Efficient Homomorphically Encrypted Privacy-Preserving Automated Biometric Classification

by

David Benjamin Stein

S.B. Computer Science
Massachusetts Institute of Technology, 2011

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfilment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
SEPTEMBER 2020

Author: ....
David Stein
Department of Electrical Engineering and Computer Science
August 20, 2020

Certified by: ....
Daniela Rus
Erna Viterbi Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: ....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Efficient, Homomorphically Encrypted, Privacy-Preserving Automated Biometric Classification

by

David Benjamin Stein

Submitted to the Department of Electrical Engineering and Computer Science on August 21, 2020

In partial fulfilment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis investigates whether biometric recognition can be performed on encrypted data without decrypting the data. Borrowing the concept from machine learning, we develop approaches that cache as much computation as possible to a pre-computation step, allowing for efficient, homomorphically encrypted biometric recognition. We demonstrate two algorithms: an improved version of the k-ishNN algorithm originally designed by Shaul et. al. in [1] and a homomorphically encrypted implementation of a SVM classifier. We provide experimental demonstrations of the accuracy and practical efficiency of both of these algorithms.

Thesis Supervisor: Daniela Rus

Title: Erna Viterbi Professor of Electrical Engineering and Computer Science

Table of Contents

CHAPTER 1: INTRODUCTION **15**

1.1 MOTIVATION 15

1.2 HOMOMORPHIC ENCRYPTION (HE) 16

1.2.1 Homomorphically Encrypted Classification Algorithms 18

1.3 CONTRIBUTIONS 19

1.3.1 Algorithmic Contributions 20

1.3.2 Experimental Contributions 21

1.4 TECHNOLOGY VS. REGULATION 22

1.5 GOALS 23

CHAPTER 2: RELATED WORK **25**

2.1 BIOMETRIC RECOGNITION 25

2.2 PRIVACY ENHANCING TECHNOLOGIES (PETS) AND ENCRYPTION 27

2.3 HOMOMORPHIC ENCRYPTION 28

CHAPTER 3: PROBLEM FORMULATION AND FOUNDATIONAL WORK **31**

3.1 PROBLEM FORMULATION 31

3.2 MOTIVATING PROBLEM: FACIAL RECOGNITION 32

3.3 PRELIMINARIES 33

3.3.1 Notation and pre-defined functions 33

3.3.2 Variables Names 35

3.3.3 Base-p representation 36

3.4 FOUNDATIONAL WORK 36

3.4.1 Overview of $\boldsymbol{k}$-ishNN 37

3.4.2 $\boldsymbol{k}$-ishNN in more detail 38

3.4.3 Issues and Assumptions in $\boldsymbol{k}$ishNN 40

CHAPTER 4: TECHNICAL APPROACH **45**

4.1 Generalization of $\boldsymbol{k}$-ishNN: adding functional generators of candidate threshold families. 46

4.1.1 Overview of $\boldsymbol{k}$-ishNN with precomputation 47

4.1.2 Discussion 49

4.1.3 Efficiency Analysis 53

4.1.4 Security 54

4.2 H.E. SVM 56

4.2.1 Efficiency 58

4.2.2 Security 58

CHAPTER 5: RESULTS **60**

5.1 Implementation Details 60

5.2 Tests 61

5.2.1 Breast Cancer 62

5.2.2 Cars 64

5.2.3 Faces 66

5.3 Metrics 70

5.3.1 Accuracy 70

5.4 The Experiments 71

5.4.1 Accuracy 71

5.4.2 Timing 77

5.5 Discussion 80

CHAPTER 6: CONCLUSION **81**

6.1 Next Steps / Future Work 81

6.2 Practical applications unlocked 82

6.3 POTENTIAL INTERACTIONS WITH TECHNOLOGY POLICY ____ 83

6.4 LESSONS LEARNED ____ 84

6.5 FINAL THOUGHTS ____ 86

BIBLIOGRAPHY ____ 87

APPENDIX A – SOURCE CODE ____ 97

A.1 – SIMULATION CODE FOR PPKNN AND SIMULATION AND IMPLEMENTATION OF PPSVM ____ 97

Table of Figures

Figure 1 – the expected value of $T*$ assuming perfect approximations on a normalized 2d LDA projection of Wisconsin Breast Cancer database used in both this thesis and the original assessment of the k-ishNN algorithm. Red regions are areas in which $T*$ is expected to be negative, implying a greater than 50% chance of an overflow and an effectively arbitrary value of κ. 41

Figure 2 – The expected value of κ without overflow for benign (orange) and malignant (blue) tumors. The graphs show the value of κ along the horizontal axis. The charts, from left to right, top to bottom, represent the distribution of values of κ for target values of k = {2,4,8,16,32,64} respectively. The spikes at $\kappa = 0$ correspond to a high likelihood of overflow when computing a threshold. These graphs were generated using the 2d LDA projection of Wisconsin Breast Cancer database used in both this thesis and the original assessment of the $k$ishNN algorithm..43

Figure 3 - the accuracy of the k-ishNN algorithm on a 2D projection of the motivating facial recognition dataset is low, even with large grid sizes...44

Figure 4: LDA projection of the breast cancer diagnostic dataset into two dimensions. The green cluster shows benign tumors; the red cluster is malignant...63

Figure 5: LDA projection of the breast cancer diagnostic dataset in three dimensions. As in Figure 4, the green cluster (right) shows benign tumors, while red shows malignant tumors (left). Axes are still meaningless. Note that again the clusters are relatively distinct from each other, although the malignant set is less tightly clustered. 63

Figure 6: LDA projection in two-dimensional space of modified breast cancer dataset. Meanings of the clusters are the same as Figure 4. The benign tumor data is unchanged. Additional malignant tumor data has been randomly generated with a convex hull around the original malignant tumor feature cloud. 64

Figure 7: LDA projection of cars dataset in two dimensions. The large, red cluster (right) are undesirable cars. The green (left, top), blue (left, bottom), and yellow (far left) are the various levels of “desirability”....................65

Figure 8: LDA projection of cars dataset to three dimensions. Colors are the same as Figure 7..66

Figure 9 - Yale Extended B dataset projected to two dimensions using t-SNE. The t-SNE project seeks to preserve “clusters” of points. High degrees of visual separation on this graph tends to correlate with an original embedding space that has distinct regions for different classes of features. Each color corresponds to a single subject. Some of the 28 colors are very similar; each class is tightly clusters in this dataset....................67

Figure 10: Yale Extended B dataset projected to three dimensions using t-SNE. The interpretation of this graph follows the same logic as Figure 9. When inspected interactively, the clusters here have a visual margin between them. 68

Figure 11 Yale Extended B dataset projected into two dimensions using non-metric MDS. MDS is a non-linear mapping from high dimensional space that seeks to maintain distance between points. Unlike t-SNE in figures 9 and 10, it is possible to map newly introduced points in the original feature space into an MDS projection. As is apparent from the high levels of overlap between classes in this image, MDS does not preserve class separation on our dataset; so MDS isn’t particularly useful in this application....................68

Figure 12: An LDA projection into two dimensions of the Yale Face Database B. New points in the embedding space can be efficiently mapped into the low dimensional space using only additions and multiplications (the projection is a dot product with the basis vectors of the new space). This makes an LDA projection a natural choice for HE classifiers if enough separation between classes is preserved. 69

Figure 13 An LDA projection into three dimensions of the Yale Face Database B. There is visual separation between many of the classes in the dataset in this projection, and LDA supports efficient projection from the original embedding space using only HE-supported operations. This makes the LDA projection an attractive choice for a dimensionality reduction technique in this trial. 70

Figure 14 – the precision (solid) and error rates (dotted) of plaintext KNN (red) and StaticKishNN blue) on the 3d LDA projection of the Yale Face Database. The X axis is an unordered permutation of values of $k \in \{2,4,8,16,32,64,128,256\}$ and $grid\_size \in 20, 40, 80, 120, 160, 200, 300$. The design choice of building the static k-ishNN classifier to dismiss ambiguous results means that, though it has a lower recall rate than classic KNN, it also has a consistently lower false classification rate...........73

Figure 15: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the cars dataset using a 2D LDA projection...73

Figure 16: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the cars dataset. Because the cars dataset is already on a grid, the grid size has no effect on output............73

Figure 17 Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 2d LDA projection of the Wisconsin Breast Cancer dataset...74

Figure 18: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 3d LDA projection of the Wisconsin Breast Cancer dataset...74

Figure 19 Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 2d LDA projection of the extended breast cancer dataset In contrast to Figure 17, here k-ishNN's performance decays rapidly as k and grid size decrease. 74

Figure 20: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the faces dataset using a 3D LDA projection...75

Figure 21: Performance of KNN (red), Static$k$-ishNN (blue), and k-ishNN(green) on the faces dataset using a 2D LDA projection...75

Figure 22: Average precision of selected algorithms on face dataset in multiple dimensions. The OpenFace implementation of FaceNet (blue) serves as a benchmark. The KNN approximation algorithms were only run on 2 and 3 dimensional datasets because the computation demands of those approaches rapidly become impractical for datasets with large numbers of training features, such as the facial recognition database used here. 76

Figure 23 A one-to-one comparison of the OpenFace and ppSVM classifiers on vectors projected into an identical embedding space. OpenFace uses the scikit-learn [57] SVM classifier. The accuracy of the two algorithms begin to converge as the number of dimensions increases. 76

Figure 24 Precision of ppSVM and scikit-learn's wrapper of LibLINEAR [50], [57]. Though the two lines appear to overlap, the computation noise in ppSVM has caused a very slight decrease in precision on the test dataset. Both achieve nearly 80% accuracy in single element cross validation on a ten-dimensional projection of our facial recognition embedding space. 77

Figure 25 – The runtime of classification tasks using the k-ishNN algorithm both with and without precomputation. Colors are displayed based on a normalized log scale (dark green is a fraction of a second per query, dark red is over a minute per query). Timing was computed by taking the average time per query of 300 queries during cross-validation. Measurements were taken on a docker container running on a 2016 MacBook Pro with a 2.9 GHz Intel Core i5 processor and 8GB of LPDDR3 RAM. The data contains some clusters of outliers caused by the thermal throttling by the host OS. The algorithms that use precomputation can run in seconds instead of minutes on large grid sizes. As expected, runtime does not vary as a function of k. 78

Figure 26 – The runtime of biometric classification tasks using the k-ishNN algorithm both with and without precomputation. Colors are plotted on the displayed based on a normalized log scale. The runtimes were collected using the same system as the one described in Figure 25. On this dataset the KishNN algorithm exceeded the available parallelism, causing significant slowdowns.......................79

Table of Algorithms

Algorithm 1 – $GenerateThresholdFunction(S, params)$ 52

Table of Protocols

Protocol 1 - k-ish Nearest Neighbors Classifier 40

Protocol 2 – k-ishNN with preprocessing 51

Protocol 3 – Privacy Preserving SVM (ppSVM) classification 56

Acknowledgements

Daniela – for originally encouraging me to pursue this degree, for being just as supportive and encouraging a decade later, and for years of guidance and advocacy.

Hayim – for taking on a surprise mentee and spending countless hours on Zoom teaching me cryptography and providing feedback on my approaches and ideas.

Brandi, Vera, and the rest of the course 6 team – for taking the time to create a path for my return after a decade away.

Emily – for dealing with a summer of sleepless nights, for keeping me healthy and sane though the final three-week crunch, and for learning an entire field so that i could have someone to discuss problems with.

Chapter 1: Introduction

This thesis investigates whether biometric recognition can be performed on encrypted data without decrypting the data.

1.1 Motivation

As of the writing of this thesis in Summer 2020, bills proposing a moratorium on the governmental use of “facial recognition and biometric technology” are under consideration in both the U.S. House of Representatives and Senate [2], [3]. In a press release accompanying the introduction of the legislation, the co-sponsors of the bill published a press release calling facial recognition a “grave threat to our privacy” that has “no place in our society”.

The concern makes sense. Biometric recognition is a new technology that still suffers from bias [4] (though efforts exist to mitigate these effects [5]), and it decreases the costs of surveillance in ways that can and have been abused [6].

Yet facial recognition, and biometric recognition more generally, is an important part of our modern interactions with technology. Facial recognition is used to help us log into devices [7] and to organize our photos. It’s used by governments to improve pedestrian safety [8], to search for missing and abducted children [9], to diagnose diseases [10]. Biometric classification more broadly is used to enable new modes of interaction with technology, such as voice command-driven interfaces, and increase cybersecurity as people increasingly move their lives online [11].

