Of course, by adding more points to the dataset $\mathcal{D}$, it would be possible to re-train the model.

3.5 Classification

This part is built on previous reasoning. Basically, this phase needs two elements: a set of centroids C and a point p to be classified. The idea is to calculate a $1 \times k$ distance matrix, and return it to the client. They will be the ones to find the minimum distance in the array and classify the point since:

- they have to perform an operation that has a linear complexity $\mathcal{O}(k)$;
- calculating this result server-side would require a comparison matrix, defined by a quadratic complexity, quite a long computational time (a few seconds on high-precision version), and a greater possibility of errors, whereas on the client is immediate since its aim is to literally find the minimum element in an array that is most likely tiny, since it has a size of k.

Here, the workflow is not split as during the training iterations, since the client has to perform the comparison phase.

Although the idea behind the algorithm is that the server must execute most of the work, this different approach can be accepted only for the classification phase, as it is not a difficult computation (it is rather almost negligible, find the minimum in an array of length k) and doing it server-side would waste too much time and resources (just think about the number of unnecessary slots since the number of columns in a dataset is almost always $<< 2^{15}$).

Notice that executing the comparison phase client side during the training phase would not be this trivial, as the client would have the task of finding the minimum in more, and definitely longer arrays. The classification phase can be summarised as presented in Algorithm 4.

The performance of the model built using encrypted data has been evaluated using the results of a plain k-means classification as an external (or supervised) structure; therefore, the encrypted classifications will be evaluated using an accuracy metric which is defined as the ratio of the number of correctly classified points (with respect to the plain algorithm) to the number of total points considered. In particular, all the points in the dataset have been individually chosen and classified,

Algorithm 4 Homomorphic k-means classification

Input: Point p, set of k centroids
Output: Index of the assigned cluster

1: **procedure** DISTANCES CALCULATION(p, C) ▷ Server-side
2: calculate distances matrix
3: **return** `distmat`
4: **end procedure**

5: **procedure** EVALUATE DISTANCES(`distmat`) ▷ Client-side
6: **return** index of minimum in `distmat`
7: **end procedure**

Table 9 High-precision encrypted classification performance

	m	k	Acc	Execution time
Iris	4	4	1.000	1.305s ± 0.165s
Wine	12	3	0.994	2.805s ± 0.216s
Breast cancer (diagnostic)	30	2	0.996	3.414s ± 0.194s
Absenteeism at work	20	2	0.998	2.512s ± 0.201s
Seeds	6	3	1.000	1.795s ± 0.131s
Transfusion	4	3	0.998	2.011s ± 0.099s
Banknote authentication	4	2	0.987	2.088s ± 0.141s
Tripadvisor review	10	5	0.997	2.124s ± 0.177s

Table 10 High-speed encrypted classification performance

	m	k	Acc	Execution time
Iris	4	4	0.993	0.451s ± 0.015s
Wine	12	3	0.994	1.141s ± 0.156s
Breast cancer (diagnostic)	30	2	0.987	1.794s ± 0.208s
Absenteeism at work	20	2	0.994	1.312s ± 0.145s
Seeds	6	3	0.990	0.541s ± 0.084s
Transfusion	4	3	0.989	0.371s ± 0.031s
Banknote authentication	4	2	0.984	0.241s ± 0.040s
Tripadvisor review	10	5	0.994	1.350s ± 0.121s

using a new set of k random centroids each time (for a total of ten times, in order to have robust results). The results are presented in Table 9 for the high-precision version, in Table 10 for the high-speed one (Acc. stands for accuracy).

One last experiment is done by classifying using another set of parameters for CKKS: $N = 2^{13}$, $\Delta = 2^{50}$. Only one multiplication is available; nevertheless, only one is needed (squaring).

This new version (that has been called extreme high-speed version) handles superbly the classification phase, classifying correctly (namely, just like the plain k-means algorithm) almost all the points with minimal execution times (Table 11), thanks to the high value of the scale.

Considering these results, it would be a good idea to execute the high-precision version for the training phase and the extreme high speed one for the classification phase. In this case, the client should encrypt the final set of centroids generated from the training phase with the new set of parameters ($N = 2^{13}$ and $\Delta = 2^{50}$), in order to set it up for the extreme high-speed classification.

