Yau’s Computer Award 2019

# Differentially Private M-band Wavelet-Based Mechanisms in Machine Learning Environments

Authors: Kenneth Choi and Tony Lee

School: Danbury Math Academy

Danbury, Connecticut, United States of America

Advisor: Xiaodi Wang

August 2019

# Differentially Private M-band Wavelet Based Mechanisms in Machine Learning Environments

Kenneth Choi and Tony Lee

## Abstract

In the post-industrial world, data science and analytics have gained paramount importance regarding digital data privacy. Improper methods of establishing privacy for accessible datasets can compromise large amounts of user data even if the adversary has a small amount of preliminary knowledge of a user. Many researchers have been developing high-level privacy-preserving mechanisms that also retain the statistical integrity of the data to apply to machine learning. Recent developments of differential privacy, such as those in [11], [16], [17], [25], [34], and [35], drastically decrease the probability that an adversary can distinguish the elements in a data set and thus extract user information. In this paper, we develop three privacy-preserving mechanisms with the discrete M-band wavelet transform that embed noise into data. The first two methods ($LS$ and $LS^+$) add noise through a "Laplace-Sigmoid" distribution that multiplies Laplace- distributed values with the sigmoid function, and the third method utilizes pseudo-quantum steganography to embed noise into the data. We then show that our mechanisms successfully retain both differential privacy and learnability through statistical analysis in various machine learning environments.

*Key Words: Differential Privacy, Discrete M-band Wavelet Transform, Laplace-Sigmoid Distribution, Pseudo-Quantum Steganography, Statistical Analysis, Machine Learning Environments*

## Highlights

In this paper, we create three different input perturbation stochastic mechanisms that add or embed noise to sensitive datasets. Our mechanisms improve upon traditional noise addition methods, such as the Laplace mechanism and exponential mechanism mentioned in [11], by using the discrete M-band wavelet transform (DMWT) to convert the dataset into a wavelet domain before adding noise. For the first two mechanisms, we combine the Laplace distribution and the sigmoid function to create a complex stochastic function, and we optimize the mechanisms based on the size of the dataset. In the third mechanism, we propose the use of pseudo-quantum steganography to embed noise into a dataset. Due to the nature of the quantum signal, the noisy dataset has an extremely low probability of being correctly denoised by an adversary. While our proposed mechanisms preserve $\varepsilon$-differential privacy, they also maintain the statistical integrity of the datasets. Using five different supervised machine learning environments—logistic regression, support vector machine, support vector regression, classical artificial neural networks, and deep learning—the mechanisms achieve high accuracies in binary classification across multiple datasets. Moreover, our (pseudo-) quantum mechanism is one of the first to use higher computational power to add noise to private data. As data privacy becomes an extremely important issue in our world, and as quantum computing emerges as a major field, our research can link the two branches and shine a light on what data privacy could potentially look like in the future.

## Table of Contents

## 1 Introduction

More in today's world than ever, there is extreme tension between the mass collection of people's data by corporations and the people's rights to privacy for their personal data. Especially on social media, user data is constantly being collected, and a single mishandling of the data pool can lead to the compromisation of millions of people's data. Recent examples are the Facebook $5 billion FTC fine [14] and the Equifax data breach settlement [12].

There is an important tradeoff between data privacy and statistical analysis. Companies and their research units often rely on user-submitted data for making their products better suited for the market. Users not only wish to preserve anonymity when submitting data but also to be reassured that their individual data cannot be easily identified from a data pool.

If a person has submitted sensitive information in a study whose data has been leaked, an adversary with prior knowledge of the person can use the data to learn something new about the person, and the person's privacy is effectively compromised. For example, a smoker who participates in a survey that requires her to state whether she smokes or not can be harmed if the data is not private. If the study concludes that smoking leads to higher rates of cancer, her insurance rates may be raised if the insurance company finds out she is a smoker from the survey data.

Unfortunately, anonymizing a dataset is not enough to guarantee that someone's information is safely hidden from adversaries. One famous example of a so-called "re-identification attack" is the discovery of Massachusetts governor William Held's medical records in 1997 by a correlation between multiple released datasets [33]: the health database and voter registry. If even anonymization does not protect an individual in a dataset, then what method does?

### 1.1 Prior Works

Differential privacy is a term coined by Dwork *et al*. in [8], and further explained in [9], which sets a privacy budget that quantifies data loss for a mechanism during data release. Over the last decade and a half, researchers have been developing different mechanisms that achieve differential privacy by minimizing the effect of each individual in the dataset. They do this by adding randomly distributed noise to the dataset or a query that slightly obscures the results, enough to ensure privacy but not too much to significantly alter the statistical outcomes.

There are three main ways to add noise to ensure differential privacy, which are input perturbation, which is adding noise to the dataset before a query; objective perturbation, which is adding noise to the objective function in the machine learning model; and output perturbation, which is adding noise to the results after the machine learning model is used.

One well-known differentially private mechanism is the Laplace mechanism [8], which adds Laplace noise with a continuous distribution to a statistical query (output perturbation). However, although it preserves differential privacy, the Laplace mechanism requires the addition of a large amount of noise to both small and large datasets, which hinders the statistical use of the noisy data [30]. Additionally, [30] shows that not adding enough Laplace noise can make the dataset unprotected from a tracker attack.

Another basic mechanism is the exponential mechanism [25], which provides a utilitarian insight into differential privacy by utilizing a quality function to improve upon the usefulness of the

Laplace mechanism. The exponential mechanism is also able to apply to non-numeric queries. However, [3] shows that the noise added through the exponential mechanism is asymptotically non-negligible.

Similar to the exponential mechanism, posterior sampling [7] provides a Bayesian approach to differential privacy, and it uses problem-dependent distributions.

Other mechanisms stated in [36], [18], [13], [5], [34], and [2] use composition theorems and the post-processing invariance property stated in [10] to derive mechanisms based on the basic Laplace and exponential mechanisms. Some mechanisms also use alternative definitions of privacy that may relax or tighten its stipulations, such as [28].

## 1.2 New Ideas

In this research, we use input perturbation to add noise to the dataset directly. We propose three newly derived mechanisms that preserve differential privacy—the first two use a doubly stochastic process that adds Laplace-Sigmoid distributed noise to data, and the third uses pseudo-quantum signals to embed the noise into the data. For all three of our mechanisms, we use discrete 3-band wavelets to transform the data set into the wavelet domain before embedding noise, making the processes impossible to completely reverse without access to the randomized noise.

When first reading our work, one may ask, "What are the benefits of using the discrete M-band wavelet transform?" For one, adding noise to only the approximation part of the transformed matrix, in the case of the first Laplace-Sigmoid mechanism (*LS*), allows the resulting matrix to retain its detailed parts, which are essential for statistical analysis. As a result, we have a transformed matrix that is differentially private yet preserves statistical integrity. Moreover, wavelets effectively scale non-stationary data, which has changing variance and a short-term mean. But, why specifically the M-Band wavelet transform?

Xiao *et al.* in [35] introduces a method of preserving $\varepsilon$-differential privacy via *Privelet* and *Privelet*$^{+}$, which transform the original dataset via the Haar Wavelet Transform (HWT) and add Laplace noise based on calculated weights. *Privelet* performs well with range-count queries, which solves the inability to produce meaningful results from large aggregate queries using Dwork *et al.*'s mechanism.

However, Privelet's use of the Haar wavelet, the simplest wavelet, can result in easier de-noising of the data by an adversary. Additionally, the Haar wavelet's design is static, unlike M-band wavelets. Most M-band wavelets with $M > 2$ have real values and finite support and are orthogonal, which is a property that is hard to obtain with simpler types of wavelets. M-Band wavelets are also smoother and perform well with image analysis.

Using the discrete M-band wavelet transform, we create two mechanisms that use a combination of Laplace noise and the sigmoid function, and one mechanism that utilizes pseudo-quantum signals to input noise into data (pseudo-quantum steganography).

For the first two mechanisms, the combination of the Laplace distribution and the sigmoid function allows the added noise to be dependent on the dataset, not just a single distribution, similar to the Report Noisy Max mechanism in [17]. The addition of the sigmoid function also adds another degree of randomization that does not add to the privacy budget, since the function is bounded.

For the third mechanism, we propose a new method of treating the noise as a watermark, or secret information, to the data instead of a simple additive distribution. Steganography is the technique of masking information behind seemingly innocent text or images. Following principles of steganography, we develop a mechanism that transforms data into pseudo-quantum signals, which are highly difficult to detect by adversaries. Although we do not have access to true quantum computers, we are able to use pseudo-quantum methods that mimic qubits. As a result, the mechanism is able to encrypt the noise into an indistinguishable signal, then embed the signal in the discrete wavelet domain. Because the mechanism includes stochastic elements and does not store any values, the key to decrypt the noise is impossible to be retrieved from an adversary.

One such paper that uses a similar model, especially with the use of quantum computing, is [31]. However, [31] only adds Laplace noise in mechanism, which has the same disadvantages as the Laplace mechanism proposed in [8]: the noise is too large in magnitude for small datasets. The magnitude of the noise in our pseudo-quantum mechanism is able to be controlled by a noise embedding factor.

## 2 Background

### 2.1 Discrete M-band Wavelet Transform (DMWT)

In numerical and function analysis, wavelet transforms decompose an input signal into different frequency levels. The wavelets are discretely sampled and capture both frequency and location information.

Discrete M-band wavelet transforms (DMWT) use M filter banks, where M ≥ 2, to break a K-dimensional signal into M frequency levels. In our paper, we use a slightly modified version of DMWT, choosing to use the signal in multiple column vectors after it is transformed once. In other words, we use the product *WX*, where *W* is the M-band wavelet transform matrix and *X* is the input dataset. The resulting frequency levels include the low-pass approximation (scaling) matrix and M - 1 high-pass detail matrices.

One application of wavelets is in Multiresolution Analysis, shown in [23]. The low-pass filter bank forms linearly independent vectors that span the approximation spaces $V_i$, while the high-pass filter banks form the detail spaces $W_i$. As transformation continues, the approximation spaces are decomposed as the direct sum of higher-level approximation and detail subspaces: $V_i = V_{i+1} \oplus W_{i+1}$.

For instance, the Daubechies 4 wavelet approximation space can be decomposed as $\mathbb{R}^{16} = V_0 = V_2 \oplus W_1 \oplus W_2 \oplus W_3$. When *i* = 3, the subspaces have dimension 2; when *i* = 2, the subspace has dimension 4; and when *i* = 1, the subspace has dimension 8. In the case a 3-band wavelet transform, $V_i = V_{i+1} \oplus W_{i+1,1} \oplus W_{i+1,2}$, and the space can be decomposed as $\mathbb{R}^9 = V_0 = V_2 \oplus W_{2,1} \oplus W_{2,2} \oplus W_{1,1} \oplus W_{1,2}$.

Each M-band orthonormal wavelet has M filter banks. Let an M-band wavelet have filter banks $\alpha^{(1)}$, $\beta^{(1)}$, … , $\beta^{(M-1)}$. Then the filter banks have the properties for $m = 1, \ldots, M - 1$:

$$\|\alpha_i\| = \left\|\beta^{(1)}\right\| = \ldots = \left\|\beta^{(M-1)}\right\| = 1 \qquad (1)$$

$$\sum_{i=1}^{N} \alpha_i = \sqrt{M}, \ \sum_{i=1}^{N} \beta_i^{(m)} = 0 \qquad (2)$$

$$\alpha \cdot \beta^{(m)} = 0 \qquad (3)$$

where $N$ is the length of each filter bank. Additionally, if the M-band wavelet is also k-regular, it has a fourth property:

$$\sum_{i=1}^{N} i^j \cdot \beta_i^{(m)} = 0 \qquad (4)$$

for $j = 0, \dots, K - 1$.

For both of our mechanisms, we use a 2-regular 3-band (k = 2, M = 3) orthonormal wavelet transform to break down our input data into the frequency levels. We obtain the 3-band filter banks from [19].

| $\alpha$ | $\beta^{(1)}$ | $\beta^{(2)}$ |
|---|---|---|
| $\alpha_1$ = 0.33838609728386 | $\beta_1^{(1)}$ = -0.11737701613483 | $\beta_1^{(2)}$ = 0.40363686892892 |
| $\alpha_2$ = 0.53083618701374 | $\beta_2^{(1)}$ = 0.54433105395181 | $\beta_2^{(2)}$ = -0.62853936105471 |
| $\alpha_3$ = 0.72328627674361 | $\beta_3^{(1)}$ = -0.01870574735313 | $\beta_3^{(2)}$ = 0.46060475252131 |
| $\alpha_4$ = 0.23896417190576 | $\beta_4^{(1)}$ = -0.69911956479289 | $\beta_4^{(2)}$ = -0.40363686892892 |
| $\alpha_5$ = 0.04651408217589 | $\beta_5^{(1)}$ = -0.13608276348796 | $\beta_5^{(2)}$ = -0.07856742013185 |
| $\alpha_6$ = -0.14593600755399 | $\beta_6^{(1)}$ = 0.42695403781698 | $\beta_6^{(2)}$ = 0.24650202866523 |

Table 1.1: Filter banks for our paper

$$\begin{bmatrix}
\alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 \\
\alpha_4 & \alpha_5 & \alpha_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_1 & \alpha_2 & \alpha_3 \\
\beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} & \beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} \\
\beta_4^{(1)} & \beta_5^{(1)} & \beta_6^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(1)} & \beta_2^{(1)} & \beta_3^{(1)} \\
\beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)} & \beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} \\
\beta_4^{(2)} & \beta_5^{(2)} & \beta_6^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1^{(2)} & \beta_2^{(2)} & \beta_3^{(2)}
\end{bmatrix}$$

Figure 1.1: Example of a 2-regular 3-band 27 x 27 wavelet transform matrix

## 2.2 Quantum Computing and Pseudo-Quantum Signals

Normal computers store information in classical bits, which exist in one of two states: 0 or 1. In contrast, quantum computers use quantum bits, or qubits, which can be in any linear combinations of $|0\rangle = [1,0]^T$ and $|1\rangle = [1,0]^T$ as states. Qubit states can be expressed as

$$|\psi\rangle = a|0\rangle + b|1\rangle$$

where $a$ and $b$ are complex numbers and $|a|^2 + |b|^2 = 1$. Qubits also represent points on a 3-D unit sphere. As there are infinite points on a unit sphere, there are infinite states for a qubit.