It seems odd that a bill proposing a ban on facial recognition can gain support from a country's political leadership at the same time it is advertised as a key feature on the most popular phone in the country. The tension between these two positions is a question of agency and trust. By imbuing technology with the ability to recognize people, we give the providers of that technology the ability to closely monitor our lives. We face the problem of balancing a desire for privacy and control of our data with a desire to use increasingly "smart" technology.

1.2 Homomorphic Encryption (HE)

One particularly exciting technical approach to the problem is homomorphic encryption. Originally theorized by Ron Rivest in 1978 [12], homomorphic encryption is a technique for encrypting data in a way that enables computation on that data without access to encryption secrets, resulting in computation and outputs that remain encrypted.

Over the past ten years, innovations in the implementation and parallelization of homomorphic encryption has enabled the development of a growing range of practical uses of the technology. In 2009, Craig Gentry described in his Ph.D. thesis an scheme capable of performing homomorphically encrypted computation that supports both addition and multiplication of encrypted cyphertexts [13]. In this thesis, Gentry's scheme perform noisy operations (eventually complete destroying the data) capable of computing its own encryption/decryption circuit and at least one more operation before being overwhelmed by noise. By “bootstrapping”, or repeatedly encrypting the results of operations, Gentry’s thesis described the first scheme theoretically capable of performing basic homomorphically encrypted operations. Though mathematically elegant, this solution required a practically infeasible amount of additional computation to perform useful operations.

In 2012 Gentry and his collaborators designed a scheme (referred to as the “BGV” scheme after the author names) that limits the amount of noise per operation, allowing for less aggressive bootstrapping [14].

Implementations of this scheme required implementing a lattice-based cryptosystem both in plaintext and within the encryption scheme; a mathematically and programmatically complicated task. As with any encryption software, correctness is measured not only by the ability of the algorithm to perform computations correctly, but also it the implementation’s data leakage, which can take years of testing to fully assess. Because of these complexities, it was nearly three years before any group released a practical opensource implementation of BGV. IBM’s HElib was the first to do so in 2015, followed shortly after by Microsoft’s Simple Encryption Algorithm Library (SEAL) [15], [16].

These easily used implementations of homomorphic encryption schemes have led to a wave of new research into the practical applications of the technology. Operations are still expensive (a computer running HElib, the most popular HE implementation, takes almost 100,000 times longer to perform an encrypted multiplication of two 32-bit numbers than it would without encryption), and limited (BGV, for example, only supports the computation of addition and multiplication within $\mathbb{Z}_p$ for some prime constant p). Practical development of more complex operations such as machine learning algorithms are still an area of active research.

1.2.1 Homomorphically Encrypted Classification Algorithms

This theoretical contributions of this thesis pick up where, Shaul, Feldman, and Rus leave off in their 2020 paper "Secure k-ish Nearest Neighbors Classifier" [1]. In this work, the authors describe a system for efficiently performing the k nearest neighbors algorithms (KNN)–an algorithm that classifies a datum based on the most similar data in some reference database–in a homomorphically encrypted context.

Shaul describes a technique for performing a homomorphically encypted comparison between two numbers, which he uses to compute the result of a "coin toss" by comparing a random plaintext with an encrypted cyphertext. Using these coin tosses, one can approximate the sum of any monotonic, invertible function on an array of encrypted cyphertexts by adjusting the bias of the random plaintext selection.

Using these new algorithms, Shaul finds the nearest neighbors of a query point by approximating the mean and standard deviation of the distances between some set of reference points and a query point, which can be used to approximate the threshold around which a query points' nearest neighbors reside. This algorithm is discussed in length in Chapter 3.

The coin toss and approximation schemes discussed in this work have the potential to enable the translation of a wide range of algorithms into HE schemes. The KNN implementation used as an example, however, does not fully account for rounding and overflow errors, meaning it is only effective on carefully selected datasets that accommodate these errors (this is discussed further in Chapter 3). The algorithm also requires computations and comparisons (which have a runtime which grows with the number of reference points) on every point in the reference database. As databases grow large, as they do in biometric classification tasks, the runtime of a homomorphically encrypted KNN implementation becomes intractable.

1.3 Contributions

Using the work described in [1] as a starting point, this thesis tests the existing work on biometric recognition tasks, analyses reasons that the current implementation fails, and then describes the modifications necessary to perform biometric recognition using a homomorphically encrypted KNN approximation. We also explore an alternative algorithm that corrects for the growth in computation overhead accompanying large and complex reference databases. Both of these approaches apply the insight that moving as much computation as possible into cached, plaintext precomputation allows real-time, homomorphically encrypted algorithms to run with improved performance and accuracy.

There are three main contributions in this thesis, two algorithms for efficiently performing biometric classification and recognition, and a collection of experiments demonstrating their effectiveness.

1.3.1 Algorithmic Contributions

Improvements to the efficiency, accuracy, and generalizability of the *k*-ishNN algorithm introduced in [1]**.** Using this precomputation technique, we describe in Section 4.1 a new algorithm based on the prior work by Shaul et. al. that displays improvement in efficiency and runtime against all of the prior work's test metrics. We demonstrate that these improvements allow the algorithm the precision necessary to perform biometric recognition tasks.

The *k*-ishNN algorithm works by approximating statistical properties of a database of examples to perform classification tasks. By replacing the model of the space used by the algorithm (a modified Gaussian distribution) with a simpler uniform distribution, we are able to move much of the algorithm into plaintext precomputation, which both decreases the computational demands of formulating a response to a query, and increases the experimentally measured accuracy of the algorithm's classifications. These improvements enable the use of the k-ishNN algorithm to perform biometric classification tasks, which require a greater degree of precision than the approximation schemes and model than the original implementation of the algorithm allows.

A homomorphically encrypted support vector-based classification technique. Because KNNs become less efficient as the amount of example data grows, in Section 4.2 we also describe and implement a homomorphically encrypted one-versus-all multiclass SVM that has computational demands that are independent of the size of the reference data. The SVM is implemented using the one-to-many training approach outlined in [17], the most salient property of which is the ability to classify a point by pre-computing a single basis vector for each class. If one projects a query point onto these vectors, the projection with the largest absolute value corresponds to the class of the query. Projections are relatively easy and efficient to compute in HE, and finding the maximum value in an array is demonstrated in the foundational work [1], enabling a simple implementation.

1.3.2 Experimental Contributions

In Chapter 5, we perform a series of experiments testing the accuracy and efficiency of these algorithms against both the original datasets used in a testing and on a new dataset prepared as a model of the challenges faces in biometric recognition tasks. We show that the new algorithms presented in this thesis not only achieve a level of improvement performance necessary for biometric recognition, but that they achieve accuracy decent accuracy rates on a motivating biometric recognition dataset. We also show that our modifications to the $k$isnNN algorithm introduced in [1] result in an order of magnitude improvement in runtime under practical conditions.

1.4 Technology vs. Regulation

In the bill described at the top of this chapter, Senator Markey seeks to address the abuse of biometric recognition systems with legislation, banning technologies that might be misused. Another approach to the problem of misuse of powerful technologies is to build technical safeguards into the technology itself. If classification is useful, but data collection might be abused, then systems that can extend classification services to also provide strong, technically enforceable guarantees about what data is collected and how it is stored could limit the potential downsides of new innovations without restricting their benefits.

Two of the mechanisms for preventing abuse of collected data are “notice and consent” and breach notification requirements, which levy huge fines against companies that fail to protect their user’s personal data or use it in a manner that the user has not consented to. There is evidence these mechanisms, which largely rely on self-enforcement, are ineffective at actually restricting data use or data protection [18], [19].

Especially in contexts where consumers might be especially concerned about data privacy (such as biometric recognition) or misuse (such as when interacting with a government or untrusted authority), technical safeguards provide stronger guarantees and agency than policy safeguards, which at some level always requires people to either trust organizations will comply with policies, or authorities will enforce those policies. Since the Snowden revelations in 2013 [20], [21], this trust-based approach has become less effective, especially in international markets.

This thesis describes technical systems that allow users to send biometric data to untrusted servers for analysis without requiring the user trust the provider to have policies in place to avoid storing or accessing that data. The systems also allow a provider to build services without exposing themselves to the liability risks inherent in handling user data.

Rather than adopt increasingly heavy-handed restrictions on the functionality of technology, legislators can use results like these recent innovations in homomorphic encryption and the extensions in this thesis to restrict where data flows and how it is used, rather than placing broad moratoriums on entire technical fields with far reaching ranges of use and utility.

1.5 Goals

We hope that this thesis' demonstration that fully encrypted biometric recognition can be performed efficiently serves both as a starting point for the development of technical alternatives to legislative privacy protection, and as a technique for service providers to continue to deliver the useful services biometric recognition enables without relying on untestable trust of their users or raising the legitimate privacy concerns associated with biometric data collection.

Chapter 2: Related Work

This thesis describes a system for performing biometric recognition tasks on encrypted data in order to improve privacy guarantees provided by these systems. To do this, we incorporate techniques and standards from the field of biometric recognition and adapt them to fit into the computational models needed to perform encrypted computation. This chapter provides an overview of previous work in biometric recognition, describes some of the previous work in developing privacy-enhancing technologies, and introduces prior work in homomorphic encryption, the specific technology used in this thesis to enhance the privacy of biometric recognition systems.

2.1 Biometric Recognition

Biometric recognition systems fall largely into two categories. *Verification* systems take a claimed identity and some sensor input and determine whether the input corresponds to the claimed identity. The fingerprint sensor on a modern smartphone is an example of a biometric verification system. *Identification* systems seek to identify which member of a population triggered a set of sensor measurements. Automatic photo tagging is an example of an identification system.

Biometric recognition systems span a wide range of applications, methods, and measurements.

Facial recognition is a common and widely used biometric recognition technique. Historically, facial recognition has relied on extracting recognizable features in images. For much of the late 1990s and early 2000s facial recognition relied on “eigenfaces”, vectors in extremely high dimensional image space that have face-like properties [22]. Several test datasets for facial recognition were developed during this era, notably the popular Yale Facial Recognition dataset and Yale Extended Facial Recognition Dataset B [23].

When ImageNet established the efficacy of deep convolutional neural networks for image classification in 2011 [24], the use of embedding spaces–vectors close to the end of a trained classifier–became a powerful tool in reducing images of faces into a computationally tractable number of dimensions while retaining enough data to differentiate faces. The FaceNet algorithm built a facial embedding space that maximizes the distance between images of different people and minimizes the distance between images of the same person [25]. It does so by using a triplet loss function, where a neural net is trained on three images, two of which are of the same person, and the side of resulting triangle connecting the two matching faces is minimized relative to the size of the triangle.

Other biometric recognition systems, such as those recognizing fingerprints and irises, also use embedding spaces. Recognition of sequential data like voice and gaits using deep learning often uses recurrent neural networks; this sequential recognition work is less mature than its convolutional counterpart and we treat at outside the scope of this initial investigation of biometric classification.

Recent concerns about the fairness of machine learning, especially as practical applications in policing and surveillance causally connect the underlying bias in training sets to real work consequences [26], has led to the investigation of algorithmic methods for detecting and mitigating bias in the training of these algorithms [5].

2.2 Privacy Enhancing Technologies (PETs) and Encryption

There are a wide range of technologies that seek to enhance privacy. Heurix et. al. describe a taxonomy that categorizes these technologies in [27], and also provide examples of technologies in most of the permutation of categories in that taxonomy. The taxonomy is useful in order to identify how this thesis fits into the larger context of PETs, so we summarize it here:

At a high level, privacy technologies seek to obscure a user's identity (or, less often, their content or observable behavior). PETs seek to obscure this information by either denying access, disassociating that information from other context, or rendering it indistinguishable from other information. This is achieved either through policy measures (security) or by performing algorithmic obfuscation on the underlying data (cryptography). Many PETs focus on information at a particular part of an algorithm, for example in transmission or during processing.

PETs are analyzed under a “threat model”, or the conditions under which data must be protected. This thesis focuses on cryptographic approaches (untrusted server) with an “honest-but-curios” server the follows the given protocol faithfully but tries to extract information from the protocols execution. We do not consider a “malicious server”, which might ignore or subvert the protocol itself.