Table 11 Extreme high-speed encrypted classification performance

	m	k	Acc	Execution time
Iris	4	4	1.000	0.006s ± 0.003s
Wine	12	3	1.000	0.012s ± 0.004s
Breast cancer (diagnostic)	30	2	1.000	0.045s ± 0.014s
Absenteeism at work	20	2	1.000	0.023s ± 0.009s
Seeds	6	3	1.000	0.009s ± 0.003s
Transfusion	4	3	0.998	0.005s ± 0.010s
Banknote authentication	4	2	0.999	0.006s ± 0.003s
Tripadvisor review	10	5	1.000	0.021s ± 0.006s

4 Conclusion

The main question, when it comes to this kind of problem, is the following: What are the advantages in building this whole system instead of running a plain k-means locally? Until now, building a homomorphic system that can perform as good as a local computation is still not possible; the main point of this study though, is to show that, at the moment, is somehow possible to build a system that can work *in practice*. Although it is still quicker for the client to execute the algorithm locally (because of the decryption phase, as shown in 3.3), this work improves the practicability of previous privacy-preserving k-means algorithms, achieving execution times in the order of seconds.. This comes at some cost; in fact precision of calculations is sacrificed.

In this paper, a new approach to algorithms based on homomorphic computing, using a method called masking technique, has been proposed. This approach tries to bring almost all computations server-side, so that the client only has to perform a simple masking phase (with a linear complexity), in order to obtain the results. As confirmed by experiments, this phase is faster to execute with respect to a local k-means iteration. Nevertheless, the decryption phase takes a lot of time; this aspect, though, has not been analysed in depth since these procedures depend on the scheme and its implementations. What is intended to be demonstrated here is that, ignoring the decryption phase, for a client is more convenient to ask for a outsourced k-means iteration to a hypothetical server instead of computing it by itself. Lastly, one of the most important and studied aspects has been the practicability of the algorithm (being able to be executed in a short time). It has been demonstrated that the execution can be performed in the order of seconds.

There are several options for future developments. Starting from the mathematical definition of the $\text{sgn}(x)$ approximations, by expanding their range, it would be possible to process columns that are not transformed in $[0, \sqrt{m}/m]$, but in wider intervals. This would make the distances between points larger, resulting in $\text{sgn}'(x)$ giving results closer to 1 (or to -1), hence more precise calculations, meaning more accurate comparisons using the same circuit. Another starting point for future work might be the issue of the calculation of the final masks; it has been shown at the end of Sect. 2 how k defines the number of multiplications necessary for the calculation of the final masks. A large k requires more multiplications than a smaller one, meaning that the comparison function will have a lower degree, hence a lower precision. By finding another way of evaluating the comparison matrix, it would be possible to keep the degree of $\text{sgn}'(x)$ fixed, meaning that the precision of the algorithm would not depend on k. Another task that may be improved is the algorithm that calculates the sum of the first m slots (Algorithm 1), especially in the cases where $m = 2^n - 1$, perhaps by parallelising calculations.

Acknowledgements I would like to sincerely thank Professor Alberto Leporati (Università degli Studi Milano-Bicocca) for introducing me to cryptography and supporting me from the beginning of this research.

Funding Open access funding provided by Università degli Studi di Milano - Bicocca within the CRUI-CARE Agreement. The author did not receive support from any organisation for the submitted work.

Data availibility The datasets generated during and/or analysed during the current study are available in the UCI Machine Learning [14] repository, https://archive.ics.uci.edu/.

Declaration

Conflict of interest The authors declare that they have no competing interests. The authors certify that they have no affiliations with or involvement in any organisation or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors

References

1. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology—CRYPTO 2018, pp. 483–512. Springer, Cham (2018)
2. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology—CRYPTO 2012—Volume 7417, pp. 868-886. Springer, Berlin (2012)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Paper 2011/277. (2011) https://eprint.iacr.org/2011/277
4. Bunn, P. ,Ostrovsky, R.: Secure two-party k-means clustering. In Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07, pp 486-497. Association for Computing Machinery, New York (2007)
5. Catak, F.O., Aydin, I., Elezaj, O., Yildirim-Yayilgan, S.: Practical implementation of privacy preserving clustering methods using a partially homomorphic encryption algorithm. Electronics **9**(2), 1 (2020)
6. Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Hoffstein, J., Lauter, K., Lokam, S., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Security of homomorphic encryption. HomomorphicEncryption.org, Redmond WA, USA, Technical report (2017)
7. Chen, H., Gilad-Bachrach, R., Han, K., Huang, Z., Jalali, A., Laine, K., Lauter, K.: Logistic regression over encrypted data from fully homomorphic encryption. BMC Med. Genom. **11**(4), 81 (2018)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology—ASIACRYPT 2017, pp. 409–437. Springer, Cham (2017)
9. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology—ASIACRYPT 2020, pp. 221–256. Springer, Cham (2020)
10. Cheon, J.H., Kim, D., Kim, D., Lee, H.H., Lee, K.: Numerical method for comparison on homomorphically encrypted numbers. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology—ASIACRYPT 2019, pp. 415–445. Springer, Cham (2019)
11. Cheon, J.H., Kim, D., Park, J.H.: Towards a practical cluster analysis over encrypted data. In: Selected Areas in Cryptography—SAC 2019: 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers, pp. 227-249. Springer, Berlin (2019b)
12. Chiang, J. (2022). Privacy-Preserving Logistic Regression Training with a Faster Gradient Variant. arXiv e-prints, arXiv:2201.10838
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020)
14. Dua, D., Graff, C.: UCI machine learning repository (2017)
15. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugen. **7**(2), 179–188 (1936)
16. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, STOC '09, pp. 169–178. Association for Computing Machinery, New York (2009)
17. Jäschke, A., Armknecht, F.: Unsupervised machine learning on encrypted data. In: Cid, C., Jacobson, M.J., Jr. (eds.) Selected Areas in Cryptography—SAC 2018, pp. 453–478. Springer, Cham (2019)
18. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: Canteaut, A., Standaert, F.-X.