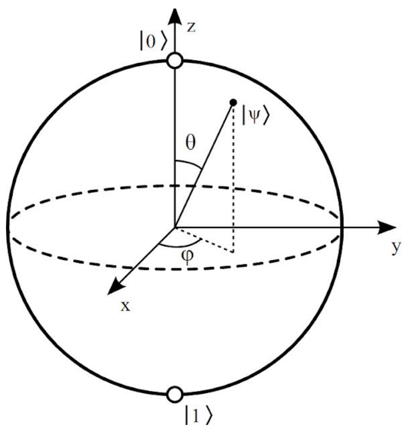


Figure 2.3: The qubit

Another way to write the state of the qubit is

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\varphi} sin\frac{\theta}{2}|1\rangle$$

where θ and φ are real numbers.

As shown in [21], signal $S = [s_1\ s_2\ ...\ s_n]^T$ can be transformed into a pseudo-quantum signal with a transformation $F$, where $F$ is called a pseudo quantum signal converter. Let $F$ be a linear transformation where

$$F(s^*) = \frac{m\pi}{3} \text{ and } F(s_*) = \frac{m\pi}{6}$$

$$s^* = \max(s_i),\ s_* = \min(s_i), m \in \mathbb{N}$$

Then, $F$ transforms signal $S$ into the interval $[\frac{m\pi}{6}, \frac{n\pi}{3}]$ with $\theta_i = F(s_i)$ for $i = 1, 2, \ldots, n$. (In our paper, we use the case were $m = 1$ and $n = 1$.) The transformed signal values are changed into angles $\theta_i$, and $\theta_i$ can be used to redefine a pseudo qubit as

$$|s_i\rangle = \cos(\theta_i)|0\rangle\ +\ \sin(\theta_i)|1\rangle$$

In the case of a pseudo qubit, $\phi = 0$ and the signal can take on values from a circle instead of the sphere shown in Figure 2.3. The transformation result, while not a real quantum signal, is a pseudo-quantum angle that can be computed by classical computers. Thus, pseudo-quantum signals are important for situations when only classical computers can be used to simulate quantum signals. In this research, we propose a new privacy-preserving pseudo-quantum steganography mechanism.

## 3 Privacy-Preserving Mechanisms

### 3.1 ε -differential privacy

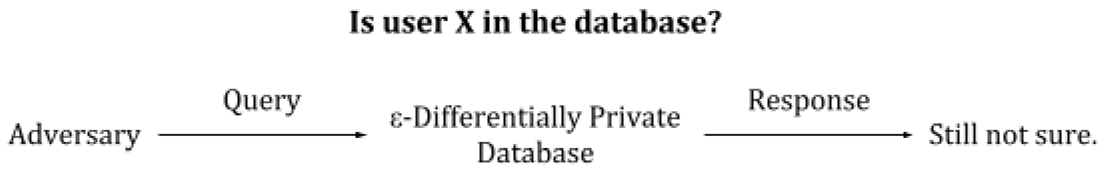
In 2006, Dwork, McSherry, Nissim, and Smith's article [8] introduced the concept of ε -differential privacy, a mathematical definition for the privacy loss associated with any data release drawn from a statistical database.

**Definition 1** *Let D and D' be neighboring datasets, i.e. that they differ in only one element. A randomized mechanism M satisfies* ε *-differential privacy if, for any outputs t,*

$$\frac{\Pr[M(D) = t]}{\Pr[M(D') = t]} \leq \exp(\epsilon)$$

For small values of ε, exp (ε) ≈ ε + 1.

ε -differential privacy guarantees that there is low probability for an adversary to discover new information about a unique individual in the dataset, despite having known prior outside knowledge. This fact means the dataset is robust to post-processing. In other words, the adversary is no more likely to pick the true values than if they were to guess randomly.



A privacy-preserving mechanism also ensures that the inclusion or removal of one data sample will not alter the overall data set significantly. Individuals who are reluctant to submit data in fear that an adversary will identify their information will be assured that their participation has a marginal effect on the output. It is guaranteed that an ε -differentially private result would not be significantly affected regardless of any one individual's truthful participation. However, in order for the overall data to be useful for statistical analysis, ε needs to be scaled accordingly, i.e. each individual data sample needs to have a non-zero impact.

**Definition 2** *Let $f: D^n \to \mathbb{R}^k$, and let D and D' be neighboring datasets. The sensitivity S(f) of f is defined to be*

$$S(f) = \max_{D, D' \in D^n} \|f(D) - f(D')\|_1$$

*where $\|\cdot\|_1$ is the $L_1$ norm.*

For randomized mechanisms *A*, the sensitivity is at most 1, since the neighboring datasets differ in at most 1 element. Dwork *et al.* show in [8] that for any function $f(x) = \Sigma_i x_i$ in a query

$q = f(x) + Y$, where $Y$ is a random variable from the Laplace distribution with mean 0 and scale $1/\varepsilon$, this mechanism is $\varepsilon$-differentially private. However, Dwork *et al.*'s output perturbation Laplace mechanism adds too much noise to even small datasets.

## 3.2 LS Mechanism

### 3.2.1 Steps to Embed LS Noise

Step 1: Discrete M-band Wavelet Transform

Perform the discrete M-band wavelet transform on *D* to obtain the respective approximation and detail parts of the signal. Although the approximation and detail parts are actually composed of individual column vectors representing each sample, we combine the vectors into the corresponding approximation matrix *A* and detail matrices $d_i^{(1)}$ for $i = 1, \ldots, k$.

$$WD = \begin{bmatrix} A \\ d_1^{(1)} \\ \vdots \\ d_k^{(1)} \end{bmatrix}$$

where *k* = *M* - 1. In the case of 3-Band wavelets, *k* = 2.

Step 2: Data-Sensitive Bound Creation

Define μ to be the maximum value of all elements in *A*, and *v* to be the minimum value of all elements in *A*. Then, define a new matrix *A** by transforming each element of *A* to a value bound in the interval [-$\gamma$, $\gamma$] through a linear function $g$. Hence, *A** has the property of being sensitive to the values of the original dataset *D*, and the noise added later is, therefore, fitting for *D*. For $\gamma > 0$, we create *A** by the following procedure:

$$\mu = \max(A_{ij}), \ \ v = \min(A_{ij})$$

$$A_{ij}^* = g(A_{ij}) = \frac{\gamma\,(2A_{ij} - mu - v)}{\mu - v}$$

Step 3: Laplace-Sigmoid Distribution

**Definition 3** *A variable* $N_{ij}(y)$ *from the Laplace-Sigmoid distribution is defined to be*

$$N_{ij}(y) = \begin{cases} (1 - S(y))\,X_{ij} & \text{if } X_{ij} \geq 0 \\ (S(y))\,X_{ij} & \text{if } X_{ij} < 0 \end{cases}$$

*where* $X \sim Lap(0,\ 1/\varepsilon)$ *for some* $\varepsilon > 0$, *and* $S(y) = \frac{1}{1+e^{-y}}$ *is the sigmoid function.*

Like $X$, the noise $N$ has mean 0.

We can create a final noise matrix $N(A^*)$ by using the Laplace-Sigmoid distribution. This hybrid noise-generating mechanism uses the shrunken data values obtained in Step 2 and the values from the Laplace-Sigmoid distribution. We can achieve a roughly equal distribution of positive and negative noise values, as the mean of $N$ is 0. The new noise matrix $N$ is sensitive to the values of the original dataset.

<u>Step 4: Insert New Approximation Matrix</u>

We construct a new approximation matrix $\widehat{A} = A + N(A^*)$ and insert it back into the wavelet transformed data. Because the wavelet matrix $W$ is orthogonal, it is easy to transform the transformed data back to its original domain since $W^T = W^{-1}$. We obtain the "noisy" dataset.

$$\widehat{D} = W^T \begin{bmatrix} \widehat{A} \\ d_1^{(1)} \\ \vdots \\ d_k^{(1)} \end{bmatrix}$$

Notice that $\widehat{D}$ is now differentially private, yet it can still be used with relatively little error in machine learning environments. Experiments in various machine learning environments are covered in Section 4. Note also that because we use binary classification in our statistical analysis, we set a threshold for the last column (labels) in the transformed data such that if the label $y_i \geq 0.5$, then $y_i = 1$, and if $y_i < 0.5$, then $y_i = 0$. The threshold is specific to the transformed dataset and is set based on which threshold effectuates the highest accuracy value. The threshold "rounding" can be thought of as a post-processing mechanism and does not change $\varepsilon$-differential privacy. The following Lemma 1 and Theorem 1 prove that *LS* is $\varepsilon$-differentially private.

**Lemma 1** *Let $f(D) = D$ and $F(D) = f(D) + Y$, where $Y \sim Lap\ (0,\ 1/\varepsilon)$. Let $A(X)$ be a randomized mechanism. Then, $A(F(D))$ satisfies $\varepsilon$-differential privacy.*

*Proof:*

Dwork *et al.* prove in [8] that for any randomized function $F$ such that $F(D) = f(D) + Y$, where $Y \sim \text{Lap}\,(S(F)/\varepsilon)$, $F(D)$ is $\varepsilon$-differentially private. For emphasis, we restate their proof here.

Let a randomized mechanism $F : D^n \rightarrow \mathbb{R}^k$ and an arbitrary point $t \in \mathbb{R}^k$. Then, we can take the ratio of the probability density functions of the Laplace noise added to neighboring datasets $D$ and $D'$:

$$
\begin{aligned}
\frac{\Pr[F(D)=t]}{\Pr[F(D')=t]} &= \prod_{i=1}^{k}\left(\frac{\exp(-\frac{\epsilon|F(D)_i-t_i|}{S(F)})}{\exp(-\frac{\epsilon|F(D')_i-t_i|}{S(F)})}\right)\\
&= \prod_{i=1}^{k}\left(\exp\left(\frac{\epsilon}{S(F)}|F(D)_i-t_i|-|F(D')_i-t_i|\right)\right)\\
&\le \prod_{i=1}^{k}\left(\exp\left(\frac{\epsilon}{S(F)}|F(D)_i-F(D')_i|\right)\right)\\
&= \exp\left(\frac{\epsilon}{S(F)}||F(D)_i-F(D')_i||_1\right)\\
&= \exp\left(\frac{\epsilon}{S(F)}S(F)\right)\\
&= \exp(\epsilon)
\end{aligned}
$$

Thus, by the rules of composability stated in [11], $A(F(D))$ satisfies $\varepsilon$-differential privacy. ▮

**Theorem 1** *Let $LS(D)$ be the LS mechanism with original dataset D of size m x n and let W be the M-band wavelet transform matrix. Then, $LS(D)$ preserves $\varepsilon$-differential privacy.*

*Proof:*
We can write the *LS* mechanism as

$$LS(D) = W^T(WD + N') = f(D) + W^T N',$$

$$\text{where } N' = \begin{bmatrix} N \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ and } f(D) = D$$

Because the sigmoid function $S$ is bounded by $0 < S < 1$, we know that $|N_{ij}| < |X_{ij}|$, where $X_{ij}$ is a random variable given by Lap $(0, 1/\varepsilon)$. Since $W^T$ is an orthonormal M-band wavelet, it preserves the norm of *N'*:

$$||W^T N'|| = ||N'|| < ||X||$$

So, $W^T N'$ adds less noise than *X*. By Lemma 1, a randomized mechanism $A(F(D)) = f(D) + Y$, where $Y \sim \text{Lap}(0,\ S(F)/\varepsilon)$, satisfies $\varepsilon$-differential privacy. Therefore, $LS(D)$ must satisfy $\varepsilon$-differential privacy. ▮

### 3.2.2 De-Noising the LS Dataset

As our mechanism is a function, one should expect that an original dataset would be obtainable by simply applying the inverse transformation on the noisy dataset, which we call $\widehat{D}$:

$$D = \widehat{D} - W^T \begin{bmatrix} N \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

However, since the noise matrix is randomized based on a Laplace-Sigmoid distribution, it is impossible to regain the exact original data without having access to *N*, which is not stored in the first place. Nevertheless, it is possible to regain an approximation of the original dataset, though not very accurate, by having stored the logical values of Laplace-distributed *X* from the mechanism. In order words, we can define matrix the same size as the noise matrix that stores 1 if the noise in that position is positive and -1 if the noise in that position is negative. We call this logical matrix trace(*X*). An adversary should not have direct access to trace(*X*), but instead can create a matrix of random logical values of the same size.

Since we use Laplace noise with mean 0 and scale $1/\varepsilon$, we can construct an approximation of the noise matrix by multiplying trace(*X*) by a factor, then using it in the inverse transformation function to obtain *D*. However, the factor does not directly correspond to *X*, as the random noise used to transform the data is not accessible. So, one way to approach de-noising is to test a new factor *r* that brings us close to the true values of the noise in the original mechanism:

$$D = \widehat{D} - W^T \begin{bmatrix} rN \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

How close of an estimate to the original dataset we can obtain is dependent on the values of $\gamma$ and $\varepsilon$ (in the case of denoising, *r*), though the former does not affect the noise as much as the latter. Of course, an adversary would not know both of the values used in the mechanism.