Examples of other technologies that have a similar goal of obscuring the identity and data of a user during computation includes most digital encryption schemes. Private machine learning, such as secure encrypted federated machine learning [28] is a recent example in this category. Modern work in this space seeks to provide the scale and aggregate learning that comes from computation “in the cloud” while still maintaining the privacy and data agency provided by running scripts on a local device. Federated secure machine learning uses encrypted aggregation techniques [29] to accumulate enough data to perform stochastic gradient decent on the weights of a machine learning system [28]. This is the technology that allows Google to train AndroidOS’s keyboards’ predictive text without collecting the actual text or typing information of individual users.

2.3 Homomorphic Encryption

Homomorphic encryption (HE) was originally theorized in 1978 by Ron Rivest [30]; a feasibility proof of HE was first described in 2009 by Craig Gentry in his PhD thesis [31]. Gentry extended the work for practical use in 2011 in collaboration with Brakerski and

Vaikuntanathan (the "BGV" scheme) [14]. The BGV homomorphic encryption scheme allows for addition and multiplication operations performed in a ring $\mathbb{Z}_p$. In 2016 Choen, Kin, Kin and Song (CKKS) described a homomorphically encrypted approximation scheme [32] that can efficiently run the same operations on approximations of numbers in $\mathbb{R}$.

IBM's HElib [33] (released 2015) implements the BGV and CKKS schemes with a number of optimizations. Microsoft Simple Encryption Algorithm Library (SEAL) [34] similarly provides a wrapper around the BGV and CKKS schemes. A number of c++ and python wrappers exist for both libraries [35], [36].

The release of these libraries has accelerated research into HE implementations of algorithms. Results demonstrating the computation of hyperplane thresholding, sigmoids, and ReLU in HE have begun to lay the groundwork for running trained deep learning systems in an HE context [37]. Efficient homomorphic facial verification tasks have been implemented by computing distances from reference points in an encrypted embedding space [38].

Chapter 3: Problem Formulation and Foundational Work

This chapter provides a description of the biometric recognition task we intend to solve, our development of a specific motivating problem to test potential solutions against, and the formal notation used in the rest of this thesis. It also describes the previous work that this work extends, and explores why that previous work is unable to perform encrypted biometric recognition without modification.

3.1 Problem Formulation

Previous work in homomorphic machine learning has achieved binary classification of a small number of classes or simple feature vectors, demonstrating the feasibility of algorithms generally. In order to enable biometric recognition, we select a motivating real-world biometric recognition task and then explore the efficacy of existing homomorphically encrypted algorithms' performance on that task, potential extensions to improve that performance, and out-of-band precomputation steps that can reduce the problem to one that is solvable using existing work.

3.2 Motivating Problem: Facial Recognition

We consider a facial recognition task as a motivating problem. Previous work in homomorphic facial recognition either requires many rounds of online communication and several megabytes of online communication [39] or constrains the problem to face *verification*, performing a binary one-vs-the-rest classification of a face against a claimed identity [40].

We use the Yale Extended Facial Dataset B [23], which contains photographs of 28 people in a variety of poses and lighting conditions. Using OpenFace [41], an open-source implementation of the FaceNet [25] algorithm, we map images of faces to a 128-dimensional embedding space. For the sake of algorithmic clarity, we exclusively perform experiments on these embedding vectors and assume that the client has the ability to map facial images into the embedding space. This can be trivially achieved by either sharing the weights of the trained network or by performing transfer learning to compress the network for low-memory client applications.

We also consider two dimensionality reduction techniques to decrease the number of dimensions and improve the efficiency of our algorithms, which depend on vector operations that scale with the number of dimentions. First, we use LDA to down sample the embedding space into 2, 3, 5, or 10 dimensions. Second, we retrain FaceNet, replacing the final layer with a vector of size d, the target dimentionality.

The relatively high number of classes, high dimensionality and precision of the feature vectors, and variable feature density of the embedding space make this dataset a motivating embodiment of many of the challenges of biometric recognition. State of the art biometric systems often have thousands to millions of classes and feature vectors (e.g. [42] has several million faces); this work is a step towards training against these more general datasets.

3.3 Preliminaries

Most of the HE operations in this thesis are performed following the BGV scheme [14]. This scheme supports addition and multiplication of cyphertexts in some integer ring $\mathbb{Z}_p \coloneqq [0, p)$ where p is a prime number selected at key generation time. We use $+$ to denote addition and $\cdot$ to denote multiplication. We denote encryption and decryption using public key pk and secret key sk with

$$Enc_{sk}(x) = [\![x]\!]_{pk} \text{ and } Dec_{sk}\left([\![x]\!]_{pk}\right) = x$$

3.3.1 Notation and pre-defined functions

For any plaintext variable x, we denote that variable encrypted with private key pk as the cyphertext $[\![x]\!]_{pk}$. When there is only one possible private key, we omit the subscript for clarity.

Some algorithms in this paper use approximation schemes. We denote an approximate variable as $T^* \approx T$.

We use several algorithms that are defined in [1]. Specifically, where $\mathbb{P}_{f,p}: \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ is the polynomial interpolation of some function f rounded to the nearest integer and p is the prime used by BGV, we use the comparison function

$$\text{isSmaller}_\text{p}([\![x]\!],[\![y]\!]) = \mathbb{P}_{f,p}\left(\frac{p}{2} + x - y\right) \; where \; f(x) = \begin{cases} [\![1]\!] \; if \; x > \frac{p}{2} \\ [\![0]\!] \; otherwise \end{cases}$$

We also use Shaul et. al.'s implementation of ArgMax:

$$ArgMax_c(C) := \sum_{j \in C} j \cdot \prod_{i \neq j} \text{isSmaller}\left(\text{C}_\text{i}, \text{C}_\text{j}\right)$$

Further, we use a trick described in that paper to compute the $l_1$norm between points in a feature space by computing the absolute value of the difference between each element in the vector:

$$dist_p(a, b) = \sum\left(1 - 2\text{isSmaller}(\text{a}_\text{i}, \text{b}_\text{i})\right)(\text{a}_\text{i} - \text{b}_\text{i})$$

We use capital letters to denote arrays of values, and lowercase letters to denote scalars. We use a lowercase letter with a subscript to denote a single item from an array. For example, x_i is the i^{th} element of the array X.

We denote cached constant coefficients use in functions with the Greek letters α and β.

3.3.2 Variable Names

We use the follow variable names across all algorithms and protocols in this thesis:

- d – the number of dimensions in the feature space of a training set.
- p – the prime number used in the ring.
- S – a database consisting of n vectors of size p where each element $s_i \in S$ represents a single training feature for use in classification
- q – a query sent by a client to a server for classification. A query is a d dimensional vector where $q_i \in \mathbb{Z}_p$
- X – an array of scalars $x_i \in \mathbb{Z}_p$ *and* $x_i \in X$. x_i is the distance between q and s_i
- μ and σ – the mean and standard deviation of a distribution, respectively.
- k – in KNN-like algorithms, the desired number of neighbors to consider when determining the class of q.
- κ – the number of points actually used in a KNN-like algorithm. In most cases $\kappa = k^*: \approx k$, but this is not necessarily always true.
- C – a list of positive integers where $c_i \in C$ is the class associated with the feature vector s_i. We also occasionally use the notation $class(s_i) \stackrel{\text{def}}{=} C_i$ to represent the class associated with a vector.
- $class_q$ - the predicted class of q
- Φ is the standard normal function. Φ^{-1} is the inverse normal.
- $\lfloor \ \rceil$ - is the notation we use to denote rounding to the nearest integer.
- T is a number with the property that, for x_k, the k^{th} largest element in X, $x_k \leq T \leq x_{k+1}$. This number defines the length of a radius around q that contains exact k features.

- A – a placeholder array for intermediate variables. Subscripts to A provide a note on what the values in A represent.

3.3.3 Base-p representation

Some algorithms use base-p representation. Base-p representation is a technique for avoiding large values of p, which can greatly increase computational overhead. It functions by converting some number $v \in [0, p^2)$ into a "two digit" number in base p which could be expressed as the big-endian tuple:

$$\text{base} - p \text{ rep. of } (v) := \left(\left\lfloor \frac{v}{p} \right\rfloor, v \bmod p \right), \text{where } v \in [0, p^2)$$

We denote the more significant figure in base-p notation as $high_p(v)$ and the one's digit as $low_p(v)$. If p is unambiguous in context, we omit the subscript.

3.4 Foundational Work

This work builds on top of insights outlined by Shaul et. al. in in [1] in which they develop a secure classification algorithm, the "k-ish Nearest Neighbors Classifier" (k-ishNN). k-ishNN describes a homomorphically encrypted algorithm which computes the output of the KNN algorithm on a query vector q and a feature space S with low communication requirements and running time.

3.4.1 Overview of *k*-ishNN

This section describes the intuition behind the *k*-ishNN algorithm's implementation and limitations. The remainder of Section 3.4 discusses those same topics in more technical detail.

Stated in plain English, the *k*-ishNN algorithm performs the following operations:

(1) A server is given a set of labelled example points

(2) A client gives the server a "query" point, and asks what label makes the most sense for that query.

(3) The server finds the most common label amongst the k nearest neighbors of the query to send back to the client. However, because the query is encrypted, the server does not know which points are closest. Instead, the server:

   a. Uses a series of novel techniques to estimate the radius of a "bubble" around the query that contains approximately k points. This computation is encrypted.

   b. The server computes the number of points within the bubble corresponding to each label. This computation is also encrypted.

   c. The server computes the label with the largest counts in the previous step and sends it back to the client. This entire process is encrypted, so the server doesn't know what the final result is.

(4) The client decrypts the server's result and learns the likely class of the query point.

The core intuition is that the points close to the query are likely similar to the query, and an approximate value of the threshold will likely yield the same suggested classification as the precise value of the threshold. The highest fault risk in this implementation of the algorithm lies with the techniques used to keep the operations in step (3) encrypted: if the bubble ever grows or shrinks to a point of over- or under-inclusion, the algorithm will fail.

The technique used by k-ishNN to estimate a threshold assumes that the reference points conform to a normal distribution, and uses that distribution to compute a target threshold. In practice, this approach often results in a non-sensical negative threshold (see Figure 1) which overflows to include most or all of the reference points, resulting in the selection of the most common label *in the entire space* as a class.

If the situations most likely to trigger an overflow happen to occur in regions corresponding to the most common class in the space (as is the case in the two datasets used in [1] for testing), this error might go undetected. This problem largely stems from the use of a statistical model that can produce negative estimated values for the threshold. In Section 4.1 we describe a modified form of the algorithm that eschews the assumption that example data follows a Gaussian distribution.

3.4.2 k-ishNN in more detail

The k-ishNN algorithm efficiently performs a homomorphically encrypted approximation of the output of the KNN algorithm on a query q and feature space S.

It achieves this using a novel technique which can approximate any bounded, monotonically increasing, invertible function f by performing n encrypted "double blinded coin tosses" with bias $\frac{f(\llbracket x \rrbracket)}{m}$ for an arbitrary cyphertext $\llbracket x \rrbracket$ and scaling factor $m > f(\llbracket x \rrbracket)$, the sum of which converges to $f(\llbracket x \rrbracket)$ without requiring large intermediate values.

The approach used by k-ishNN is described in Protocol 1 - k-ish Nearest Neighbors Classifier.

It starts by computing a distance array $\llbracket X \rrbracket$ such that:

$$x_i \coloneqq dist(q, s_i)\ \forall\ s_i \in S$$

It then uses the approximation technique outlined above to compute the mean and standard deviation of the distribution. The algorithm assumes that X follows a continuous Gaussian distribution and precomputes $\Phi^{-1}\left(\frac{k}{n}\right)$, allowing for the estimation of a threshold T^* around q which contains k features (the * denotes an approximated value):

$$\llbracket T^* \rrbracket = \llbracket \mu^* \rrbracket + \Phi^{-1}\left(\frac{k}{n}\right) \cdot \llbracket \sigma^* \rrbracket$$

Using this threshold and the polynomial expansion of a sigmoid around $\frac{p}{2}$, this threshold can be used to compute an array C where $C(c)$ represents the number of features within a radius of T^* from q which are of class c. By performing a series of comparisons on this resulting vector, k-ishNN computes $ArgMax_c(C(c))$, the most common class in the feature space within the threshold.