(eds.) Advances in Cryptology—EUROCRYPT 2021, pp. 648–677. Springer, Cham (2021)
19. Liu, D.: Practical fully homomorphic encryption without noise reduction. IACR Cryptol. ePrint Arch. **2015**, 468 (2015)
20. Liu, D., Bertino, E., Yi, X.: Privacy of outsourced k-means clustering. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, pp. 123–134. Association for Computing Machinery, New York (2014)
21. Liu, X., Jiang, Z.L., Yiu, S.M., Wang, X., Tan, C., Li, Y., Liu, Z., Jin, Y., Fang, J.: Outsourcing two-party privacy preserving k-means clustering protocol in wireless sensor networks. In: 2015 11th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), pp. 124–133 (2015)
22. Lou, Q., Jiang, L., Hemet, L.: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In: Meila, M., Zhang, T. (eds) Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pp. 7102–7110. PMLR (2021)
23. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. Cryptology ePrint Archive, Paper 2013/293 (2013). https://eprint.iacr.org/2013/293
24. Malik, M.B., Ghazi, M.A., Ali, R.: Privacy preserving data mining techniques: current scenario and future prospects. In: 2012 Third International Conference on Computer and Communication Technology, pp. 26–32 (2012)
25. Martiniano, A., Pinto Ferreira, R., Sassi, R., Affonso, C.: Application of a neuro fuzzy network in prediction of absenteeism at work. Iberian Conference on Information Systems and Technologies, CISTI, pp. 1–4 (2012)
26. Matwin, S.: Privacy-Preserving Data Mining Techniques: Survey and Challenges, pp. 209–221. Springer, Berlin (2013)
27. Mohassel, P., Rosulek, M., Trieu, N.: Practical privacy-preserving k-means clustering. Proc. Privacy Enhancing Technol. **2020**, 414–433 (2020)
28. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology—EUROCRYPT '99, pp 223–238. Springer, Berlin (1999)
29. Pulido-Gaytan, B., Tchernykh, A., Cortés-Mendoza, J.M., Babenko, M., Radchenko, G., Avetisyan, A., Drozdov, A.Y.: Privacy-preserving neural networks with homomorphic encryption: challenges and opportunities. Peer-to-Peer Networking Appl. **14**(3), 1666–1691 (2021)
30. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 1 (2009)
31. Renjith, S., Sreekumar, A., Jathavedan, M.: Evaluation of partitioning clustering algorithms for processing social media data in tourism domain. In: 2018 IEEE Recent Advances in Intelligent Computational Systems (RAICS), pp. 127–131 (2018)
32. Sakellariou, G., Gounaris, A.: Homomorphically encrypted k-means on cloud-hosted servers with low client-side load. Computing **101**(12), 1813–1836 (2019)
33. SEAL .: Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA (2021)
34. Theodouli, A., Draziotis, K.A., Gounaris, A.: Implementing private k-means clustering using a lwe-based cryptosystem. In: 2017 IEEE Symposium on Computers and Communications (ISCC), pp. 88–93 (2017)
35. Wang, Y.: Notes on two fully homomorphic encryption schemes without bootstrapping. Cryptology ePrint Archive, Paper 2015/519 (2015). https://eprint.iacr.org/2015/519
36. Wood, A., Najarian, K., Kahrobaei, D.: Homomorphic encryption for machine learning in medicine and bioinformatics. ACM Comput. Surv. **53**(4), 1 (2020)
37. Yang, Y., Huang, X., Liu, X., Cheng, H., Weng, J., Luo, X., Chang, V.: A comprehensive survey on secure outsourced computation and its applications. IEEE Access **1**, 1 (2019)
38. Yeh, I.-C., Yang, K.-J., Ting, T.-M.: Knowledge discovery on RFM model using Bernoulli sequence. Expert Syst. Appl. **36**, 5866–5871 (2009)
39. Zhang, E., Li, H., Huang, Y., Hong, S., Zhao, L., Ji, C.: Practical multi-party private collaborative k-means clustering. Neurocomputing **467**, 256–265 (2022)