To test this de-noising method, we use a 243 x 10 training set from Dataset A, mentioned in Section 4.1. To justify our point that obtaining the original dataset is nearly impossible without knowing the mechanism's variables, we take the mean of the absolute difference between the original dataset and the dataset obtained by varing *r*. We do this for values of *r* ranging from 0.001

to 1 with an increment of 0.001, then we take the minimum of the trials. Finally, we do the entire process for 100 trials, corresponding to the value of $\varepsilon$. We keep $\gamma = 1$ constant. Let $H$ = the average minimum absolute difference between the original $D$ and the de-noised $D$.

| $\varepsilon$ | 4 | 2 | 4/3 | 1 | 2/3 | 0.5 |
|---|---|---|---|---|---|---|
| $H$ | 0.0363 | 0.0724 | 0.1088 | 0.1448 | 0.2177 | 0.2895 |

As is shown, $H$ approximately directly varies with $1/\varepsilon$, i.e. $H \propto 1/\varepsilon$. Therefore, unless the value of $\varepsilon$ is large and trace($X$) is somehow released to the public, an adversary would not be able to retrieve a close form of the original dataset by using a de-noising method.

## 3.3 The LS$^+$ Mechanism

### 3.3.1 Versus the LS Mechanism

At times, it is not adequate to apply the $LS$ mechanism on an entire dataset. One such case is if the dataset is too large, then the wavelet matrix may take too long to generate. We encountered this problem by generating wavelets of dimension $3^{11}$ and upwards, as our computers did not have sufficient memory.

The second such case is if the individual wants to disperse the approximation and detail portions of individual blocks in the database rather than add noise to only one of the whole dataset's approximation or details.

Another case in which the $LS^+$ mechanism is the better option is if the number of rows of the dataset is not numerically close to the wavelet's size. Additionally, using the $LS^+$ mechanism allows new entries to be incorporated quicker without having to wait for large amounts of samples.

### 3.3.2 Steps to Embed LS$^{+}$ Noise

To utilize the *LS*$^{+}$ mechanism with 3-band wavelets, as in this paper, the dataset's number of rows must be divisible by the size of the wavelet matrix. We apply the Laplace-Sigmoid distributed noise and add it to the full matrix.

$$D = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} & a_{29} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{18} & a_{39} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} & a_{49} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} & a_{88} & a_{89} \\ a_{91} & a_{92} & a_{93} & a_{94} & a_{95} & a_{96} & a_{97} & a_{98} & a_{99} \\ - & - & - & - & - & - & - & - & - \\ b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} & b_{17} & b_{18} & b_{19} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} & b_{27} & b_{28} & b_{29} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} a \\ - \\ b \\ - \\ \vdots \end{bmatrix}$$

$$\Big\Downarrow$$

$$\begin{bmatrix} a \\ - \\ b \\ - \\ \vdots \end{bmatrix} \xrightarrow{W} \begin{bmatrix} Wa \\ - \\ Wb \\ - \\ \vdots \end{bmatrix} = \begin{bmatrix} a^* \\ - \\ b^* \\ - \\ \vdots \end{bmatrix}$$

Laplace-Sigmoid Distribution

$$\Big\Downarrow$$

$$\widehat{D} = W^T \begin{bmatrix} a^* + N_1 \\ b^* + N_2 \\ \vdots \end{bmatrix}$$

In this case, $N = [N_1\ N_2\ \ldots\ ]^T$ has the same number of rows as *D*. We can write the mechanism as a function $f(D) = D + W^T N$.

The *LS*$^{+}$ mechanism can be proved to be differentially private by Lemma 1 and Theorem 1. It is mathematically similar to the *LS* mechanism, but it is a more complete method of adding noise. We use this mechanism for bigger datasets. For example, since Dataset B described in Section 4.1 is size 3190032 x 9, we can split the dataset into 354448 blocks, each of which is transformed by DMWT.

**3.33 De-Noising the LS⁺ Dataset**

Similar to the *LS* mechanism, we are able, to an extent, to de-noise the transformed dataset $\widehat{D}$ by varying *r* to optimize the function

$$D = \widehat{D} - r \begin{bmatrix} W^T N_1 \\ W^T N_2 \\ \vdots \end{bmatrix}$$

For this example, we use Dataset B of size 3190032 x 9, mentioned in Section 4.1. For 3 trials, we take the minimum of the average absolute difference between the original dataset and the dataset obtained by varying *r*. This time, we test for values of *r* ranging from 0.1 to 1 with an increment of 0.1. $\gamma$ = 1 stays constant. Let the average of 3 trials of the mean of the minimum absolute difference between the original $D$ and the de-noised $D$ be represented by *H*.

| ε | 4 | 2 | 4/3 | 1 | 2/3 | 0.5 |
|---|---|---|---|---|---|---|
| *H* | 23.4855 | 23.4855 | 23.4855 | 23.4855 | 23.4855 | 23.4855 |

The average minimum absolute difference between the original $D$ and the de-noised $D$ does not change as $\varepsilon$ changes. Since *H* > 0, the *LS⁺* mechanism ensures that the $\varepsilon$-differentially private dataset cannot be reversed without much deviation from the original dataset.

## 3.4 Pseudo-Quantum Steganography

### 3.4.1 Quantum Steganography

When using a quantum computer, one can generate a true random qubit $S_{ij}$ that satisfies

$$|S_{ij}\rangle = P_{ij_1}|0\rangle + P_{ij_2}|1\rangle$$

to be used in the quantum embedding of the noise into the approximation coefficients. So, each index can be embedded with qubits instead of a pseudo-quantum simulation:

$$\theta_{ij}^{E} = \begin{cases} \cos^{-1}\left(\cos(\theta_{ij}) + \delta\cos(x_{ij})\right) & \text{if } |P_{ij_1}| \geq |P_{ij_2}| \\ \sin^{-1}\left(\sin(\theta_{ij}) + \delta\sin(x_{ij})\right) & \text{if } |P_{ij_1}| < |P_{ij_2}| \end{cases}$$

However, if one does not have access to a quantum computer, the following pseudo-quantum algorithm must be used.

### 3.4.2 Steps to Embed Pseudo-Quantum Noise

Step 1: Discrete M-band Wavelet Transform

We perform DMWT on the dataset *D* of size *m* x *n* with wavelet matrix *W*. Instead of keeping the approximation part of the dataset in the wavelet domain as a matrix, we split it up into column vectors representing each sample. This mechanism shows the case of a discrete 3-band wavelet transform, which we use in our experiments.

$$WD = \begin{bmatrix} A^1 & A^2 & \dots & A^n \\ d_1^1 & d_1^2 & \dots & d_1^n \\ d_2^1 & d_2^2 & \dots & d_2^n \end{bmatrix}$$

Step 2: Transform Approximation Coefficients into Angles

In order to embed the noise into the approximation signal, we must transform the approximation coefficients into pseudo-quantum signals, or angles. For $k = 1, \dots, n$,

$$\theta_i^k = \frac{\pi(A_i^k + \mu_1^k - 2v_1^k)}{6(\mu_1^k - v_1^k)} \quad \text{for } i = 1, 2, \dots, m_A \text{ and } m_A = \text{number of rows of} A^k,$$

$$\text{where } \mu_1^k = \max(A_i^k),\ v_1^k = \min(A_i^k)$$

This procedure ensures that the new approximation coefficients $\theta_i^k$ are bounded in $[\frac{\pi}{6}, \frac{\pi}{3}]$. Let matrix $\theta = [\theta^1\ \theta^2\ \cdots\ \theta^n]$.

Step 3: Generate Laplace Noise and Transform into Pseudo-Quantum Signals

To make our mechanism stochastic, we randomly generate Laplace-distributed noise with mean 0 and standard deviation $4 / \varepsilon$.

$$X_{ij} = \text{Lap}\left(0, \frac{4}{\epsilon}\right)$$

Then, we transform *X* into the same angle bounds as the approximation coefficients. The noise matrix is created to be the same size as the combined approximation matrix, whose number of columns is just *n*. To be added, it must be transformed into pseudo-quantum signals $x_{ij}$ :

$$x_{ij} = \frac{\pi(X_{ij} + \mu_2 - 2v_2)}{6(\mu_2 - v_2)} \quad \text{for } i = 1, 2, \dots, m_A \text{ and } j = 1, 2, \dots, k,$$

$$\text{where } \mu_2 = \max(X_{ij}),\ v_2 = \min(X_{ij})$$

Step 4: Pseudo-Quantum Embedding

We embed the Laplace noise into the transformed approximation coefficients by mimicking a quantum computer's random generation. Using a classical computer, we randomly generate values

$$k_{ij} = \text{rand}(0,1)$$

and use the values to embed the noise into a new matrix $\theta^E$ such that

$$\theta_{ij}^E = \begin{cases} \cos^{-1}\left(\cos(\theta_{ij}) + \delta\cos(x_{ij})\right) & \text{if } k_{ij} \geq (1-\eta)/2 \\ \sin^{-1}\left(\sin(\theta_{ij}) + \delta\sin(x_{ij})\right) & \text{if } k_{ij} < (1+\eta)/2 \end{cases}$$

where $\delta$ is the embedding intensity and $\eta$ is the embedding bias, $0 < \eta < 1$. We use $\delta = 0.1$ in our trials.

Step 5: Inverse Transformation

In order to obtain the new approximation matrix $A^*$ with the embedded noise, we do the inverse transformation of the linear transformation before:

$$A_{ij}^* = \frac{6\theta_{ij}^E(\mu_1^j - v_1^j)}{\pi} - \mu_1^j + 2v_2^j$$

where $i = 1, \ldots, m_A$ and $j = 1, \ldots, n$.

Step 6: Inverse DMWT

Finally, we perform the inverse wavelet transform after inserting the new approximation matrix $A^*$.

$$D^* = W^T \begin{bmatrix} & A^* & & \\ d_1^1 & d_1^2 & \cdots & d_1^n \\ d_2^1 & d_2^2 & \cdots & d_2^n \end{bmatrix}$$

Because the dataset is binary, we must have two outputs: 0 and 1. Like in the *LS* and *LS*$^+$ mechanisms, we test different values of "rounding" for the specific dataset we use, and we set the threshold for the labels: if $y_i \geq -1.5$, then $y_i = 1$, and $y_i < -1.5$, then $y_i = 0$.

**Lemma 2** *Let* $X_{ij} \sim \text{Lap}\ (0, \sigma)$ *and let* $x_{ij}$ *be the angle bound between* $[\frac{\pi}{6}, \frac{\pi}{3}]$ *corresponding to* $X_{ij}$. *Then, the resulting probability density function of any signal z embedded with x through pseudo-quantum steganography is equal to*

$$\frac{\delta}{\sigma\sqrt{2}}e^{-\frac{2|z|}{\sigma}}$$

*where $\delta$ is the embedding intensity.*

*Proof:*
Let $\theta_{ij}^{E}$ be the resulting signal of $z$ embedded with $x$. We can write the steganography the same way as in Step 4 of the pseudo-quantum mechanism.

Then, the probability density function $f(y)$ of $\delta\cos(x_{ij})$ can be written as

$$\begin{aligned} f(y) &= \delta\frac{d}{dy}\left(\int_0^{\cos^{-1}y}\frac{1}{\sigma\sqrt{2}}e^{\frac{-2|t|}{2}}dt\right)\\ &= -\frac{\delta}{\sigma\sqrt{2}(\sqrt{1-y^2})}e^{-\frac{2\cos^{-1}(y)}{\sigma}}\end{aligned}$$

where $y\in[\frac{1}{2},\frac{\sqrt{3}}{2}]$. Hence, the probability density function $p_1(z)$ of $\cos^{-1}(\delta\cos(x_{ij}))$ is given by

$$\begin{aligned} p_1(z) &= \frac{d}{dz}\left(\int_0^{\cos z}-\frac{\delta}{\sigma\sqrt{2}(\sqrt{1-y^2})}e^{-\frac{2\cos^{-1}y}{\sigma}}dy\right)\\ &= \frac{\delta}{\sigma\sqrt{2}}e^{-\frac{2z}{\sigma}}\end{aligned}$$

where $z\in[\frac{\pi}{6},\frac{\pi}{3}]$.

Similarly, assuming $\sin^{-1}y>0$, the probability density function $g(y)$ of $\delta\sin(x_{ij})$ is

$$\begin{aligned} g(y) &= \delta\frac{d}{dy}\left(\int_0^{\sin^{-1}y}\frac{1}{\sigma\sqrt{2}}e^{\frac{-2|t|}{2}}dt\right)\\ &= \frac{\delta}{\sigma\sqrt{2}(\sqrt{1-y^2})}e^{\frac{2\sin^{-1}(y)}{\sigma}}\end{aligned}$$

where $y\in[\frac{1}{2},\frac{\sqrt{3}}{2}]$, and the probability density function $p_2(z)$ of $\sin^{-1}(\delta\sin(x_{ij}))$ as

$$
\begin{aligned}
p_2(z) &= \frac{d}{dz}\left(\int_0^{\sin z} \frac{\delta}{\sigma\sqrt{2}(\sqrt{1-y^2})} e^{-\frac{2\sin^{-1} y}{\sigma}} dy\right) \\
&= \frac{\delta}{\sigma\sqrt{2}} e^{-\frac{2z}{\sigma}}
\end{aligned}
$$

where $z \in [\frac{\pi}{6}, \frac{\pi}{3}]$. Since $z > 0$, $z = |z|$. Thus, taking the weighted sum of $p_1(z)$ and $p_2(z)$, the probability distribution function $h(z)$ of $\theta_{ij}^E$ can be written as

$$
h(z) = \frac{1-\eta}{2} p_1(z) + \frac{1+\eta}{2} p_2(z) = \frac{\delta}{\sigma\sqrt{2}} e^{-\frac{2|z|}{\sigma}} \quad \blacksquare
$$

**Theorem 2** *The pseudo-quantum mechanism with* $X_{ij} \sim \mathrm{Lap}\ (0, 4/\varepsilon)$ *is* $\varepsilon$*-differentially private.*

*Proof:*
The pseudo-quantum mechanism can be written as a function *f* of *D*, where *A* is the original approximation matrix and *A** is the approximation matrix with embedded noise:

$$
\begin{aligned}
f(D) &= W^T \left( WD + \begin{bmatrix} A^* - A \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right) \\
&= D + W^T \begin{bmatrix} A^* \\ 0 \\ \vdots \\ 0 \end{bmatrix} - W^T \begin{bmatrix} A \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
&\triangleq D + N - D_A
\end{aligned}
$$

Using Lemma 2, we can obtain the probability density function $h(z)$ of $\theta_{ij}^E$. Then, we see that for neighboring datasets *D* and *D'* and for a query *q*,

$$\begin{aligned}\frac{\Pr[q(D+N-D_A)=R]}{\Pr[q(D'+N'-D'_A)=R]} &= \frac{\Pr[N=q^{-1}(R)-D+D_A]}{\Pr[N'=q^{-1}(R)-D+D'_A]}\\ &= \prod_{i=1}^{k}\left(\frac{\frac{\delta\epsilon}{4\sqrt{2}}\exp\left(-\frac{\epsilon}{2}\left(q^{-1}(R)_i-D+D_A\right)\right)}{\frac{\delta\epsilon}{4\sqrt{2}}\exp\left(-\frac{\epsilon}{2}\left(q^{-1}(R)_i-D'+D'_A\right)\right)}\right)\\ &= \prod_{i=1}^{k}\left(\exp\left(\frac{\epsilon}{2}\left(\left|q^{-1}(R)_i-D'+D'_A\right|-\left|q^{-1}(R)_i-D+D_A\right|\right)\right)\right)\\ &\leq \exp\left(\frac{\epsilon}{2}\left(\|D-D'\|+\|D'_A-D_A\|\right)\right)\\ &\leq \exp(\epsilon)\end{aligned}$$

Therefore, by Definition 1, the pseudo-quantum mechanism is $\varepsilon$ -differentially private. ▌

### 3.4.3 De-Noising the Pseudo-Quantum Dataset

Data encrypted with noise from quantum steganography is impossible to de-noise completely. By the Heisenberg Uncertainty Principle and no-cloning mentioned, the noise simply cannot be retrieved without knowing the original dataset, the embedding intensity $\delta$, and the embedding bias $\eta$.