Protocol 1 - k-ish Nearest Neighbors Classifier

Shared Input:	integers $\boldsymbol{p},\ \boldsymbol{d}, \boldsymbol{c} > \mathbf{1}$
Client Input:	a point $q \in \mathbb{Z}_p^d$ and a security parameter λ
Server Input:	integers $k < n$
	points $s_1, \dots, s_n \in \mathbb{Z}_p^d$
	A matrix $M \in \{0,1\}^{n\times c}, s.t. M(i,j) = 1$ iff $class(s_i) = j$
Client Output:	$class_q \in [c]$, the majority class of κ nearest neighbors of q where $\frac{k}{2} < \kappa < \frac{3k}{2}$ with high probability

```
Client Performs:
1   Generate Keys $(\boldsymbol{sk}, \boldsymbol{pk})\boldsymbol{S} = \boldsymbol{Gen}(\mathbf{1}^{\boldsymbol{\lambda}}, \boldsymbol{p})$
2   $[\![\boldsymbol{q}]\!] \coloneqq \boldsymbol{Enc_{pk}}(\boldsymbol{q})$
3   Send $(\boldsymbol{pk}), [\![\boldsymbol{q}]\!])$ to the server
Server Performs:
4   for each $\boldsymbol{i} \in \mathbf{1}, \dots, \boldsymbol{n}$ do
5       $[\![\boldsymbol{x_i}]\!] \coloneqq \boldsymbol{computeDist}([\![\boldsymbol{q}]\!], \boldsymbol{s_i})$
6   $[\![\boldsymbol{\mu}^*]\!] \coloneqq \boldsymbol{approximate}\frac{\mathbf{1}}{\boldsymbol{n}}\sum[\![\boldsymbol{x_i}]\!]$
7   $([\![\boldsymbol{low}((\boldsymbol{\mu}^*)^2)]\!], [\![\boldsymbol{high}((\boldsymbol{\mu}^*)^2)]\!]) \coloneqq$ base-$\boldsymbol{p}$ rep. of $(\boldsymbol{\mu}^*)^2$
8   $([\![\boldsymbol{low}(\boldsymbol{\mu}_2^*)]\!], [\![\boldsymbol{high}(\boldsymbol{\mu}_2^*)]\!]) \coloneqq$ base-$\boldsymbol{p}$ rep. of $(\boldsymbol{\mu}_2^*)$
9   $[\![\boldsymbol{\sigma}^*]\!] \coloneqq \boldsymbol{approximate}\sqrt{\boldsymbol{\mu}^* - \boldsymbol{\mu}_2^*}$
10  $[\![\boldsymbol{T}^*]\!] \coloneqq [\![\boldsymbol{\mu}^*]\!] + \left\lceil \boldsymbol{\Phi}^{-1}\left(\frac{\boldsymbol{k}}{\boldsymbol{n}}\right)\right\rceil [\![\boldsymbol{\sigma}^*]\!]$
11  $[\![\boldsymbol{C}]\!] \coloneqq (\mathbf{0}, \dots, \mathbf{0})$
12  for each $\boldsymbol{c} \in \mathbf{1}, \dots, \boldsymbol{j}$ do
13      $[\![\boldsymbol{C}(\boldsymbol{j})]\!] \coloneqq \sum_{i=1}^{n} \boldsymbol{isSmaller}([\![\boldsymbol{x_i}]\!], [\![\boldsymbol{T}^*]\!]) \cdot \boldsymbol{M}(\boldsymbol{i}, \boldsymbol{j})$
14  $[\![\boldsymbol{class_q}]\!] \coloneqq \boldsymbol{ArgMax_c}([\![\boldsymbol{C}]\!])$
15  Send $[\![\boldsymbol{class_q}]\!]$ to the client
Client Performs:
16  $\boldsymbol{class_q} \coloneqq \boldsymbol{Dec_{sk}}([\![\boldsymbol{class_q}]\!])$
```

3.4.3 Issues and Assumptions in k-ishNN

k-ishNN has two critical assumptions that can cause it to have precipitous drops in performance. First, if the expected threshold is close to zero or the distribution is skewed right, T^* might be negative, meaning $[\![T^*]\!]$, which is computed *modulo* p, will overflow.

Because X is discrete and non-negative, even when X is roughly Gaussian it will still be truncated at 0 and have a sample standard deviation rather than the actual standard deviation assumed in the algorithm formulation. The sample standard deviation is by definition smaller than the actual standard deviation, and a left-truncated Gaussian distribution has right skew, meaning in practice these error modes are likely.

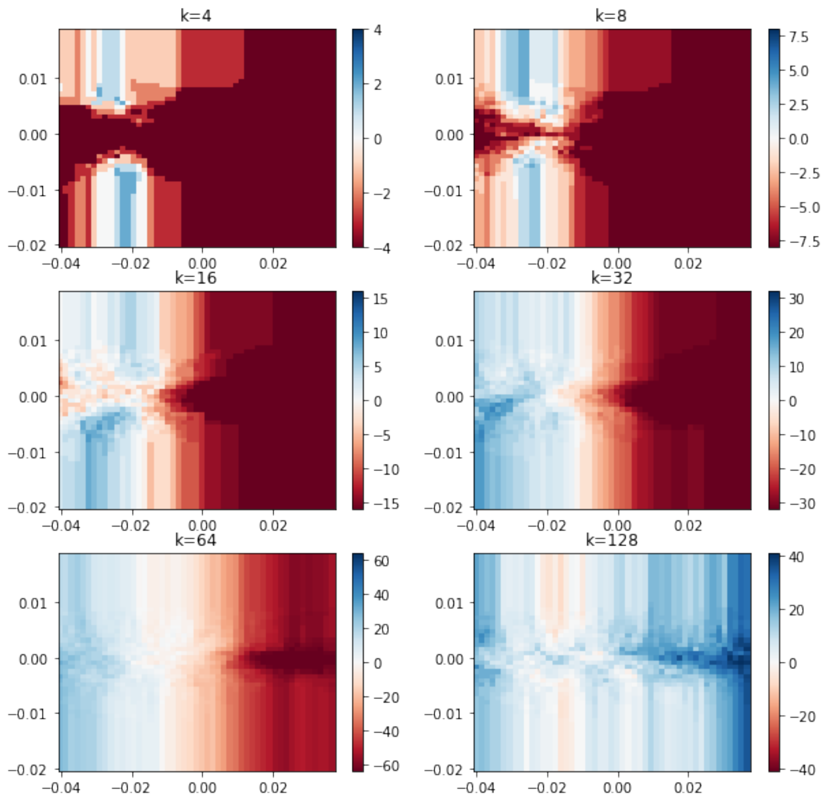


Figure 1 – the expected value of T^* assuming perfect approximations on a normalized 2d LDA projection of Wisconsin Breast Cancer database used in both this thesis and the original assessment of the k-ishNN algorithm. Red regions are areas in which T^* is expected to be negative, implying a greater than 50% chance of an overflow and an effectively arbitrary value of κ.

Figure 1 and Figure 2 depict the likelihood of a miscalculation of T^* due to overflow in the Wisconsin Breast Tumor dataset used to test the algorithm in both [1] and this thesis. In Figure 1, the red regions in the charts reflect areas where there is a greater than

50% chance that there will be an overflow in threshold computation. For even fairly large values of k, almost half of the query space is expected to have an overflow. Figure 2 shows the computation of κ in an unencrypted setting. There are large spikes at $\kappa = 0$, corresponding to a negative expected threshold which would result in overflow in the actual implementation of the algorithm. On a binary classification task like the one analyzed here; an overflow typically will result in a classification of the class most heavily represented in the entire feature space (because a high threshold from an overflow will capture "too much" of the space). In the case of the test in use, the region most likely to trigger an overflow corresponds to the most populous class, an eccentricity of the test dataset that results in a correct default classification when the system is in an error state. Section 5.4 describes a test in which this correlation is not present and finds that the accuracy of k-ishNN decreases sharply in that situation.

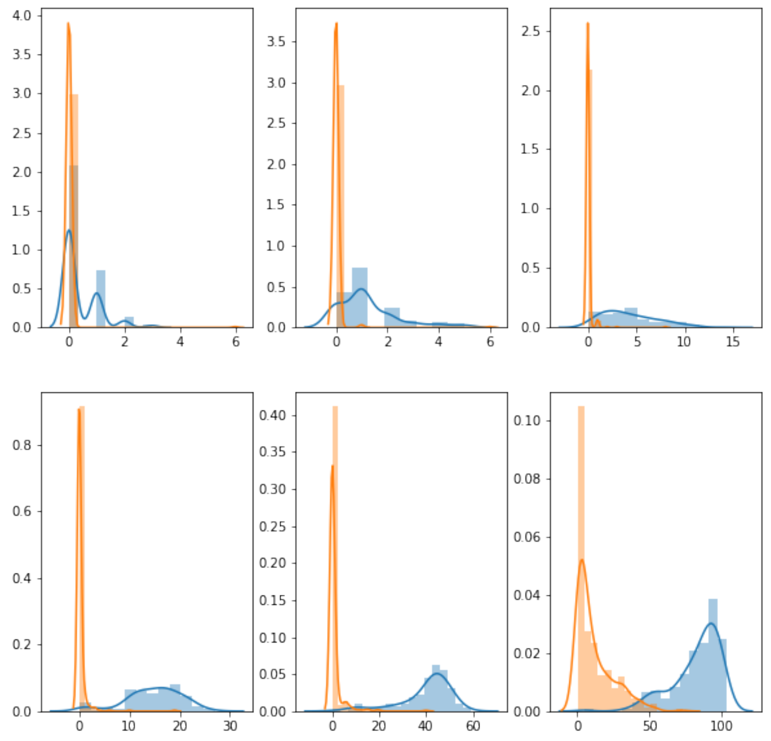


Figure 2 – The expected value of κ without overflow for benign (orange) and malignant (blue) tumors. The graphs show the value of κ along the horizontal axis. The charts, from left to right, top to bottom, represent the distribution of values of κ for target values of k = {2,4,8,16,32,64} respectively. The spikes at $\kappa = 0$ correspond to a high likelihood of overflow when computing a threshold. These graphs were generated using the 2d LDA projection of Wisconsin Breast Cancer database used in both this thesis and the original assessment of the $k$ishNN algorithm

The open source implementation of k-ishNN [43] mitigates this issue by computing a range of candidate thresholds across $(0, p]$, computing the class counts of each of these thresholds, and then applying filters to the resulting class counts to perform a classification. Section 4.1 is partially motivated by this practical mitigation of the shortcoming of the original algorithm.

In non-binary classification, the inaccuracy inherent in the Gaussian model for X and the risk of overflow cause k-ishNN to exhibit impracticably low accuracy. An example of this is shown in Figure 3 and discussed in more detail in Section 5.4.1.

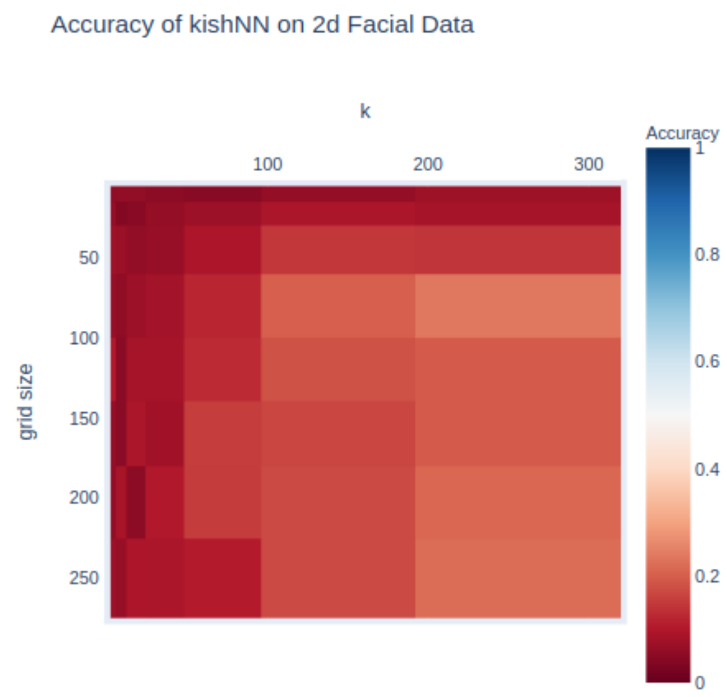


Figure 3 - the accuracy of the k-ishNN algorithm on a 2D projection of the motivating facial recognition dataset is low, even with large grid sizes.

Chapter 4: Technical Approach

Most homomorphically encrypted classification models to date have focused on binary classification [37], [44], [45] and left multi-class classification as a future extension. In the year preceding this thesis, some homomorphically encrypted classifiers, including the k-ishNN classifier have been built to handle multiple classes, but tests continue to focus on datasets with a small number of classes (e.g. [1] includes tests on a 4-class dataset, [46] focuses mainly on binary classification of text, but includes data from 4- and 14- class text datasets).

Biometric classification requires differentiation between a large number of complicated classes, in the initial tests outlined in Section 3.4.3 we found that the current implementation of k-ishNN lacks the precision necessary to perform this classification. This thesis provides two potential solutions to this issue.

Section 4.1 re-builds the formal definition of the k-ishNN algorithm (see Protocol 2) to isolate the part of the algorithm causing this loss of precision (Algorithm 1b) and then considers a simple replacement for that algorithms (Algorithm 1a), leaving further improvement for future work. This approach reframes work by Shaul et. al. [1] as the precomputation of a threshold generating function, and then uses the factory pattern [47] to load plaintext precomputation into that generator. The approach borrows the insight from machine learning literature that high-cost out-of-band precomputation can enable efficient performance at runtime.

Though the extension in 4.1 address the precision issue, it does not address the fact the number of reference points necessary to differentiate between classes rapidly becomes very large. Algorithms, like k-ishNN, that have a complexity that is a function of the size of the training database rapidly become computationally impractical.

Section 4.2 defines an entirely new approach to homomorphically encrypted classification that scales with the number of classes while also providing the necessary precision for multi-class classification tasks. To do this we developed a privacy preserving version of an SVM, leveraging recent innovations in homomorphically encrypted computation.

4.1 Generalization of k-ishNN: adding functional generators of candidate threshold families

To explore the threshold computation portion of the k-ishNN algorithm outlined in Protocol 1, we modify the protocol to accept an any arbitrary function of type $(X, pk) \rightarrow [\![T]\!]$, where T is some set of candidate thresholds. By performing some plaintext precomputation, the threshold function can be tuned for computable global properties of feature space. Protocol 2 describes this candidate selection formulation. The collection of implementations of Algorithm 1 describe several approaches to threshold computation.

4.1.1 Overview of k-ishNN with precomputation

Protocol 2 describes a modified version of k-ishNN (Protocol 1) that introduces a precomputation step. Algorithms 1a, 1b, and 1c describe three potential threshold computation schemes. 1a and 1b are tested in Chapter 5. 1c is included because it is functionally equivalent to the open source implementation of k-ishNN, ppKNN [43].

Like the original k-ishNN, the modified version of the algorithm seeks to identify the class of a query point q by finding the most common class in the k points in a feature space closest to q. It does this by computing a threshold distance from q and then computing the class of the points within that threshold distance from q.

At some point before interacting with a client or query, a server implementing Protocol 2 builds a function *generateThresholds* that returns a small number of candidate thresholds when given the distances between q and each reference point in feature space. This function is only computed once per feature space, and is reused between queries.

Algorithms 1a-1d describe several ways to construct *generateThresholds:* (1a) precomputes three constant values as candidate thresholds, and returns the identity function for those thresholds; (1b) pre-computes a first-degree polynomial based on the inverse Gaussian which approximates T and constructs an intermediate function that approximates the mean and median of the distance distribution for use as coefficients of that Gaussian. This algorithm causes Protocol 2 to functionally reduce to Protocol 1. (1c) performs the same pre-computation as (1b), and approximates the same coefficients, but then multiplies those coefficients by a combination of integers and the multiplicative inverse of 2 in such a way that a series of threshold "pairs" are created on opposite sides of the ring of possible threshold values (the encryption scheme requires that all variables are integers between 0 and p, so for each threshold t this scheme also creates a candidate threshold at $\left(t + \frac{p}{2}\right) \mathit{mod}\, p$). Algorithm (1c) is functionally equivalent to the experiments run in [1]. Algorithm (1d) selects 10 random thresholds. In initial experimentation, algorithms (1c) and (1d) produced indistinguishable accuracy scores; our experiments in Chapter 5 focus on providing data on the performance of algorithms (1a) and (1b).

Returning to Protocol 2: after generating a threshold generation function, the client encrypts and sends a query to the server (lines 2-4). While keeping all computation encrypted, the server builds a list of the distances between each reference point and the query point (lines 5-6), and then uses *generateThresholds* to generate several candidate thresholds (line 7). For each of these thresholds, the server counts the number of instances of points of each class within that threshold (lines 9-11), and then finds the most common class by identifying the class with the highest count (line 12). This results in an encrypted array of potential classes for q, which is sent back to the client (line 13). The client then decrypts the list (line 14), and, if there is a most popular class, accepts that as the class of the query (line 15). If there is a tie the behavior of the algorithm is undefined. (In Section 5.4.1 we show that if this undefined behavior is treated as a non-classification, Protocol 2 with Algorithm 1a can sometimes achieve lower error rates than the reference implementation of “normal” unencrypted KNN).

4.1.2 Discussion

The k-ishNN protocol precomputes the value of an inverse Gaussian and then scales and translates that value based on approximations of the first and second moment of the distribution X to estimate a threshold T which defines the radius from q to its k^{th} neighbor, $[\![T^*]\!] := [\![\mu^*]\!] + \left\lceil \Phi^{-1}\left(\frac{k}{n}\right) \right\rfloor [\![\sigma^*]\!]$ (Algorithm 1c or Protocol 1). The value of $\left\lceil \Phi^{-1}\left(\frac{k}{n}\right) \right\rfloor$ is only precise to within a single standard deviation, and values of $\sigma^* > \frac{\mu}{2}$ will often result in an overflow. We explore the implications of overflows on classification accuracy in Section 5.4.1. The open source implementation of k-ishNN [43] accounts for this issue by using the family of candidate thresholds $[\![T^*]\!] \coloneqq [\![\mu^*]\!] - \alpha([\![\sigma^*]\!]) + \beta([\![\sigma^*]\!] + \gamma) \bmod p$ for some hand-chosen parameters $\alpha \in \{1,2,10\}, \beta \in \{2^{-1}, 1\}, \gamma \in \{0,1\}$, selecting the threshold resulting in the best approximation of k by repeatedly applying $isSmaller$ to $sum(C)$ to identify a class.

Modeling the distribution of X as a cumulative density function (CDF) is difficult. If the density of feature vectors is non-uniform, a PDF approximating the distribution of points for an arbitrary query will be asymmetric, requiring operations other than stretching and translating a base CDF in order to accurately approximate the space. Symmetric functions run the risk of undetectable overflow, as encountered with the normal distribution in practice. We experimented with approximating the third moment using a double-blinded coin toss and encountered increased overflow. The log normal distribution is always positive and therefore immune to negative overflow, but requires a polynomial on σ to compute, resulting in high sensitivity to small approximation errors in σ^*.

For many applications the distribution of feature points is roughly uniform. In these cases, a threshold function $T(X, pk) \rightarrow [\![\mathrm{T}]\!]$ that returns some set of constants T might have improved accuracy over a noisy threshold computation. This is especially applicable to feature spaces in which an overflow is likely to cause a misclassification, as is the case in feature spaces with a large number of classes.

Given the method for data preparation in Section 3.2, uniform density can be induced by including a measure of the density in the loss function when training the vectorization scheme. Unified embedding algorithms, like FaceNet, are both common in biometrics and already tend towards using all of their embedding space, and therefore having roughly uniform volumes for different classes when the number of examples per class does not have high variation. Using this observation, we use the l_1 norm and the identity in Algorithm 1a as a motivating alternative pre-computed threshold generation function.

Protocol 2 – k-ishNN with preprocessing

Shared Input:	integers $\boldsymbol{p},\ \boldsymbol{d}, \boldsymbol{c} > \mathbf{1}$
Client Input:	a point $q \in \mathbb{Z}_p^d$ and a security parameter λ
Server Input:	integers $k < n$
	points $s_1, \ldots, s_n \in \mathbb{Z}_p^d$
	A matrix $M \in \{0,1\}^{n \times c}, s.t. M(i,j) = 1$ iff $class(s_i) = j$
Client Output:	$class_q \in [c]$, the majority class of κnearest neighbors of q where $\frac{k}{2} < \kappa < \frac{3k}{2}$ with high probability

```
    Server Performs:
1       $\boldsymbol{genereateThresholds} \coloneqq \boldsymbol{GenerateThresholdFunction}(\boldsymbol{S})$
    Client Performs:
2       Generate Keys $(\boldsymbol{sk}, \boldsymbol{pk})\boldsymbol{S} = \boldsymbol{Gen}(\mathbf{1}^{\boldsymbol{\lambda}}, \boldsymbol{p})$
3       $[\![\boldsymbol{q}]\!] \coloneqq \boldsymbol{Enc}_{\boldsymbol{pk}}(\boldsymbol{q})$
4       Send $(\boldsymbol{pk}), [\![\boldsymbol{q}]\!])$ to the server
    Server Performs:
5       for each $\boldsymbol{i} \in \mathbf{1}, \ldots, \boldsymbol{n}$ do
6           $[\![\boldsymbol{x_i}]\!] \coloneqq \boldsymbol{computeDist}([\![\boldsymbol{q}]\!], \boldsymbol{s_i})$
7       $[\![\boldsymbol{T}]\!] \in \boldsymbol{generateThresholds}(\boldsymbol{S}, \boldsymbol{pk})$
8       for $\boldsymbol{t} \in \boldsymbol{T}$
9           $[\![\boldsymbol{C}]\!] \coloneqq (\mathbf{0}, \ldots, \mathbf{0})$
10          for each $\boldsymbol{c} \in \mathbf{1}, \ldots, \boldsymbol{j}$ do
11              $[\![\boldsymbol{C(j)}]\!] \coloneqq \sum_{i=1}^{n} \boldsymbol{isSmaller}([\![\boldsymbol{x_i}]\!], [\![\boldsymbol{T}^*]\!]) \cdot \boldsymbol{M(i,j)}$
12          $[\![\boldsymbol{class_{q,t}}]\!] \coloneqq \boldsymbol{ArgMax_c}([\![\boldsymbol{C}]\!])$
13      Send $[\![\boldsymbol{class_q}]\!]$ to the client
    Client Performs:
14      $\boldsymbol{class_{q,i}} \coloneqq \boldsymbol{Dec_{sk}}([\![\boldsymbol{class_{q,i}}]\!]) \forall \boldsymbol{i}$
15      $\boldsymbol{class_q} \coloneqq \boldsymbol{ArgMax_c}\left(\sum_{class_{q,i}=c} \mathbf{1}\right)$ *in case of a tie, return null.*
```

Algorithm 1 – $GenerateThresholdFunction(S, params)$

Input: Database $\boldsymbol{S}$: $\boldsymbol{s_1, \dots, s_n \in \mathbb{Z}_p^d}$,
$\boldsymbol{params}$ – a set of parameters for the algorithm

Output: $generateThresholds$ – a function of type $(X, pk) \rightarrow [\![T]\!]_{pk}$, which given a set of distances returns a set of candidate thresholds

Algorithm 1a – Precomputed constant threshold (“Static$\boldsymbol{k}$-ishNN” introduced by this thesis)

params: $(\boldsymbol{k})$

```
1 $\boldsymbol{initialize(A_{minimum}, A_{average}, A_{maximum}) := empty\ arrays}$
2 for $\boldsymbol{s_i \in S}$
3     $\boldsymbol{D := sorted([l_1(s_i, s_j) \forall i \neq j])}$
4     $\boldsymbol{A_{minimum} \leftarrow d_k, A_{average} \leftarrow \frac{d_k + d_{k+1}}{2}, A_{maximum} \leftarrow d_{k+1}}$
5 $\boldsymbol{generateThresholds := (X) \rightarrow}$
      $\boldsymbol{[\![T]\!] := [\![\{\max(A_{minimum}), \min(A_{maximum}), median(A_{average})\}]\!]}$
```

Algorithm 1b – $\boldsymbol{k}$-ishNN threshold function (adapted from [1])

params: $(\boldsymbol{k})$

```
1 $\boldsymbol{\beta := \left\lceil \Phi^{-1}\left(\frac{k}{n}\right) \right\rceil}$
2 $\boldsymbol{generateThresholds := (X) \rightarrow}$
3     $\boldsymbol{[\![\mu^*]\!] := approximate \frac{1}{n} \sum [\![x_i]\!]}$
4     $\boldsymbol{([\![low((\mu^*)^2)]\!], [\![high((\mu^*)^2)]\!]) :=}$ base-$\boldsymbol{p}$ rep. of $\boldsymbol{(\mu^*)^2}$
5     $\boldsymbol{([\![low(\mu_2^*)]\!], [\![high(\mu_2^*)]\!]) :=}$ base-$\boldsymbol{p}$ rep. of $\boldsymbol{(\mu_2^*)}$
6     $\boldsymbol{[\![\sigma^*]\!] := approximate \sqrt{\mu^* - \mu_2^*}}$
7     $\boldsymbol{[\![T^*]\!] := [\![\mu^*]\!] + \beta [\![\sigma^*]\!]}$
```