The only way to extract the noise $x$ embedded in the data as pseudo-quantum signals is to use the following formula:

$$x_{ij} = \begin{cases}\cos^{-1}\left(\frac{\cos(\theta_{ij}^{E})-\cos(\theta_{ij})}{\delta}\right) & \text{if } k_{ij} \geq (1-\eta)/2\\ \\ \sin^{-1}\left(\frac{\sin(\theta_{ij}^{E})-\sin(\theta_{ij})}{\delta}\right) & \text{if } k_{ij} < (1+\eta)/2\end{cases}$$

However, extracting the noise is redundant because the adversary would need to have the original approximation coefficients in the first place. Therefore, the transformed dataset is resistant to harmful post-processing.

Moreover, for $\eta$ = 0.5, the probability for an adversary to correctly decode the noise added to the approximation coefficients by randomization in $k_{ij}$ is only $2^{-mn/3}$, where the data is size *m* x *n*.

## 4 Machine Learning Environments and Experiment Results

In our paper, we employ the three mechanisms in five machine learning environments.

### 4.1 Datasets

We utilize 2 different sets of data in our paper. The first dataset, which we name Dataset A, is provided by the MATLAB Statistics and Machine Learning toolbox from MathWorks® [24]. The dataset includes 699 instances of resultant benign or malignant breast cancer, based on 9 predictors: Clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitosis.

We use this dataset for logistic regression, support vector machine (SVM), support vector regression (SVR), and classical artificial neural networks (classical ANN) because of its smaller scale. Traditional machine learning methods do not fare well with the larger dataset, which we reserve for Deep Learning using Google Colab. Additionally, the computers we have available for research are incapable of training and testing very large datasets with the three machine learning environments mentioned.

Since we use a 3-band discrete wavelet in our mechanism, the dataset $D$ must have $3^K$ rows (in which $K \in \mathbb{Z}^+$). For the three machine learning environments, we use 243 samples as a training set and use the other 456 samples to test the statistical integrity of our mechanism.

The second dataset, which we call Dataset B, is obtained from IPUMS [29] and has 8 predictors and 1 binary results column with 3190032 instances. (The original number of instances was 3190040, but we reduce the size by 8 samples to use our $LS^+$ mechanism.) We use this dataset for the classical ANN and Deep Learning. The predictors are: number of generations, detailed information for number of generations, family size, race, detailed information for race, whether the participant is deaf, whether the participant has a cognitive disability, and whether the participant is blind. The binary resultant is whether the participant speaks English or not.

In our paper, we label the dataset we use for a specific experiment as $D$, and the transformed dataset as $D'$.

## 4.2 Logistic Regression

Data analysis often requires tools and algorithms to predict future cases based on pre-existing datasets. Machine learning algorithms, such as logistic regression, take pre-existing datasets separated into predictor variables and label variables, analyze the relationship between the two, and create models that classify or predict future samples of data [27].

We use a logistic regression model as the first machine learning environment to test the $LS$ mechanism with Database A. Logistic regression uses the logistic curve, which uses the sigmoid function. We chose not to use linear regression because our output data is binary; linear regression models would be unhelpful and would predict output values less than zero or greater than one.

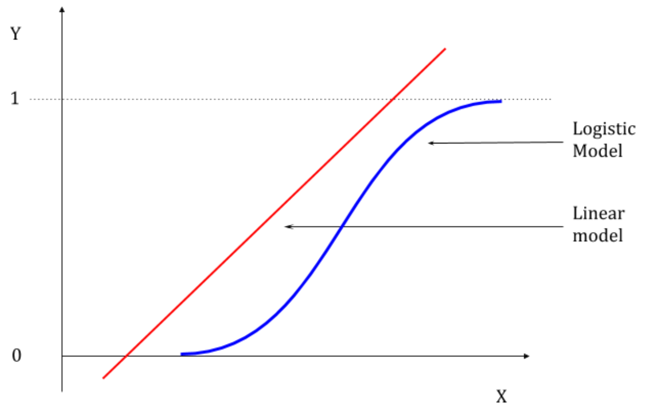


Figure 4.1: Logistic regression model vs. linear regression model

Logistic regression is a classification model represented by *p*, a function of x that represents the probability that the respective label to x is 1. If $p \geq 0.5$, then the model predicts that the x value corresponds to a label value of 1; conversely, if $p < 0.5$, then the model predicts 0 for the label value. The function *p* for a logistic regression model is defined to have the property:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \cdot x$$

where $x$ is the data vector, $\beta_0$ is the bias constant, and $\beta_1$ is the coefficient vector to the predictor variables. The solution of this equation, $\beta_0 + \beta_1 \cdot x$, is called the decision boundary and divides the two label classes. If $x$ is one dimension, the decision boundary would be a point, and if x is two dimensional, it would be a line [4].

The equation of a logistic regression model is the sigmoid function of the logit function of *p*. The logit function is the natural logarithm of the odds of *p*.

$$\text{odds} = \frac{p}{1-p}$$
$$\text{logit}(p) = \ln(\text{odds}) = \ln\left(\frac{p}{1-p}\right)$$

Substituting the logit function of $p$ for the equation $\beta_0 + \beta_1 \cdot x$, we obtain the form of a logistic regression model to be

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot x)}}$$

A logistic regression model uses maximum likelihood estimation (MLE) to solve for the parameters $\beta_0$ and $\beta_1$. Let $p$ be a probability function where the probability of the label variable $y = 1$ is $p$, and the probability of $y = 0$ is $1 - p$. With predictor variable $X = \{x_1, x_2, ..., x_i\}$ and binary label variable $Y = \{y_1, y_2, ..., y_i\}$, the likelihood function is:

$$L(\beta_o, \beta_1) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i)^{1-y_i})$$

Solving for the maximum of the likelihood function requires finding the derivative, but since the likelihood function is difficult to differentiate, the log-likelihood function is used instead:

$$l(\beta_o, \beta_1) = \sum_{i=1}^{n} y_i \log p(x_i) + (1 - y_i) \log 1 - p(x_i)$$

Simplifying the log-likelihood function gives:

$$l(\beta_o, \beta_1) = \sum_{i=1}^{n} -\log 1 + e^{\beta_0 + x_i \cdot \beta_1} + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta_1)$$

Then, we take the partial derivative of the log-likelihood function with respect to $\beta_1$. The partial derivative cannot be set to zero, but can be solved approximately.

$$\begin{aligned} \frac{\partial l}{\partial \beta_1} &= -\sum_{i=1}^{n} \frac{1}{1 + e^{\beta_0 + x_i \cdot \beta_1}} e^{\beta_0 + x_i \cdot \beta_1} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=1}^{n} (y_i - p(x_i; \beta_0, \beta_1)) x_{ij} \end{aligned}$$

The Newton method is a numerical optimization method that minimizes $\beta$ using Taylor polynomials [ ]. Let $\beta*$ be the location of the global minimum, the first derivative of $\beta*$ be zero, and the second derivative of $\beta*$ be positive. A Taylor expansion can be used to approximate the minimum of $f(\beta)$ with $f(\beta^*)$.

$$f(\beta) \approx f(\beta^*) + \frac{1}{2}(\beta - \beta^*)^2 \frac{d^2 f}{d\beta^2}\bigg|_{\beta=\beta^*}$$

A Taylor expansion of the second order can be used to find the global minimum with initial guess $\beta^{(0)}$, as in [4] and [27].

$$f(\beta) \approx f(\beta^{(0)}) + (\beta - \beta^{(0)}) \frac{df}{dw}\bigg|_{\beta=\beta^{(0)}} + \frac{1}{2}(\beta - \beta^{(0)})^2 \frac{d^2 f}{dw^2}\bigg|_{\beta=\beta^{(0)}}$$

Let's call $\frac{df}{dw}|_{\beta=\beta^{(0)}} = f'(\beta^{(0)})$ and $\frac{d^2 f}{dw^2}|_{\beta=\beta^{(0)}} = f''(\beta^{(0)})$. Then we differentiate with respect to $\beta$ and set the derivative to zero. The point at which the derivative is zero is called $\beta^{(1)}$.

$$\frac{\partial}{\partial \beta} = f'(\beta^{(0)}) + \frac{1}{2} f''(\beta^{(0)} 2(\beta^{(1)} - \beta^{(0)}))$$

$$\beta^{(1)} = \beta^{(0)} - \frac{f'(\beta^{(0)})}{f''(\beta^{(0)})}$$

Then $\beta^{(1)}$ can be used to approximate another $\beta$ value that is closer to the location of the global minimum ($\beta^*$). This process is repeated until the lowest two $\beta$ values are found.

$$\beta^{(n+1)} = \beta^{(n)} - \frac{f'(\beta^{(n)})}{f''(\beta^{(n)})}$$

For logistic regression with Dataset A, before training and using a model, we use training dataset $D$ of size 243 x 10 in each mechanism, obtaining a new set $\widehat{D}$ of size 243 x 10. We separate $\widehat{D}$ into the 9 predictors and their resulting targets.

After that, we separate the 456 x 10 testing dataset into the 9 predictors and their respective targets. To test the statistical integrity of the transformed data, the previously constructed logistic regression model uses the testing inputs to predict their resulting targets. We then obtain the absolute difference between the real target values and the target values predicted by the constructed logistic regression model. We sum the number of correct predictions, divide the value by 456, and multiply by 100 to obtain the accuracy in a percentage value. This process makes up one trial. We run 1000 trials for each pair of $\gamma$ and $\varepsilon$ values and record the average accuracy.

Results for the *LS* mechanism with Dataset A:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 79.11% | 77.01% | 74.84% | 73.83% | 78.73% | 81.14% | 80.23% | 81.94% |

Results for the pseudo-quantum mechanism with Dataset B:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% |

## 4.3 Support Vector Machine and Regression

### 4.3.1 Support Vector Machine (SVM)

The support vector machine is a machine learning algorithm used for classifying data into binary classes. An SVM model takes a training data set consisting of predictors $\{x_1, x_2, ..., x_n\}$ and their respective binary labels $\{y_1, y_2, ..., y_n\}$ where $y \in \{0, 1\}$, and constructs a hyperplane that separates the data based on the labels.

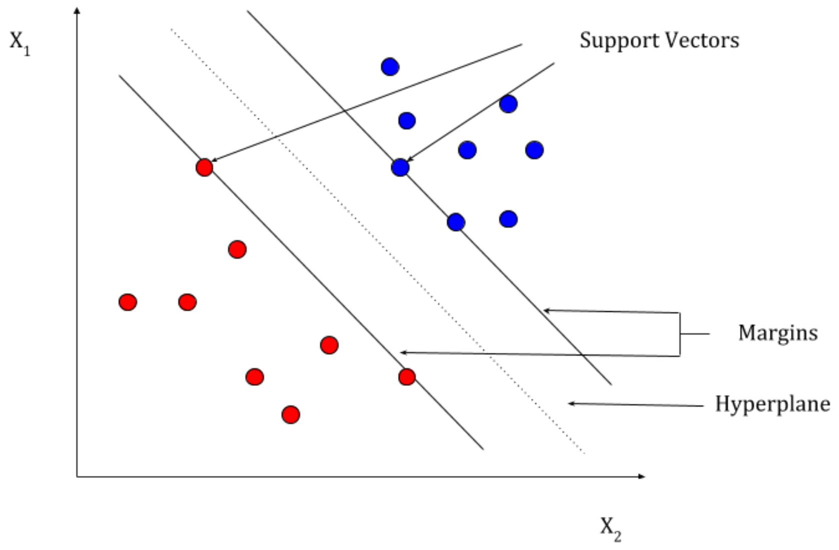


Figure 4.2: Support vector machine model with binary targets

Labels y are classified from predictors *x* and parameters *w* and *b*:

$$y = \operatorname{sign}(w^T x + b)$$

SVM models achieve high classification accuracy by maximizing the width between the margins, or the distance between the hyperplane and the closest vectors on each side (the "support vectors") [15]. With small margins, the hyperplane may overfit the training data on either side and make classification errors with actual testing data.