Algorithm 1c – **ppKNN** threshold function (implementation used by [43])

params: (none)

```
1 $\boldsymbol{generateThresholds := (X) \rightarrow}$
2     $\boldsymbol{[\![\mu^*]\!] := approximate \frac{1}{n} \sum [\![x_i]\!]}$
3     $\boldsymbol{([\![low((\mu^*)^2)]\!], [\![high((\mu^*)^2)]\!]) :=}$ base-$\boldsymbol{p}$ rep. of $\boldsymbol{(\mu^*)^2}$
4     $\boldsymbol{([\![low(\mu_2^*)]\!], [\![high(\mu_2^*)]\!]) :=}$ base-$\boldsymbol{p}$ rep. of $\boldsymbol{(\mu_2^*)}$
5     $\boldsymbol{[\![\sigma^*]\!] := approximate \sqrt{\mu^* - \mu_2^*}}$
6     $\boldsymbol{[\![T^*]\!] := [\![\mu^*]\!] - \alpha([\![\sigma^*]\!]) + \beta([\![\sigma^*]\!] + \gamma) \bmod p \ \forall \alpha \in \{1, 2, 10\}, \beta \in \{2^{-1}, 1\}, \gamma \in \{0, 1\}}$
```

Algorithm 1d – random threshold function

params: (none)

```
7 $\boldsymbol{generateThresholds := (X) \rightarrow}$
8     $\boldsymbol{[\![T^*]\!]} := Enc(R)$ where $R$ is a list of random numbers $\in [0, \mathrm{p})$
```

4.1.3 Efficiency Analysis

Protocol 2 has a preprocessing efficiency of:

$$efficiency(generateThresholdFunction(\mathrm{S}))$$

The protocol's runtime consists of:

1. Compute $x_1, \dots, x_n$
2. Compute $T(0), \dots, T(t)$
3. Compute $C(0), \dots, C(c)$ for each $t \in T$
4. Compute $class_q$ for each $t \in T$

Step 1 is computed with n instances of $computeDist$ run in parallel.

Step 2 is a function of $generateThresholds$.

Step 3 runs $O(n * t)$ parallel $isSmaller$ subcircuits.

Step 4 is performed by t instances of the $ArgMax_c$ polynomial.

We require that t is some small constant of $O(1)$. This results in a general size a depth of:

$$\begin{aligned}\text{depth}(k\text{ishNN with precomputation}) \\ &= O\big(\text{depth}(computeDist) + \text{depth}(generateThresholds) + \log c \\ &\cdot depth(isSmaller)\big)\end{aligned}$$

and

$$\text{size}(k\text{ishNN with precomputation}) = O\big(\text{n} \cdot \text{size}(computeDist) + \text{size}(generateThresholds) + n \cdot size(isSmaller)\big)$$

When using approximations as outlined in Algorithm 1b and Algorithm 1c, $generate\,Thresholds$ instantiates $O(1)$ parallel $probabalisicAverage$ and polynomial circuits in parallel, resulting in a depth of $O(\log p)$ and a size of $O(\sqrt{p})$. The precomputation step only determines the value of a single numerical value, and therefore runs in constant time and space.

When using a constant precomputed threshold as outlined in Algorithm 1a: $generateThresholds$ performs no computation and returns $O(1)$ constants, resulting in $O(1)$ size and depth. Precomputation of a constant threshold by determining the range of values of κ that result in a precisise measurement of k for any point in feature space requires computing and sorting a distance array of size $O(n)$ a total of $O(n)$ times, resulting in a precomputation time of $O(n^2 \log n)$. The algorithm can have a parallelization factor of up to n, resulting in $O(n)$ parallel instances of runtime $O(n \log n)$.

4.1.4 Security

We assume a semi-honest server (one that follows the algorithm but tries to derive knowledge about the query during the run of the algorithm). Observe that two runs of the algorithm with two values q, the actual query, and q', a randomly generated query, would be computationally indistinguishable and, assuming that our encryption scheme is sound, contain identically meaningless values across the entire server-side computation.

```
Protocol 3 – Privacy Preserving SVM (ppSVM) classification
Shared Input:   integers $\boldsymbol{p},\ \boldsymbol{d}, \boldsymbol{c} > \mathbf{1}$
Client Input:   a point $q \in \mathbb{Z}_p^d$ and a security parameter $\lambda$
Server Input:   points $s_1, \dots, s_n \in \mathbb{Z}_p^d$
Client Output:  $class_q \in [c]$, the estimated class of $q$
    Server Performs:
1       $\boldsymbol{W} \coloneqq$ an iteratively computed matrix of weighted feature vectors such that $\overrightarrow{\boldsymbol{w_i}}$
    is the weight vector associated with class $\boldsymbol{i}$ in the Crammer-Singer formulation of
    an SDN computed as described in [48].
    Client Performs:
2       Generate Keys
3       $[\![\boldsymbol{q}]\!] \coloneqq \boldsymbol{Enc_{pk}(q)}$
4       Send $(\boldsymbol{pk}), [\![\boldsymbol{q}]\!], \boldsymbol{relinearizationKeys}$ to the server
    Server Performs:
5       $[\![\boldsymbol{C}]\!] \coloneqq (\mathbf{0}, \dots, \mathbf{0})$
6       for $\boldsymbol{w_i} \in \boldsymbol{W}$
7           $\boldsymbol{C(i)} \coloneqq \sum_j [\![\boldsymbol{w_{i,j}}]\!][\![\boldsymbol{q_j}]\!]$
8       Send $\mathbf{shuffle}(\boldsymbol{C})$ to the client
    Client Performs:
9       $\boldsymbol{C'} \coloneqq (0, \dots, 0)$
10      $\boldsymbol{C'(i)} \coloneqq \begin{cases} [\![\mathbf{1}]\!] & \boldsymbol{ArgMax_i}\left(\boldsymbol{Dec_{sk}}(\boldsymbol{C(i)})\right) = \boldsymbol{i} \\ [\![\mathbf{0}]\!] & \boldsymbol{otherwise} \end{cases}$
11      send $\boldsymbol{C'}$ to Server
    Server Performs
12      send $\mathbf{unshuffle}(\boldsymbol{C'} \cdot [\![\mathbf{1}]\!])$ to client
    Client Performs:
13      $\boldsymbol{class_q} = \boldsymbol{ArgMax_i}\left(\boldsymbol{Dec_{sk}}(\boldsymbol{C'(i)})\right)$
```

4.2 H.E. SVM

As the number of features per class grows, an algorithm that scales with the number of features will become computationally intractable. This section describes a homomorphically encrypted SVM that uses its precomputation stage to build a set of vectors W that describe the classes in the feature space.

Keerthi et. al. [49] describe a method for computing the dual of a matrix of one-to-many support vectors, which is implemented in LibLINEAR [50]. This resulting weight matrix W has the convenient property that the classification computed by a linear SVM can be computed by

$$Class_q = ArgMax_i(\overrightarrow{w_i} \cdot q^T)$$

the computation of which requires only $O(|C|[g]^d)$ multiplications. A fully trained system also does not depend on the size of the feature space, which is convenient when the number of features per class is high.

This reduction of an SVM to a small number of dot products and comparisons makes it feasible to perform the actual classification using homomorphic encryption by computing and scaling W in a plaintext pre-computation. However, W necessarily contains negative numbers, and translations into positive space change the ordering of the class weightings. This precludes the use of a BGV encryption scheme, so this algorithm is implemented using CKKS, which supports large and negative numbers but loses precision.

Without BGV, the polynomial comparison $isSmaller$ is more difficult to implement. We experimented with the polynomial expansion of a step function [51], however these expansions require that the numbers being compared are relatively close. Instead, we send the class weights to the client to perform $ArgMax$ in plaintext. In order to partially mitigate server data leakage to semi-honest clients, we perform an additional roundtrip that partially obscures the secondary class weightings.

There are several projects contemporaneous with this thesis that have shown promising preliminary results in homomorphically encrypted vector comparison. Sun et. al. have demonstrated binary classification using a hyperplane [37] by extending BGV to support efficient comparisons. Cheon et. al. have shown promising results in numerical methods for approximating comparisons in CKKS schemes. Either of these lines of research would enable an implementor of Protocol 3 to bypass the relinearization and shuffling methods with a homomorphic computation of $ArgMax$, either by supporting the BGV implementation of $isSmaller$ in the former case, or by providing an alternative to that function in the latter.

4.2.1 Efficiency

The precomputation is a function of the implementation of the training of the linear SVM, which is at least quadratic in the library we use [52].

The dot product can be run in parallel and therefore can have a depth of $O\left(\frac{n}{\text{parallelization factor}}\right)$ and size O(n). The server-side reshuffle also does an O(1) parallel operation to add noise to client data.

4.2.2 Security

The analysis used above applies here as well, a semi-honest server would be unable to meaningfully distinguish between $[\![q]\!]$ and a random $[\![q']\!]$. The final step, shuffling and unshuffling the class weights, prevents a semi-honest client from reverse engineering the vectors in two queries (which it could do otherwise), but does not provide strong security guarantees if the algorithm is run multiple times.

Note [53] describes efficient comparison in CKKS and [37] demonstrates hyperplane decision boundaries in a modified implementation of BGV that would still accommodate the *isSmaller* implementation in [1]. It is likely that either CKKS comparisons will enable the implementation in this thesis, or modified BGV will support the dual of the dot product operation in lines 6 and 7 of Protocol 3, which would similarly maintain efficiency while mitigating the server information leak.

Chapter 5: Results

This section describes the implementation and testing of the algorithms described in Chapter 4: and the analysis of those results. We show that the extension to k-ishNN exhibits improved efficiency and accuracy in a wide range of conditions, and demonstrate that a homomorphically encrypted SVM can operate efficiently on real-world datasets.

5.1 Implementation Details

Here we discuss some high-level implementation details. See Appendix A – Source Code for the specific implementations of these algorithms. In order to facilitate more rigorous testing of the precision of algorithms, we built each protocol twice: once with encryption, and once with emulated encryption (which runs orders of magnitude faster). Before running the unencrypted simulator, we ran both protocols several dozen times on several hundred candidate values of S, k, and q. Because the algorithms have a degree of randomness, we decrypted and compared the distribution of intermediate variables to ensure that the simulated and actual algorithms are functionally equivalent.

Our encrypted implementations were built by modifying existing libraries. For each algorithm, we relied on several existing projects:

For the k-ishNN implementation, we used the "privacy preserving kNearestNeightbor" (*ppKNN)* open source repository [5] as a starting point. This library uses the Library of

Practical Homomorphic Encryption algorithms (liphe) [11] to access the HElib implementation of the BGV protocol [12]. We also used liphe helper functions to generate polynomial expansions of functions when necessary. We implemented an emulator that performs HElib and liphe operations in python using NumPy [13].

For the ppSVM implementation, we used PyFhel [14] as a starting point. PyFhel is a python interface to HElib and the Microsoft Simple Encrypted Arithmetic Library [15], [16], which both implementations of BGV and CKKS homomorphic encryption protocols. PyFhel has been out of maintenance for several years and required significant modification. For training, we used the LibLINEAR [8] implementation of Crammer and Singer's SDM training method. We performed data cleaning and preparation using the scikit-learn [18] data cleaning and vectorization libraries.

5.2 Tests

We tested the classifiers against three datasets: a collection of measurements and diagnoses of tumor samples, a database of car data, and a collection of facial images. The former two datasets were selected because they were used to established baselines in the foundational work this thesis extends. The latter was selected as a motivating dataset that, while sufficiently limited in scale to allow the testing of a wide range of potential approaches, still presents many of the challenges inherent in biometric recognition tasks.

For each dataset, we vary grid size and (for nearest neighbor-based algorithms) the size of k to measure the effect of these values on the accuracy of the algorithms. We also prepared versions of each dataset projected into different dimensions to measure the trade-off between computational and algorithmic efficiency at low dimensions and increased availability of precision and differentiation in high dimensions.