For a linear SVM model, the margin width is maximized by solving the following primal problem:

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$
$$\text{subject to: } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \ \xi_i \geq 0$$

where $\xi_i$ is the slack variable that gives the model flexibility for some misclassifications while still maintaining the largest possible margins. $C$ represents the trade-off between misclassification cases and large margins. The solution to this minimization problem is also the stationary point of Lagrange function:

$$L(w, b, \xi, \alpha, \beta) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i (y_i(w \cdot x_i + b) - i + \xi_i) - \sum_{i=1}^{n} \beta_i \xi_i$$

where $\alpha_i$ and $\beta_i$ are non-zero Lagrange multipliers [22]. This Lagrange equation is solved by maximizing the following dual problem:

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$\text{subject to: } \sum_{i=1}^{n} y_i \alpha_i = 0,\ \ 0 \le \alpha_i \le C$$

Then, we obtain the weight vector $w$:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

When a linear hyperplane cannot accurately separate the dataset the SVM model uses a nonlinear function $\varphi(x)$ that maps the inputs in a higher dimension and allows the SVM model to separate data that are impossible to linearly separate in lower-dimensional spaces [15]. Then the corresponding kernel function is defined by $K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$. The margin for an SVM model using a kernel function $K(x_i, x_j)$ is maximized by solving the primal problem in the transformed space:

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to: } y_i (\sum_{i=1}^{n} \alpha_i y_i K(x_i, x_j) + b) \ge 1 - \xi_i,\ \ \xi_i \ge 0$$

The margin can also be maximized by the corresponding dual problem in the transformed space:

$$W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to: } \sum_{i=1}^{n} y_i \alpha_i = 0,\ \ 0 \le \alpha_i \le C$$

The weight vector for the kernel function optimization problem then becomes:

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(i)$$

The dual problem in linear SVM and the kernel trick can look complicated, but note that most of the $\alpha_i$ values are equal to $0$. For our paper, we use the Gaussian kernel function in the form

$$K(x_i, x_j) = \exp\left(\frac{-\left\|x_i - x_j\right\|^2}{2\sigma^2}\right), \ \sigma > 0$$

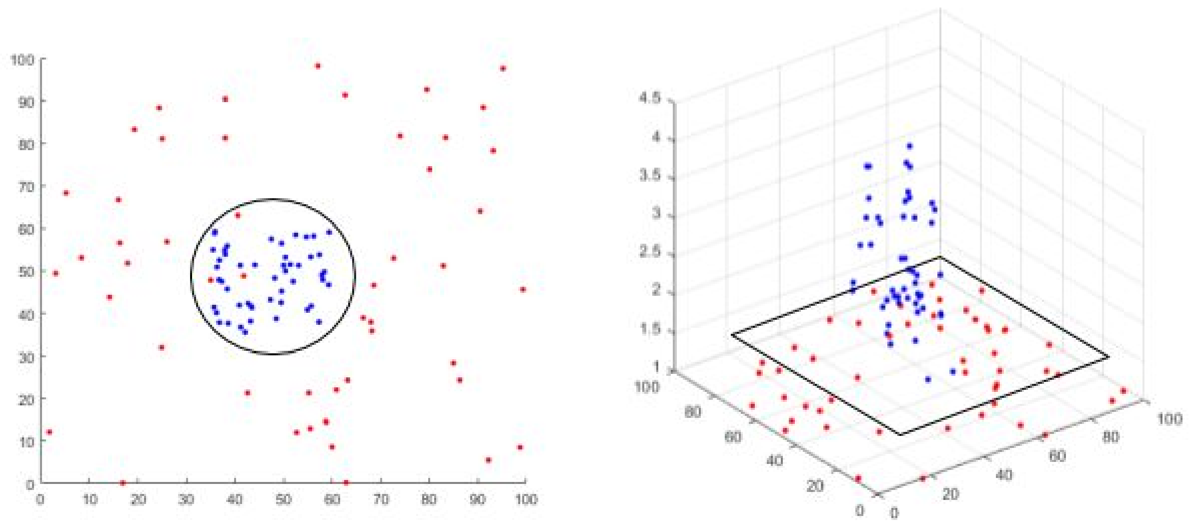


Figure 4.3: Example of data mapped in 3-dimensional space with Gaussian kernel

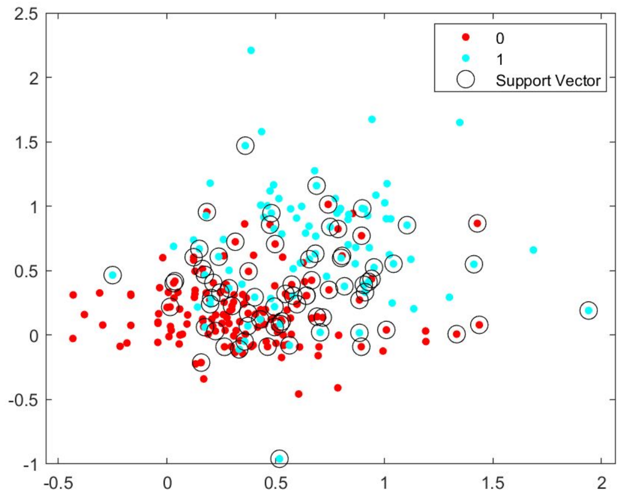


Figure 4.4: Nonlinear support vector machine classification of $\widehat{D}$ with the first two predictors

For testing with Dataset A, we transform the training dataset $D$ of size 243 x 10 with a 3-band discrete wavelet of size 243 x 243 and add noise to the approximation portion. We separate $\widehat{D}$ into the 9 predictors and their resulting targets. The two matrices are then used to train a Support Vector Machine model.

Like logistic regression testing, we separate the 456 x 10 testing dataset into the 9 predictors and their results. The SVM model uses the testing inputs to predict their resulting targets. The accuracy is then calculated in the same manner as logistic regression.

Results for the *LS* mechanism with Dataset A:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 94.71% | 94.16% | 93.52% | 92.83% | 94.78% | 95.58% | 95.33% | 95.61% |

Results for the pseudo-quantum mechanism with Dataset B:

| $\eta$ | 0 | 0 | 0 | 0 | 0.5 | 0.5 | 0.5 | 0.5 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 4/3 | 2 | 4 | 1 | 4/3 | 2 | 4 |
| Accuracy | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% | 94.87% |

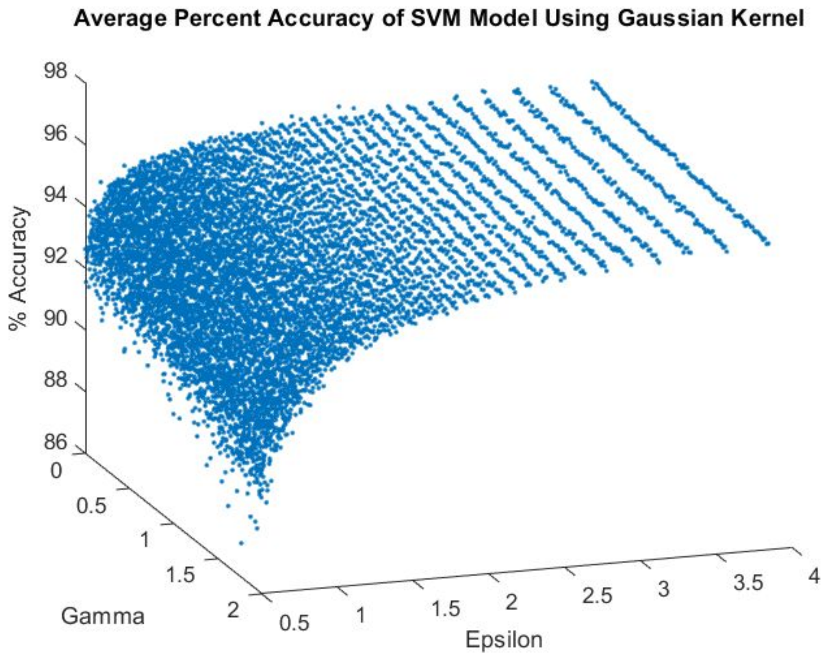


Figure 4.5: 3-D model comparing $\gamma$ and $\varepsilon$ to the average percent accuracy of support vector machine classification for 20 trials for each increment after the *LS* mechanism

### 4.3.2 Support Vector Regression (SVR)

A support vector model can also be used for regression. Instead of classifying data into binary classes in SVM models, support vector regression uses a training set of predictors $\{x_1, x_2, \dots, x_n\}$ and their respective label values $\{y_1, y_2, \dots, y_n\}$, $y \in \mathbb{R}$, to construct a model that attempts to output label values that are within an error bound of $\epsilon$ from the actual observed value (not to be confused with $\varepsilon$ in $\varepsilon$-differential privacy).

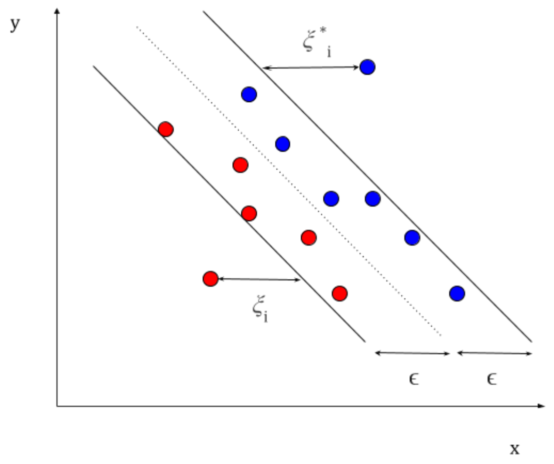


Figure 4.6: Support vector regression model with real targets

SVR differs from SVM by outputting values that are not limited to binary; instead, they range across all real numbers. Therefore, the output $y$ is defined to be

$$y = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \cdot K(x_i, x_j) + b$$

where $K(x_i, x_j)$ is the Gaussian kernel function aforementioned in Section 4.3.1. Since it is impossible for the model to perfectly output points that fall within the $\epsilon$ error bound, slack variables are used to allow errors up to the values of $\xi_i$ and $\xi_i^*$ [32]. The optimization problem for the margin width for an SVR model, which is extremely similar to the optimization equation for SVM due to both using support vector models, then becomes:

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*)$$

$$\text{subject to: } y_i - \left( \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \cdot K(x_i, x_j) + b \right) \leq \epsilon + \xi_i$$

$$\left( \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \cdot K(x_i, x_j) + b \right) - y_i \leq \epsilon + \xi_i^* ,$$

$$\xi_i, \xi_i^* \geq 0$$

In order to find the optimal $\epsilon$ value for the *LS* mechanism, we run 100 trials for each of the $\epsilon$ values ranging from 0.000 to 0.500 in intervals of 0.001. The accuracies from the 100 trials are averaged for each $\epsilon$ value. For larger $\epsilon$ values, the error bounds at either side of the SVR model are larger and the model predicts with more errors. Therefore, we pick the smallest $\epsilon$ value that corresponds with the highest average accuracy across the 100 trials to create SVR models for the 1000 trials.

For the pseudo-quantum mechanism, we run trials for each $\epsilon$ value ranging from 0.250 to 0.500 in intervals of 0.005. The $\epsilon$ testing range for the pseudo-quantum mechanism is determined by 100 test trials that concludes that the optimal $\epsilon$ values are all greater than 0.300 and less than 0.500. The accuracy of the SVR model is then calculated in the same way as the accuracy of the SVM model.

Results for the *LS* mechanism with Dataset A:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| $\epsilon$ | 0.392 | 0.392 | 0.392 | 0.392 | 0.392 | 0.341 | 0.363 | 0.423 |
| Accuracy | 94.83% | 94.25% | 93.58% | 92.87% | 94.90% | 95.60% | 95.43% | 95.88% |

Results for the pseudo-quantum mechanism with Dataset B:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | 0 | 0 | 0 | 0 | 0.5 | 0.5 | 0.5 | 0.5 |
| ε | 1 | 4/3 | 2 | 4 | 1 | 4/3 | 2 | 4 |
| $\epsilon$ | 0.395 | 0.480 | 0.335 | 0.325 | 0.450 | 0.395 | 0.330 | 0.440 |
| Accuracy | 95.30% | 95.30% | 93.80% | 95.30% | 95.33% | 95.30% | 93.33% | 93.10% |

## 4.4 Classical Artificial Neural Networks

Neural networks are an excellent tool for modeling and outperform many other traditional machine learning methods. They can also handle large amounts of data similar to Deep Learning, so after feature extraction, we can utilize the classical neural network to the fullest extent.

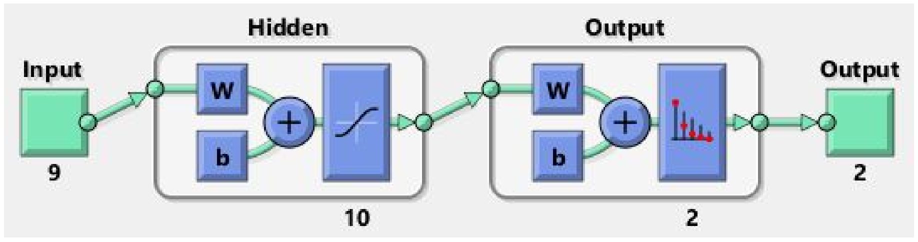


Figure 4.7: Diagram of the classical artificial neural network we utilize

Neural networks consist of an input layer, hidden layers, and an output layer. We use a shallow two-layer feed-forward network with the activation as the sigmoid function,

$$R(z) = \frac{1}{1+e^{-z}}$$

and softmax for the output neurons. To train the network through backpropagation, we use stochastic gradient descent (SGD) provided by the MATLAB Machine Learning toolbox.

### 4.4.1 The Feed-Forward Pass

Our network has parameters $(w,b) = (w^{(1)}, b^{(1)}, \dots, w^{(i)}, b^{(i)})$, where $b_i^{(l)}$ is the bias associated with unit $i$ later $l+1$, and where $w_{ij}^{(l)}$ denotes the parameter (weight) associated with unit $i$ in layer $l$ and unit $j$ in layer $l+1$.