5.2.1 Breast Cancer

The Wisconsin Breast Cancer Diagnostic Dataset is a set of vectors representing 357 benign and 212 malignant breast cancer tumors. Each vector is a 30-dimensional normalized vector representing the mean, standard error, and "worst" readings (defined as the average of the three highest readings) for ten measurements taken on each tumor.

The dataset is popular as a test dataset for initial testing of binary classification algorithms because it has two easily distinguished classes in a continuous, high dimensional space. Generally, benign tumors are clustered closely, while there is a higher degree of variability in the measurements of malignant tumors. Because the size and depth of our homomorphically encrypted KNN approximation depends heavily on the number of dimensions in the space being classified, we use 2d, 3d, and 5d LDA projections of the space in addition to the original 30-dimensional dataset. Performing an LDA projection involves multiplying by a single projection matrix, which we assume is performed by the client in these experiments. Figure 4 and Figure 5 show the 2- and 3-D projections of the breast cancer dataset respectively.

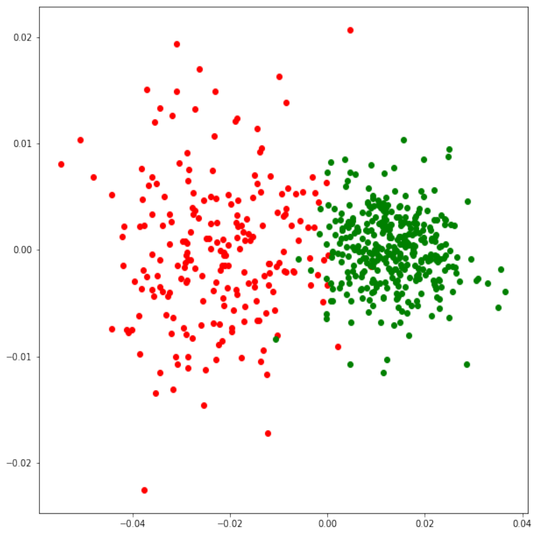


Figure 4: LDA projection of the breast cancer diagnostic dataset into two dimensions. The green cluster shows benign tumors; the red cluster is malignant.

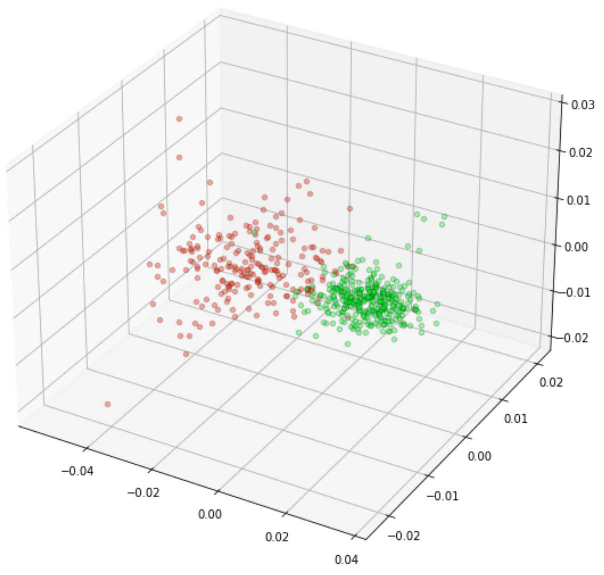


Figure 5: LDA projection of the breast cancer diagnostic dataset in three dimensions. As in Figure 4, the green cluster (right) shows benign tumors, while red shows malignant tumors (left). Axes are still meaningless. Note that again the clusters are relatively distinct from each other, although the malignant set is less tightly clustered.

As discussed in Section 3.4.3 there is an overflow risk in threshold computation. Tests involving binary classification may not capture this error if the correct classification is also the class of the element more common in the feature set; this is the case in the breast cancer set, which has over 50% more benign than malignant tumors. Because overflow sensitivity is a function of feature density, we generated a second test set with additional artificial malignant tumor vector representations (Figure 6). The ratio of malignant to benign points in this synthetic set is the inverse of the original dataset.

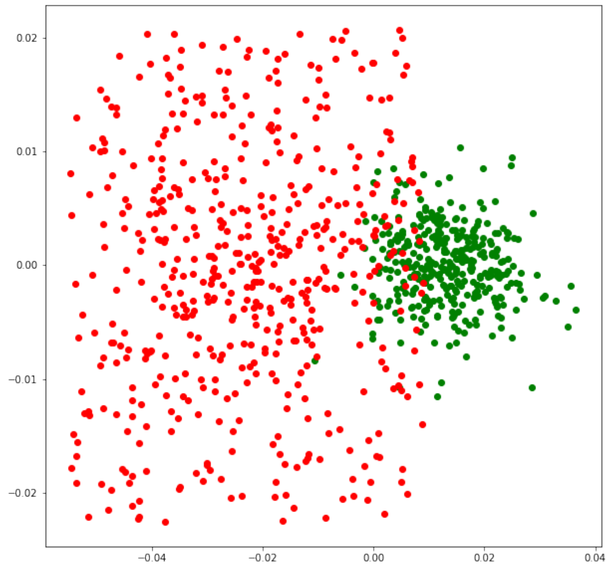


Figure 6: LDA projection in two-dimensional space of modified breast cancer dataset. Meanings of the clusters are the same as Figure 4. The benign tumor data is unchanged. Additional malignant tumor data has been randomly generated with a convex hull around the original malignant tumor feature cloud.

5.2.2 Cars

The Car Evaluation Data Set is a set of 1728 cars' profiles represented by integer vectors in a 4x4x4x3x3x3 space (totaling 1748 possible descriptions). Each vector represents the car's value, level of maintenance, number of doors, capacity, trunk size, and safety rating. Each car is then classified by desirability on a four-point scale.

The sample contains a large concave region of “undesirable” cars and smaller clusters within the “desirable” space. Similar to the breast cancer diagnostic dataset, we use 2- and 3- dimensional LDA projections of the space in addition to the original sample, which are shown in Figure 7 and Figure 8.

Because the sample almost completely describes the space, it has roughly uniform density, meaning there is a low variability in the range of thresholds necessary to accurately estimate k when building a KNN classifier. The discrete and categorical nature of the axis in this dataset also make the use of l_1 norms a natural choice of distance function.

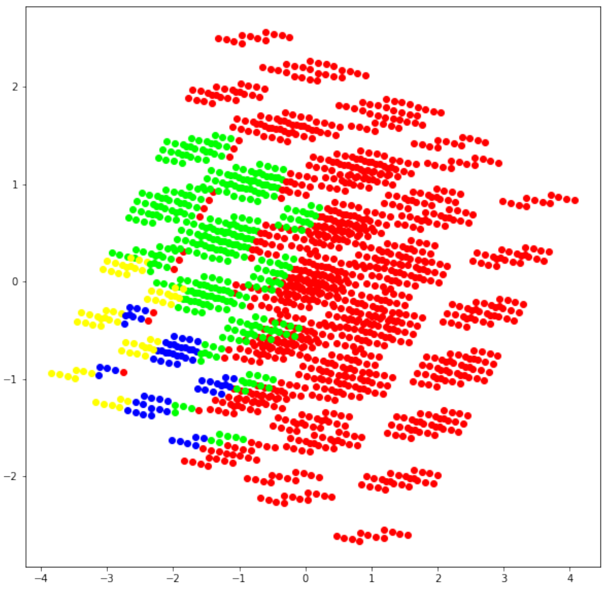


Figure 7: LDA projection of cars dataset in two dimensions. The large, red cluster (right) are undesirable cars. The green (left, top), blue (left, bottom), and yellow (far left) are the various levels of “desirability”

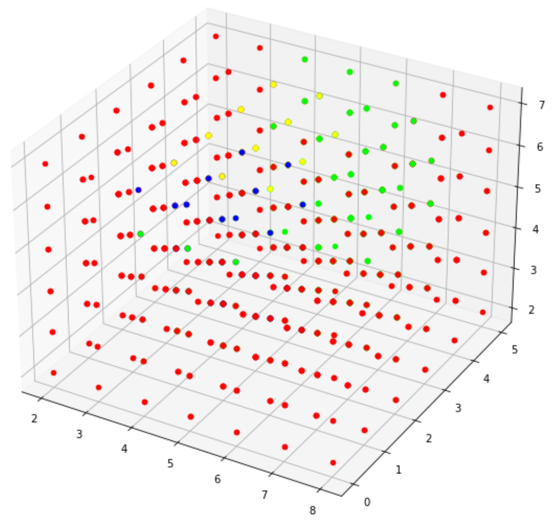


Figure 8: LDA projection of cars dataset to three dimensions. Colors are the same as Figure 7.

5.2.3 Faces

The Yale Extended Facial Dataset B [23] is a collection of photographs of the faces of 28 human subjects in a variety of lighting conditions and angles. We trained an embedding of the dataset using the OpenFace [41] PyTorch [54] implementation of FaceNet [25] to create a non-linear mapping from facial images to points in a 128-dimensional embedding space. For these experiments we consider the ability of our algorithms to correctly classify a face given the vector representation of the face in that embedding.

We selected the dataset as a difficult motivating problem. It has a comparatively high number of classes and heterogeneity as compared to other test datasets used.

As with the other two datasets, we use dimensionality reduction techniques to reduce the number of dimensions and improve the efficiency of our algorithms. We attempted several non-linear dimensionality reduction algorithms. First, we tested t-SNE (Figure 9 and Figure), which seeks to preserve "clusters" of points in the projection and was designed and is largely used as a visualization and debugging tool. Projecting queries into a reduced space generated by t-SNE [55] requires comparisons to points in the original dataset, defeating the original purpose for performing dimensionality reduction [56]. We then used MDS, which seeks to preserve distances between points during dimensionality reduction, but it fails to preserve class boundaries without significant turning, and has a similar "out of sample" mapping complexity issue to t-SNE.

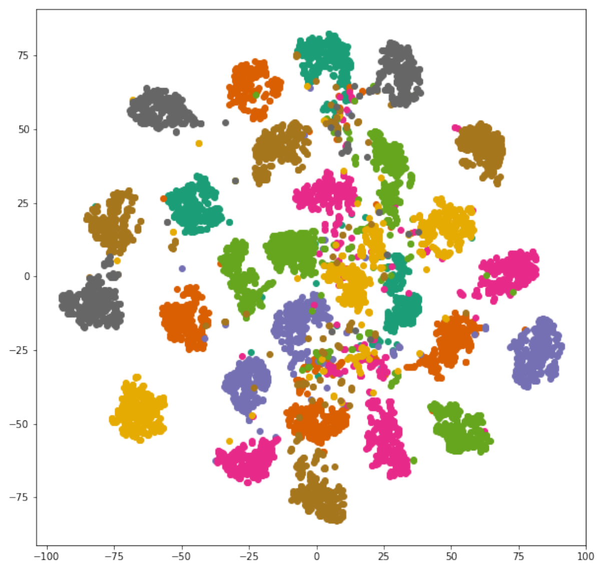


Figure 9 - Yale Extended B dataset projected to two dimensions using t-SNE. The t-SNE project seeks to preserve "clusters" of points. High degrees of visual separation on this graph tends to correlate with an original embedding space that has distinct regions for different classes of features. Each color corresponds to a single subject. Some of the 28 colors are very similar; each class is tightly clusters in this dataset.

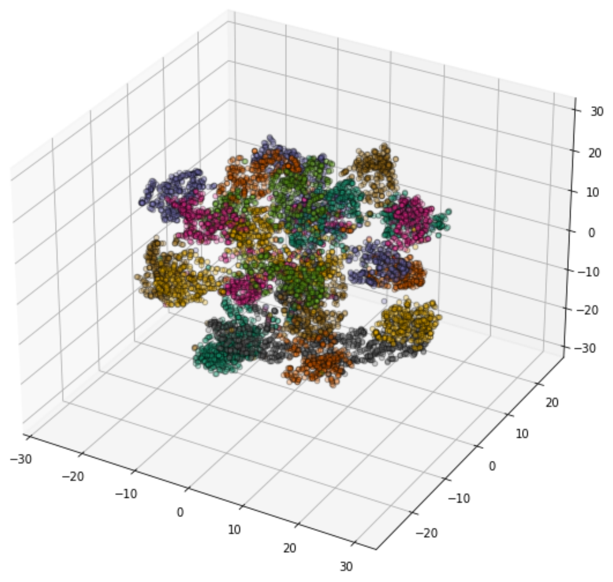


Figure 10: Yale Extended B dataset projected to three dimensions using t-SNE. The interpretation of this graph follows the same logic as Figure 9. When inspected interactively, the clusters here have a visual margin between them.