Then, we can let

$$a_i^{(1)} = x_i,\ i = 1, \dots, m \text{ and } z_i^{(2)} = \sum_{j=1}^{n} \left( w_{ij}^{(l)} a_j^{(1)} \right) + b_i^{(l)},$$
$$a_i^{(2)} = R\left(z_i^{(2)}\right), \dots, a_i^{(l)} = R\left(z_i^{(l)}\right)$$

So,

$$\begin{aligned} z_i^{(l+1)} &= \sum_{j=1}^{n} \left( w_{ij}^{(l)} a_j^{l} \right) + b_j^{(l)} \\ &= w_i^{(l)T} a^{(l)} + b_i^{(l)},\ \text{where } a^{(l)} = \begin{bmatrix} a_1^{(l)} \\ \vdots \\ a_m^{(l)} \end{bmatrix} \text{ and } l \geq 1 \end{aligned}$$

Therefore, for input $z_i^{(l+1)}$, the activation function is

$$R\left(z_i^{(l+1)}\right) = R\left(w_i^{(l)T} a^{(l)} + b_i^{(l)}\right)$$

### 4.4.2 Backpropagation

To train a neural network, we use backpropagation. In our case, we use stochastic gradient descent to find the gradient quickly.

In LMS learning, the distance (LMS) is given by $\frac{1}{2}\sum_{p}(T_p - O_p)^2$. We must minimize

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left\|h_{w,b}\left(X^{(i)}\right) - y^{(i)}\right\|^2 + \frac{\lambda}{2}\sum_{i,j}\left(w_{ij}^{(l)}\right)^2$$

where $h_{w,b}(X)$ is the output from the neural network.

First, we perform a feed-forward pass, computing the activations for $a^{(2)}$, $a^{(3)}$, ... and up to the output layer $L_{n_l} = K(z^{(n_l)})$, where $n_l$ is the number of layers. For each output $i$ in $L_{n_l}$, set

$$\delta_i^{(n_l)} = \frac{\partial E}{\partial\left(z_i^{(n_l)}\right)} = -\left(y_i - a_i^{(n_l)}\right)K'\left(z_i^{(n_l)}\right)$$

Then, for all hidden layers $l = n_l - 1, n_l - 2, ..., 2,$ for each node $i$ in layer $l$, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{l+1} w_{ji}^{(l)}\delta_j^{(l+1)}\right)K'\left(z_i^{(l)}\right), \quad \delta^{(l)} = \left(w^{(l)}\delta^{(l+1)}\right)K'\left(z_i^{(l)}\right)$$

and compute

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = a_i^{(l)}\delta_i^{(l+1)} = \nabla_{w^{(l)}} a^{(l)T}\delta^{(l+1)} \text{ and } \frac{\partial E}{\partial b_i^{(l)}} = \nabla_{b^{(l)}}E = \delta^{(l+1)}$$

To implement stochastic gradient descent, set $\Delta_{(0)}^{(l)} = 0$ and $\Delta_{(0)}^{(0)} = 0$ for all $l$. The increment is 0, so there is no change. Then, for $l = 1$ and $n_l = 1$, compute

$$\nabla_{w^{(l)}}E\left(w,b,x_j\right) \text{ and } \nabla_{b^{(l)}}E\left(w,b,x_j\right)$$

To update the gradient, set

$$\triangle w^{(l)}(k+1) = \triangle w^{(l)}(k) + \nabla_{w^{(l)}}E \text{ and } \triangle b^{(l)}(k+1) = \triangle b^{(l)}(k) + \nabla_{b^{(l)}}E$$

Finally, we have

$$w^{(l)}(k+1) = w^{(l)}(k) - \alpha \left[ \frac{1}{2} \triangle w^{(l)}(k) - \lambda w^{(l)}(k) \right]$$

In our classical ANN experiments, we use original Dataset A of size 243 x 11, and apply the privacy-preserving mechanism to obtain the new dataset $\widehat{D}$ of the same size. We separate $\widehat{D}$ into the 9 predictors and their resulting targets. The two matrices are then used to train a classical ANN with 10 hidden neurons. Since the designated machine set is 243 instances, we split it into a personal training set with 171 instances, a validation set with 36 instances, and a personal testing set with 36 instances).

We now test the statistical integrity of the transformed data for the *LS* and pseudo-quantum mechanisms by using the predictors from the 729 instances of the original dataset as a testing set for the classical ANN. We then obtain the rounded absolute difference between the real target values and the target values produced by the neural network. We collect the accuracy of classification using the transformed data itself.

For $LS^{+}$, we use Dataset B of size 3190032 x 10. The last two columns are formed by splitting the binary responses from the original dataset into neural network targets. Similar to the first experiment, we apply the privacy-preserving mechanism to obtain the new dataset $\widehat{D}$ of the same size. We then separate $\widehat{D}$ into the 8 predictors and their resulting targets, and 70% of the instances are designated to the training set while the rest of the instances are split evenly between the validation set and the testing set. Like the first experiment, we use SGD to train the neural network, then we validate and test the neural network with $\widehat{D}$.

Results for the *LS* mechanism with Dataset A:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 95.00% | 94.70% | 91.90% | 90.40% | 94.40% | 94.90% | 95.20% | 95.60% |

Results for the *LS*+ mechanism with Dataset B:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 83.74% | 84.04% | 84.97% | 85.45% | 90.54% | 96.78% | 96.84% | 99.88% |

Results for the pseudo-quantum mechanism with Dataset B:

| $\eta$ | 0 | 0 | 0 | 0 | 0.5 | 0.5 | 0.5 | 0.5 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 4/3 | 2 | 4 | 1 | 4/3 | 2 | 4 |
| Accuracy | 96.7% | 96.7% | 96.7% | 96.8% | 96.6% | 96.7% | 96.6% | 96.6% |

## 4.5 Deep Learning

Deep Learning is by far one of the most important subsets of machine learning methods. Whereas classical artificial neural networks have two or three hidden layers, deep learning models, or deep neural networks, can contain hundreds of hidden layers. Deep learning models take large labeled training datasets and require high computer processing power. Applications for deep learning include self-driving cars, handwriting recognition, and object classification from images.

We choose to use Deep Learning because of its many implications in modern research, including medical imaging and speech recognition in [6] and [20], and its connections with differential privacy [1]. Deep Learning enables automatic feature extraction during the process of learning, rather than the traditional pre-learning manual feature extraction. The result is a more accurate machine learning model and also a model that can handle vast amounts of data, contrary to methods such as logistic regression, SVM, and SVR.

Deep Learning also performs well in classifying large amounts of complex data. Although our paper uses only binary classification with less than 10 features, our mechanism is able to extend much further when used with Deep Learning.

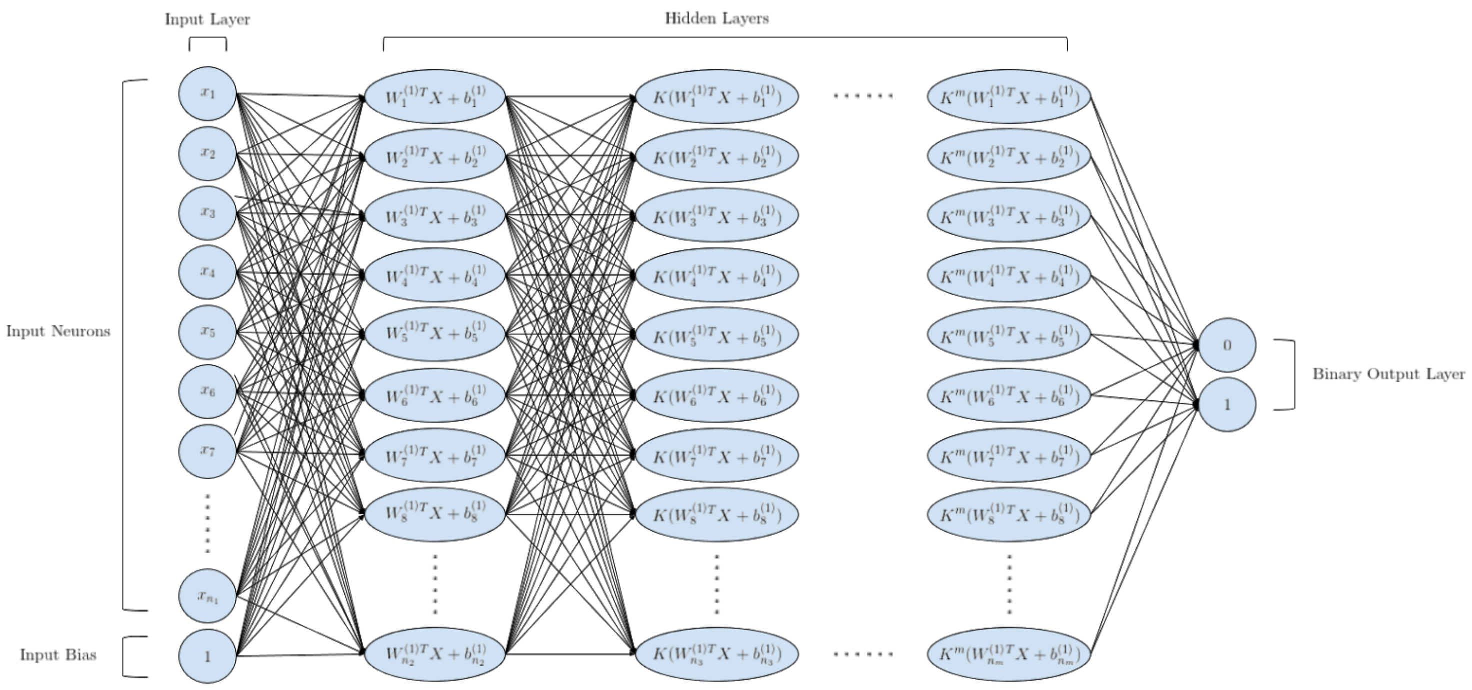

Input Layer
Hidden Layers
Input Neurons
Input Bias
Binary Output Layer
$x_1$
$x_2$
$x_{n_1}$
1
$W_1^{(1)T}X + b_1^{(1)}$
$W_{n_2}^{(1)T}X + b_{n_2}^{(1)}$
$K(W_1^{(1)T}X + b_1^{(1)})$
$K^m(W_1^{(1)T}X + b_1^{(1)})$
0
1

Figure 4.8: A neural network with many hidden layers

Since Deep Learning requires many predictors and instances to learn effectively, we use a Dataset B for $LS^+$ and the pseudo-quantum mechanism. (We use Dataset A for the *LS* mechanism, but since the dataset is small, the accuracy is limited.) We use Google Colab to generate a deep neural network with 4 hidden layers with 16, 18, 20, and 24 neuron units in each layer, respectively. We train the network with back-propagation. The activation function for the hidden layers is RELU:

$$R(z) = \max(0, z)$$

and the activation function for the output layer is the sigmoid function:

$$R(z) = \frac{1}{1 + e^{-z}}$$

Results for the *LS* mechanism with Dataset B:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 76.00% | 72.00% | 76.00% | 72.00% | 76.00% | 76.00% | 76.00% | 76.00% |

Results for the $LS^+$ mechanism with Dataset B:

| $\gamma$ | 0.5 | 1 | 1.5 | 2 | 1 | 0.5 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 1 | 1 | 1 | 4/3 | 2 | 2 | 4 |
| Accuracy | 83.74% | 84.04% | 84.97% | 85.45% | 90.54% | 96.78% | 96.84% | 99.88% |

Results for the pseudo-quantum mechanism with Dataset B:

| $\eta$ | 0 | 0 | 0 | 0 | 0.5 | 0.5 | 0.5 | 0.5 |
|---|---|---|---|---|---|---|---|---|
| ε | 1 | 4/3 | 2 | 4 | 1 | 4/3 | 2 | 4 |
| Accuracy | 95.84% | 96.34% | 96.19% | 96.19% | 96.75% | 97.16% | 97.11% | 97.00% |

## 4.6 Results

1. The average accuracies for the five machine learning environments are all above 70%, demonstrating how all three mechanisms allow companies to add noise while maintaining the same statistical trends in the original data. Furthermore, the average accuracies for the mechanisms in SVM, SVR, and classical ANN are all above 93% for all values of $\varepsilon$ .

The accuracies for the five machine learning methods are largely influenced by the $\varepsilon$ parameter. Our data shows that for the majority of the *LS* and $LS^+$ trials, $\gamma$ = 1 and $\varepsilon$ = 4 result in the highest accuracies. However, larger values of $\varepsilon$ provide less privacy for users according to Definition 1. In addition, companies should refrain from using the same $\gamma$, $\eta$, and $\varepsilon$ values repeatedly in order to prevent adversaries from correcting guessing the values and obtaining the noise matrix, which would allow them to accurately denoise the data. Thus, companies should consider the trade-off between higher differential privacy (smaller $\varepsilon$) and higher statistical integrity (larger $\varepsilon$) when choosing the parameter values.

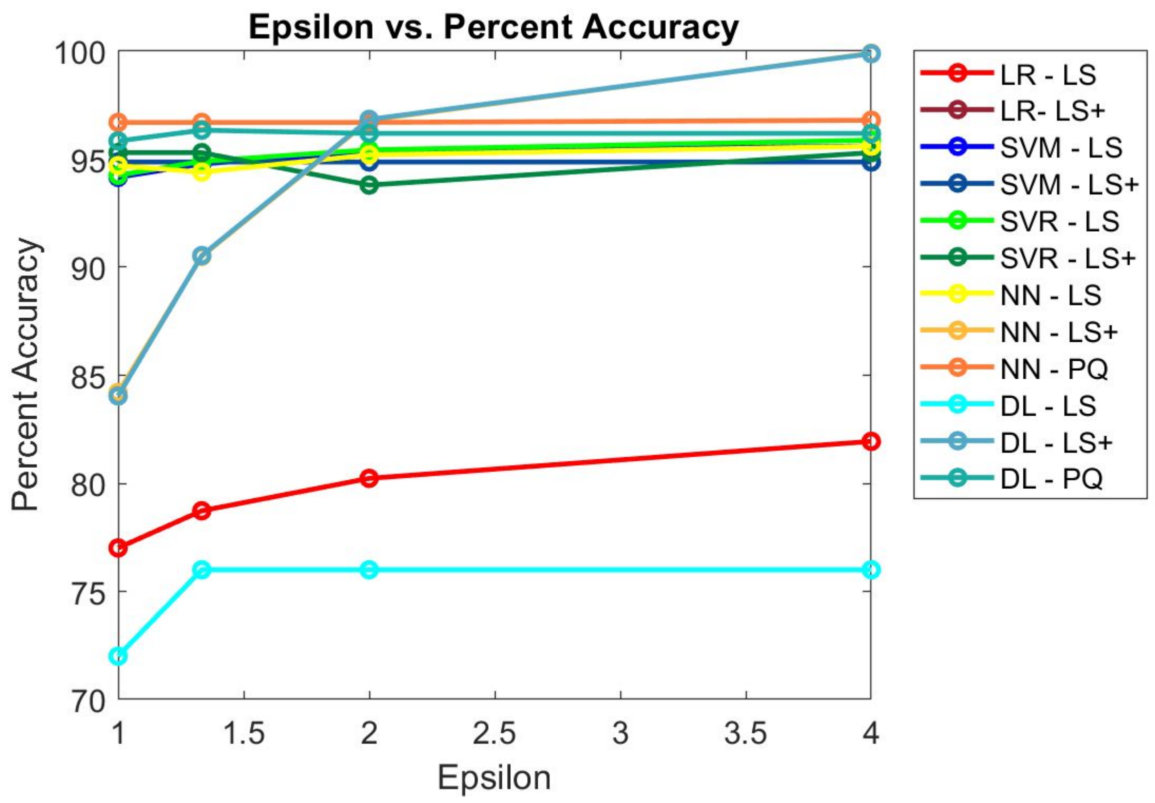


Figure 4.9: Percent accuracy of mechanisms in all machine learning environments for different values of $\varepsilon$. $\gamma = 1$ for *LS* and *LS*$^+$, and $\delta = 0.1$ and $\eta = 0$ for the Pseudo-Quantum (PQ) mechanism.

2. When using the pseudo-quantum steganography to embed noise, the accuracy for SVM and logistic regression consistently remained at high accuracy. The pseudo-quantum mechanism consistently scores high accuracies across all four $\varepsilon$ values for all five machine learning environments, with the lowest average accuracy being 94.87% and the highest being 96.80%. For low $\varepsilon$ values ($\varepsilon = 1$ and $\varepsilon = 4/3$), the pseudo-quantum algorithm outperforms both the *LS* and *LS*$^+$ mechanisms for all five environments. However, for larger $\varepsilon$ values ($\varepsilon = 2$ and $\varepsilon = 4$), LS$^+$ noticeably outperforms the pseudo-quantum and *LS* mechanisms in most machine learning environments, especially for classical artificial neural networks and deep learning models. Companies may want to employ the pseudo-quantum mechanism for small $\varepsilon$ values ($\varepsilon < 2$) but opt for *LS*$^+$ for larger $\varepsilon$ values ($\varepsilon \geq 2$) in conjunction with artificial neural networks or deep learning. Companies should also take into consideration that all three mechanisms work best with large datasets and that the results are more statistically error-prone if a small dataset is used.

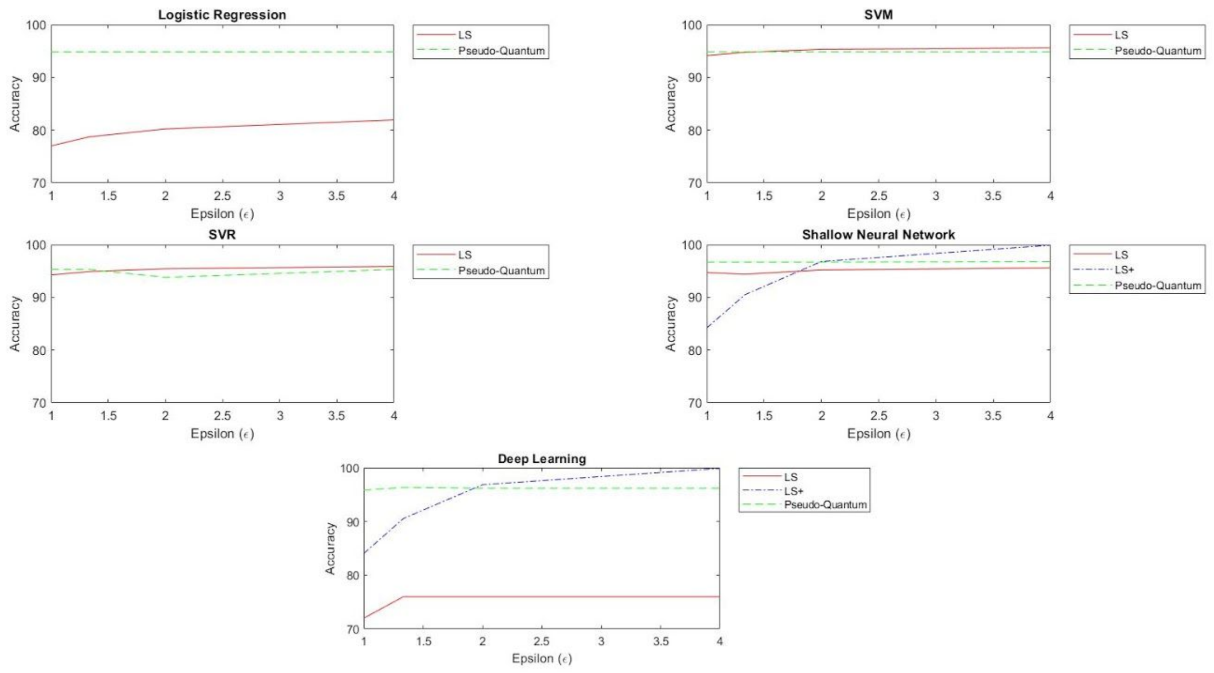


Figure 4.10: Average percent accuracies of mechanisms separated by machine learning environments. $\gamma$ = 1 for *LS* and *LS*$^+$, and $\delta$ = 0.1 and $\eta$ = 0 for the Pseudo-Quantum mechanism.

## 5 Conclusions and Future Research

Instead of spending time and resources attempting to develop encryption methods that do not necessarily guarantee $\varepsilon$-differential privacy or protection from adversary attacks, companies can use privacy-preserving mechanisms to protect personal data. Our three proposed mechanisms encrypt noise into data after DMWT, which presents many advantages over other methods. Removing the noise becomes practically impossible as long as the adversary does not know the exact noise function, which is unique to each scenario, as the function depends on the set of inputs being changed. If put into use, companies will find our input perturbation mechanisms slightly more complicated but more effective.

Our mechanisms add sufficient noise to achieve $\varepsilon$-differential privacy while still preserving overall statistical trends within the dataset. With our three mechanisms, we achieve the following average accuracies for all values of $\varepsilon$ tested in the five machine learning environments:

| | Logistic Regression | SVM | SVR | Classical ANN | Deep Learning |
|---|---|---|---|---|---|
| *LS* | 79.48% | 94.97% | 95.12% | 94.98% | 75.00% |
| *LS*$^+$ | -- | -- | -- | 92.85% | 92.85% |
| Pseudo-Quantum | 94.87% | 94.87% | 94.93% | 96.73% | 96.14% |

For all five machine learning methods, the models correctly predicted the label variables with average accuracies greater than 70%. The results show that companies can employ wavelet transformations and still be able to analyze the dataset for correlations and trends.

In the future, more research and testing can go into improving the security of our first two mechanisms. As our mechanisms require long processing times for large datasets, another priority would be to decrease the computation time by finding new ways to create and store large wavelets and perform the processes.

## 6 References

[1] Abadi, M., Chu, A., Goodfellow, I., McMahan H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep Learning with Differential Privacy. 308-318. doi:10.1145/2976749.2978318

[2] Ács, G., Chen, R., & Castelluccia, C. (2012). Differentially Private Histogram Publishing through Lossy Compression. Proceedings - IEEE International Conference on Data Mining, ICDM. 10.1109/ICDM.2012.80.

[3] Awan, J., Kenney, A., Reimherr, M., & Slavkovic, A. (2019). Benefits and Pitfalls of the Exponential Mechanism with Applications to Hilbert Spaces and Functional PCA.

[4] Breslow, N., & Holubkov, R. (1997). Maximum Likelihood Estimation of Logistic Regression Parameters Under Two-Phase, Outcome-Dependent Sampling. *Journal of the Royal Statistical Society. Series B (Methodological), 59*(2), 447-461. Retrieved from http://www.jstor.org/stable/2346056

[5] Cormode, G., Procopiuc, M., Shen, E., Srivastava, D., & Yu, T. (2011). Differentially Private Spatial Decompositions. Computing Research Repository - CORR. 10.1109/ICDE.2012.16.

[6] Deng, L., Hinton, G., & Kingsbury, B. (2013, 05). New types of deep neural network learning for speech recognition and related applications: An overview. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. doi:10.1109/icassp.2013.6639344

[7] Dimitrakakis, C., Nelson, B., Zhang, Z., Mitrokotsa, A., & Rubinstein, B. I. P. (2017). Differential privacy for bayesian inference through posterior sampling. J. Mach. Learn. Res. 18, 1 (January 2017), 343-381.

[8] Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating Noise to Sensitivity in Private Data Analysis. Theory of Cryptography. Vol. 3876. 265-284. doi:10.1007/11681878_14.

[9] Dwork, C. (2006). Differential Privacy. ICALP, Springer, 1-12. doi:10.1007/11787006_1

[10] Dwork, C. (2008). Differential Privacy: A Survey of Results. *Lecture Notes in Computer Science Theory and Applications of Models of Computation,* 1-19. doi:10.1007/978-3-540-79228-4_1

[11] Dwork, C., & Roth, A. (2014). *The Algorithmic Foundations of Differential Privacy*. Now Publ.

[12] Equifax Data Breach Settlement. (2019, August 01). Retrieved from https://www.ftc.gov/enforcement/cases-proceedings/refunds/equifax-data-breach-settlement

[13] Hay, M., Rastogi, V., Miklau, G., & Suciu, D. (2009). Boosting the Accuracy of Differentially-Private Histograms Through Consistency. Proceedings of the VLDB Endowment. 3. 10.14778/1920841.1920970.

[14] Horwitz, J. (2019, April 25). Facebook Sets Aside $3 Billion to Cover Expected FTC Fine. Retrieved from https://www.wsj.com/articles/facebook-sets-aside-3-billion-to-cover-expected-ftc-fine-11556137113

[15] Hsu, Chih-wei & Chang, Chih-chung & Lin, Chih-Jen. (2003). A Practical Guide to Support Vector Classification.

[16] Ji, Z., Lipton, Z.C., & Elkan, C. (2014). Differential Privacy and Machine Learning: a Survey and Review. *ArXiv*.

[17] Kearns, M., Roth, A., Wu, Z.S., & Yaroslavtsev, G. (2015). Privacy for the Protected (Only). *ArXiv, abs/1506.00242*.

[18] Li, C., Hay, M., Miklau, G., & Wang, Y. (2014). A Data- and Workload-Aware Algorithm for Range Queries Under Differential Privacy. Proceedings of the VLDB Endowment. 7. 10.14778/2732269.2732271.

[19] Lin, T., Xu, S., Shi, Q., & Hao, P. (2006). An algebraic construction of orthonormal M-band wavelets with perfect reconstruction. *Applied Mathematics and Computation, 172*, 717-730.

[20] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A., Ciompi, F., Ghafoorian, M., & Sánchez, C. I. (2017, 12). A survey on deep learning in medical image analysis. *Medical Image Analysis, 42*, 60-88. doi:10.1016/j.media.2017.07.005

[21] Liu, T., & Qiu, Z. (2013). Quantum Watermarking in M-band Wavelet Domain. *Dongrun-Yau Science Awards (Mathematics)*.

[22] Ma, R., Liu, T., Li S., & Wang, X. (2018). "M-band Wavelet Kernels for Classical and Quantum SVM," *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, Chongqing, China, 2018, pp. 515-521. doi: 10.1109/IICSPI.2018.8690433

[23] Mallat, S. G. (1999). *A wavelet tour of signal processing*. Academic Press.

[24] MATLAB and Statistics Toolbox Release 2019a, The MathWorks, Inc, Natick, Massachusetts, United States.

[25] McSherry, F., & Talwar, K. (2007). Mechanism Design via Differential Privacy. *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, Providence, RI, 2007, pp. 94-103. doi: 10.1109/FOCS.2007.66

[26] Murtagh, J., Taylor, K., Kellaris, G., & Vadhan, S. (2018). Usable Differential Privacy: A Case Study with PSI. arXiv:1809.04103.

[27] Park, H. (2013). An Introduction to Logistic Regression: From Basic Concepts to Interpretation with Particular Attention to Nursing Domain. *Journal of Korean Academy of Nursing, 43*(2), 154. doi:10.4040/jkan.2013.43.2.154

[28] Rubinstein, B.I., & Aldà, F. (2017). Pain-Free Random Differential Privacy with Sensitivity Sampling. *ICML*.

[29] Ruggles, S., Flood, S., Goeken, R., Grover, J., Meyer, E., Pacas, J., & Sobek, M. IPUMS USA: Version 9.0 Minnesota Population Center University of Minnesota. Minneapolis, MN: IPUMS, 2019. doi:10.18128/D010.V9.0

[30] Sarathy, R., & Muralidhar, K. (2011). Evaluating Laplace Noise Addition to Satisfy Differential Privacy for Numeric Data. Transactions on Data Privacy. 4. 1-17.

[31] Senekane, M., Mafu, M., & Taele, B. M. (2017, 09). Privacy-preserving quantum machine learning using differential privacy. *2017 IEEE Africon*. doi:10.1109/afrcon.2017.8095692

[32] Smola, A.J. & Schölkopf, B. (2004). A Tutorial on Support Vector Regression. Statistics and Computing 14:199. doi:10.1023/B:STCO.0000035301.49549.88

[33] Sweeney, L. (2002). k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10*, 557-570.

[34] Vadhan, S.P. (2017). The Complexity of Differential Privacy. *Tutorials on the Foundations of Cryptography*. doi:10.1007/978-3-319-57048-8_7

[35] Xiao, X., Wang, G., & Gehrke, J. (2010). Differential privacy via wavelet transforms. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. doi:10.1109/icde.2010.5447831

[36] Zhang, X., Chen, R., Xu, J., Meng, X., & Xie, Y. (2014). Towards Accurate Histogram Publication under Differential Privacy. *SDM*.

# 7 Appendix

## 7.1 Basic Scripts

### 7.1.1 Wavelet Creation Function

```
function A = wavsize(n)

% First 1/3 of the wavelet

A1 = zeros(1,3^n);
A1(1,1)=0.33838609728386;
A1(1,2)=0.53083618701374;
A1(1,3)=0.72328627674361;
A1(1,4)=0.23896417190576;
A1(1,5)=0.04651408217589;
A1(1,6)=-0.14593600755399;
A = zeros(3^n);
A(1,:) = A1;
for i = 1:3^(n-1)
        B = A(i,:);
        C = circshift(B,3,2);
        A(i+1,:) = C;
end

% Second 1/3 of the wavelet

A2 = zeros(1,3^n);
A2(1,1)=-0.11737701613483;
A2(1,2)=0.54433105395181;
A2(1,3)=-0.01870574735313;
A2(1,4)=-0.69911956479289;
A2(1,5)=-0.13608276348796;
A2(1,6)=0.42695403781698;
A((3^(n-1)+1),:) = A2;
for i = (3^(n-1)+1):(2*3^(n-1))
        B = A(i,:);
        C = circshift(B,3,2);
        A(i+1,:) = C;
end

% Third 1/3 of the wavelet

A3 = zeros(1,3^n);
A3(1,1)=0.40363686892892;
A3(1,2)=-0.62853936105471;
A3(1,3)=0.46060475252131;
A3(1,4)=-0.40363686892892;
A3(1,5)=-0.07856742013185;
A3(1,6)=0.24650202866523;
A((2*3^(n-1)+1),:) = A3;
```

```
for i = (2*3^(n-1)+1):3^n
        B = A(i,:);
        C = circshift(B,3,2);
        A(i+1,:) = C;
end
A(3^n+1,:) = [];
```

### 7.1.2 Laplace Noise Function

```
function y  = laprnd(m, n, mu, sigma)
%   LAPRND generate i.i.d. laplacian random number drawn from laplacian distribution
%   with mean mu and standard deviation sigma.
%   mu      : mean
%   sigma   : standard deviation
%   [m, n]  : the dimension of y.
%   Default mu = 0, sigma = 1.
%   For more information, refer to
%   http://en.wikipedia.org./wiki/Laplace_distribution

%   Author  : Elvis Chen (bee33@sjtu.edu.cn)
%   Date    : 01/19/07

%Check inputs
if nargin < 2
    error('At least two inputs are required');
end

if nargin == 2
    mu = 0; sigma = 1;
end

if nargin == 3
    sigma = 1;
end

% Generate Laplacian noise
u = rand(m, n)-0.5;
b = sigma / sqrt(2);
y = mu - b * sign(u).* log(1- 2* abs(u));
```

## 7.2 Mechanism Scripts

### 7.2.1 LS Mechanism

```
function [datanew] = LS(data, gamma, vepsilon)
W = wavsize(5);
B = W * data;
A = B(1:size(data)/3,:);
mu = max(A,[],'all');
v = min(A,[],'all');
a = gamma.*((A - mu - v)/(mu - v));
X = laprnd(size(A,1),size(a,2),0,1/vepsilon);
```

```
astar = zeros(size(X));
[rowsize,colsize] = size(X);
for ii=1:rowsize
    for jj=1:colsize
        if X(ii,jj) >= 0 || X(ii,jj) == 0
            astar(ii,jj) = (1 - 1/(1 + exp(-a(ii,jj)))) * X(ii,jj);
        else
            astar(ii,jj) = (1/(1 + exp(-a(ii,jj)))) * X(ii,jj);
        end
    end
end
Ahat = A + astar;
Bhat = [Ahat; B(size(data)/3 + 1:end,:)];
datanew = W' * Bhat;

dataresp = datanew(:,size(datanew,2));
dataresp(dataresp >= 0.5)=1;
dataresp(dataresp < 0.5) = 0;
datanew = [datanew(:,1:size(datanew,2)-1) dataresp];
end
```

### 7.2.2 LS$^+$ Mechanism

```
function [datanew] = LSplus(data, gamma, vepsilon)

%Waveletblock
W = wavsize(2);
[m,n] = size(data);
result = zeros(m,n);
for i = 1:m/9
    c = 1+9*(i-1);
    d = data(c:9*i,:);
    t = W*d;
    result(c:9*i,:) = t;
end

%Algorithm
[m,n] = size(result);
A = result(1:m,:);
mu = max(A,[],'all');
v = min(A,[],'all');
a = gamma.*((A - mu - v)/(mu - v)); %Multiply a by gamma to control noise (this is the gamma parameter)
X = laprnd(m,n,0,1/vepsilon);
[rowsize,colsize] = size(X);
astar = zeros(size(X));
for ii=1:rowsize
    for jj=1:colsize
        if X(ii,jj) >= 0 || X(ii,jj) == 0
            astar(ii,jj) = (1 - 1/(1 + exp(-a(ii,jj)))) * X(ii,jj);
        else
            astar(ii,jj) = (1/(1 + exp(-a(ii,jj)))) * X(ii,jj);
        end
    end
end
Ahat = A + astar; %Multiply astar by delta to control noise (this is the delta parameter)
Bhat = [Ahat; result(m + 1:end,:)];

%Waveletblockreverse
[m,n] = size(Bhat);
resulthat = zeros(m,n);
for i = 1:m/9
    c = 1+9*(i-1);
    d = Bhat(c:9*i,:);
    t = W' * d;
    resulthat(c:9*i,:) = t;
end

[m,n] = size(resulthat);
r = resulthat(:,n);
r(r >= 0.5) = 1;
r(r < 0.5) = 0;
datanew = [resulthat(:,1:n-1) r];
end
```

### 7.2.3 Pseudo-Quantum Mechanism

```
function [datanew] = PQ(data,vepsilon,eta)
W = wavsize(3);
B = W * data;
A = B(1:size(data,1)/3,:);

mu1 = zeros(1,size(data,2));
v1 = zeros(1,size(data,2));
theta = zeros(size(A));

for k = 1:size(data,2)
    for i = 1:size(data,1)/3
        mu1(1,k) = max(A(:,k));
        v1(1,k) = min(A(:,k));
        theta(i,k) = pi.*((A(i,k) - max(A(:,k)) - min(A(:,k)))/(6*(max(A(:,k)) - min(A(:,k)))));
    end
end

X = laprnd(size(data,1)/3,size(data,2),0,4/vepsilon);

mu2 = max(X,[],'all');
v2 = min(X,[],'all');
alpha = pi.*((A - mu2 - v2)/(6*(mu2 - v2)));

K = rand(size(data,1)/3,size(data,2));

thetae = zeros(size(X));
```

```
for ii=1:size(data,1)/3
    for jj=1:size(data,2)
        if K(ii,jj) >= (1-eta)/2
            thetae(ii,jj) = acos(cos(theta(ii,jj)+0.011*cos(alpha(ii,jj))));
        else
            thetae(ii,jj) = asin(sin(theta(ii,jj))+0.011*sin(alpha(ii,jj)));
        end
    end
end

Astar = zeros(size(A));
for m = 1:size(data,1)/3
    for n = 1:size(data,2)
        Astar(m,n) = (6/pi)*(mu1(1,n)-v1(1,n))*thetae(m,n) - mu1(1,n) + 2*v1(1,n);
    end
end

Bstar = [Astar; B(size(data,1)/3 + 1:end,:)];
datanew = W' * Bstar;
dataresp = datanew(:,size(datanew,2));
dataresp(dataresp >= -1.5) = 1;
dataresp(dataresp < -1.5) = 0;
datanew = [datanew(:,1:size(datanew,2)-1) dataresp];
end
```

## 7.3 Machine Learning Environments Scripts

### 7.3.1 Logistic Regression

```
function [averagepercentage] = logistic(datanew,testingset)
Results = zeros(1000,1);
for i = 1:1000
    %Define & train the classifier
    table = array2table(datanew);
     modelspec = 'datanew9 ~ datanew1*datanew2*datanew3*datanew4*datanew5*datanew6*datanew7*datanew8 -
datanew1:datanew2:datanew3:datanew4:datanew5:datanew6:datanew7:datanew8';
    mdl = fitglm(table,modelspec,'Distribution','binomial');
    %Testing set
    ptest = testingset(:,1:size(testingset,2)-1);
    rtest = testingset(:,size(testingset,2));
    results = predict(mdl,ptest);
    resultsround = round(results);
    diff = rtest - resultsround;
    right = sum(diff(:) == 0);
    percentage = right/size(testingset,1);
    Results(i,1) = percentage;
end
averagepercentage = 100*mean(Results);
end
```

### 7.3.2 Support Vector Machine

```
function [averagepercentage] = svm(datanew,testingset)
Results = zeros(1000,1);
for i = 1:1000
    %Define & train the classifier
    datanewinputs = datanew(:,1:size(datanew,2)-1);
    datanewtargets = datanew(:,size(datanew,2));
    mdl = fitckernel(datanewinputs, datanewtargets);
    %Testing set
    ptest = testingset(:,1:size(testingset,2)-1);
    rtest = testingset(:,size(testingset,2));
    results = predict(mdl,ptest);
    resultsround = round(results);

    diff = rtest - resultsround;
    right = sum(diff(:) == 0);
    percentage = right/size(testingset,1);
    Results(i,1) = percentage;
end
averagepercentage = 100*mean(Results);
end
```

### 7.3.3 Support Vector Regression

```
function [averagepercentage] = svr(datanew,testingset)
%   100 trials for epsilon values (0:0.001:0.5 for LS and LS+, 0.250:0.005:0.500 for PQ),
%   picks the SMALLEST epsilon value that results in highest average accuracy
e_Results = zeros(100,1);
e_compare = zeros(501,2);

for d = 0:0.001:0.5
count = round((d/0.001)+1);
e_compare(count,1) = d;
for b = 1:100
    datanewinputs = datanew(:,1:size(datanew,2)-1);
    datanewtargets = datanew(:,size(datanew,2));
    mdl = fitrkernel(datanewinputs, datanewtargets,'Epsilon',d);
    %Testing set
    ptest = testingset(:,1:size(testingset,2)-1);
    rtest = testingset(:,size(testingset,2));
    e_results = predict(mdl,ptest);
    e_resultsround = round(results);

    e_diff = rtest - e_resultsround;
    e_right = sum(diff(:) == 0);
    e_percentage = right/size(testingset,1);
    e_Results(b,1) = e_percentage;
end
e_average = mean(e_Results);
e_compare(count,2)=e_average;
end
```

```
best_acc = max(e_compare(:,2));
best_acc = round(best_acc,4);
for f = 1:501
if round(e_compare(f,2),4) == best_acc
best_e = e_compare(f,1);
break
end

%   1000 trials
Results = zeros(1000,1);
for i = 1:1000
    %Define & train the classifier
    datanewinputs = datanew(:,1:size(datanew,2)-1);
    datanewtargets = datanew(:,size(datanew,2));
    mdl = fitrkernel(datanewinputs, datanewtargets,'Epsilon',best_e);
    %Testing set
    ptest = testingset(:,1:size(testingset,2)-1);
    rtest = testingset(:,size(testingset,2));
    results = predict(mdl,ptest);
    resultsround = round(results);

    diff = rtest - resultsround;
    right = sum(diff(:) == 0);
    percentage = right/size(testingset,1);
    Results(i,1) = percentage;
end
averagepercentage = 100*mean(Results);
end
```

### 7.3.4 Deep Learning

```
from google.colab import drive

drive.mount('/content/gdrive')

#
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import keras
import io
df = pd.read_csv('/content/gdrive/My Drive/datanew.csv')
df.head(1)

df['Class'].unique() # 0 = no, 1 = yes

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.1, random_state=1)

sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

clf = Sequential([
    Dense(units=16, kernel_initializer='uniform', input_dim=9, activation='relu'),
    Dense(units=18, kernel_initializer='uniform', activation='relu'),
    Dropout(0.25),
    Dense(20, kernel_initializer='uniform', activation='relu'),
    Dense(24, kernel_initializer='uniform', activation='relu'),
    Dense(1, kernel_initializer='uniform', activation='sigmoid')
])

clf.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

clf.fit(X_train, Y_train, batch_size=15, epochs=4)

score = clf.evaluate(X_test, Y_test, batch_size=128)
print('\nScore: ', score[1] * 100, '%')
```

## 8 Acknowledgment

We would like to acknowledge the contributions of each member of the team. Kenneth led and kept the team on task, created and coded the three mechanisms, coded most of the machine learning environments in MATLAB and Python, and wrote the sections related to the mechanisms and their proofs. Tony worked extensively on writing about the machine learning environments, delving into their mathematical steps. He also coded a considerable number of scripts in MATLAB regarding the wavelet transform and support vector regression. Kenneth and Tony ran a significant amount of trials and collected their results. Both team members contributed substantial time and effort into the research.

We would also like to extend our special thanks to our adviser, Dr. Wang, who has provided us with class time to learn about wavelets and machine learning, innumerable hours of assistance, and many resources. We are also very grateful for our assistant, Meera Sharma, who helped us connect our research to the wavelet transform, and for our mentors Ralph Venezia and Hieu Nguyen, who generously introduced us to the basics of MATLAB and Python. We have come to learn many aspects of mathematics, computer science, and teamwork in a limited time.