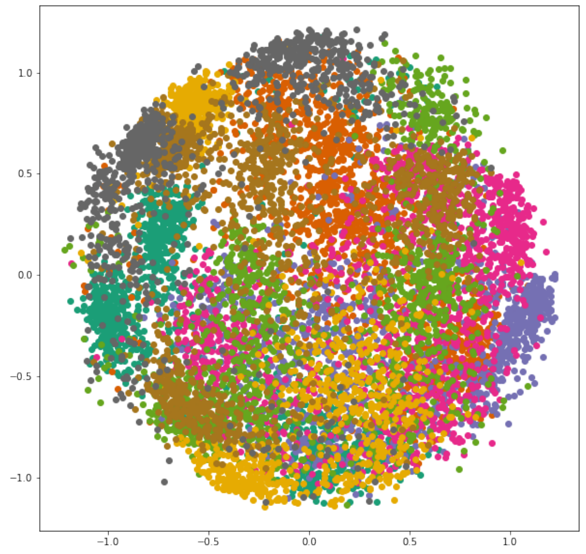


Figure 11 Yale Extended B dataset projected into two dimensions using non-metric MDS. MDS is a non-linear mapping from high dimensional space that seeks to maintain distance between points. Unlike t-SNE in figures 9 and 10, it is possible to map newly introduced points in the original feature space into an MDS projection. As is apparent from the high levels of overlap between classes in this image, MDS does not preserve class separation on our dataset; so MDS isn't particularly useful in this application.

An effective alternative to dimensionality reduction techniques is to retrain the classifier that maps faces into an embedding space by altering the size of the embedding space to the desired dimensionality. This causes some loss of expressiveness of the original mapping, but allows for efficient mapping of queries into a low dimensional feature space. The retraining can be performed efficiently on an already-trained classifier by preserving the weights in the rest of the system.

The experiments in this thesis use the LDA projection of a high-dimensional embedding space as the technique is more generally applicable than modifying and retraining an embedding.

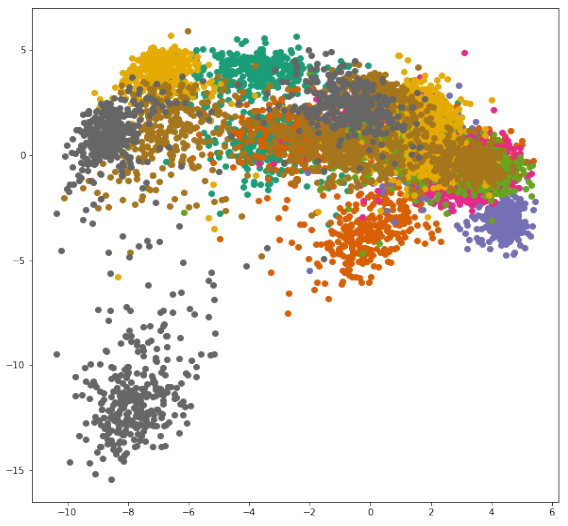


Figure 12: An LDA projection into two dimensions of the Yale Face Database B. New points in the embedding space can be efficiently mapped into the low dimensional space using only additions and multiplications (the projection is a dot product with the basis vectors of the new space). This makes an LDA projection a natural choice for HE classifiers if enough separation between classes is preserved.

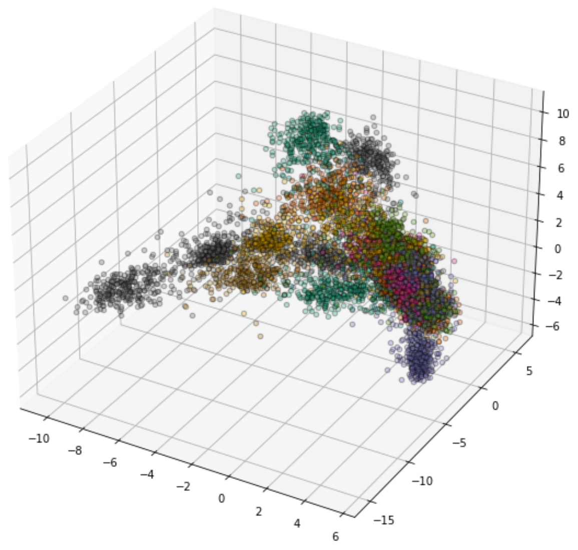


Figure 13 An LDA projection into three dimensions of the Yale Face Database B. There is visual separation between many of the classes in the dataset in this projection, and LDA supports efficient projection from the original embedding space using only HE-supported operations. This makes the LDA projection an attractive choice for a dimensionality reduction technique in this trial.

5.3 Metrics

5.3.1 Accuracy

In order to measure of the accuracy of each algorithm we perform leave-one-out cross validation to provide consistent results with [1]. We compute the accuracy of each algorithm, defined as the ratio of correct classifications to total classification attempts. We also measure the error rate. The error rate and accuracy do not always sum to one because our modification of k-ishNN can refuse to classify an ambiguous query.

5.4 The Experiments

5.4.1 Accuracy

KNN accuracy with different threshold families

We ran KNN, k-ishNN (Protocol 1), and staticKishNN (Protocol 2 with Algorithm 1a) on all three datasets for a range of grid sizes from 10 to 250 and a range of values of k from 2 to 256. For each dataset, I've performed cross-validation against reserved sets of points to measure the accuracy and precision of each algorithm.

In most experiments, both values are identical, however because the static threshold approach can choose a null response if it does not have agreement between at least two thresholds, it achieves a higher accuracy than even the reference KNN algorithm, though it continues to have lower precision.

Figures 15-21 depict the results of these experiments. The reference implementation of KNN is depicted in red, the thesis' contribution of k-ishNN with precomputation is blue, and the original k-ishNN algorithm is green. Across all three datasets, the static precomputation approach outperforms the k-ishNN algorithm for all values of k and grid resolutions in both accuracy and runtime. The difference is particularly pronounced when performing facial recognition, our motivating biometric recognition task. On the biometric recognition task, k-ishNN barely outperforms random chance, with between 0 and 30% correct classifications, while KNN and k-ishNN achieve accuracies as high as 55%.

Also noteworthy is the comparison between figures 17 and 19. Figure 17 depicts the breast tumor dataset binary classification task (Figure 4), and both HE systems achieve the reasonable accuracy originally observed in [1]. In Figure 19, the modified version of the same dataset (figure 6) exhibits significant decay in accuracy for the k-ishNN algorithm. The former dataset has more training features for the class with higher spatial feature density, the latter has more features for the more widely distributed class. The latter dataset results in high misclassification rates in the case of an overflow. Also notable between these two datasets: the decreased variation in point density across the feature space results in a significant improvement in classification using static thresholds, as the variance in the value of T that would result in $\kappa = k$ is a function of density variance.

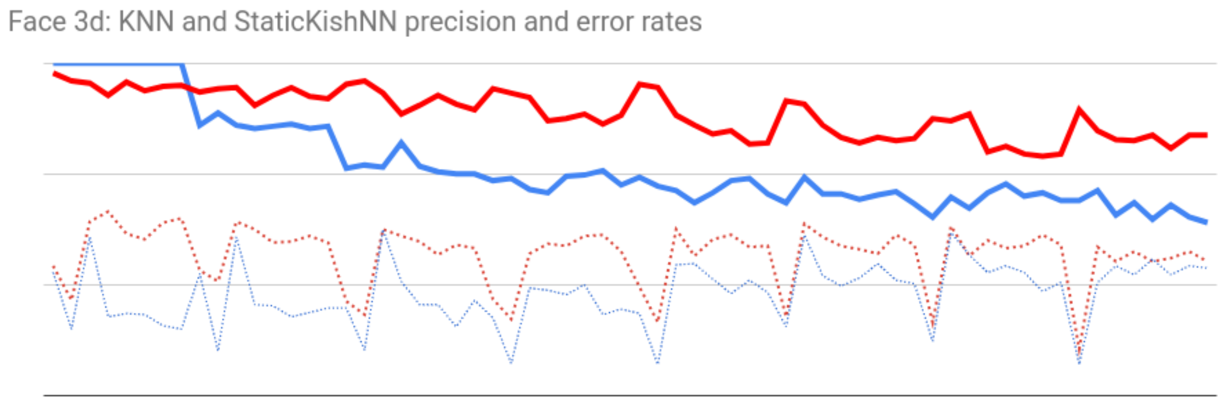
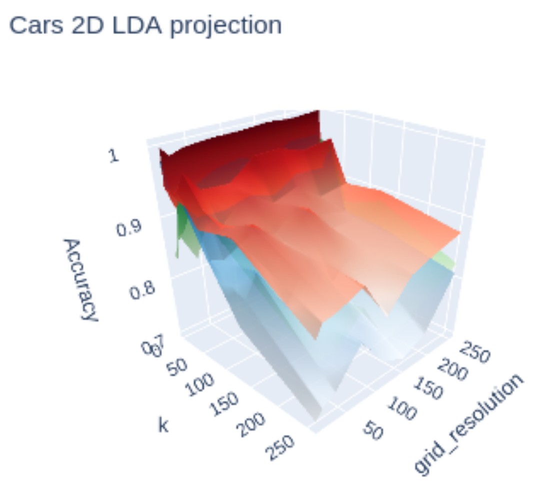


Figure 14 – the precision (solid) and error rates (dotted) of plaintext KNN (red) and StaticKishNN blue) on the 3d LDA projection of the Yale Face Database. The X axis is an unordered permutation of values of $k \in \{2,4,8,16,32,64,128,256\}$ and $grid\_size \in \{20, 40, 80, 120, 160, 200, 300\}$. The design choice of building the static k-ishNN classifier to dismiss ambiguous results means that, though it has a lower recall rate than classic KNN, it also has a consistently lower false classification rate.



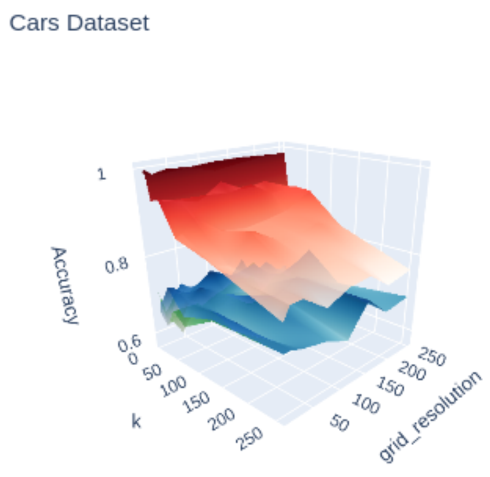


Figure 15: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the cars dataset using a 2D LDA projection.

Figure 16: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the cars dataset. Because the cars dataset is already on a grid, the grid size has no effect on output.

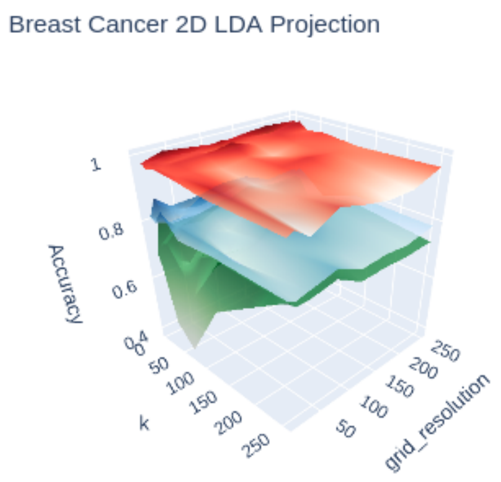


Figure 17 Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 2d LDA projection of the Wisconsin Breast Cancer dataset

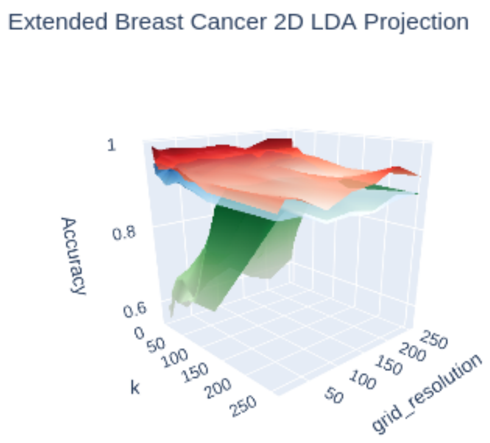


Figure 19 Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 2d LDA projection of the extended breast cancer dataset In contrast to Figure 17, here k-ishNN's performance decays rapidly as k and grid size decrease.

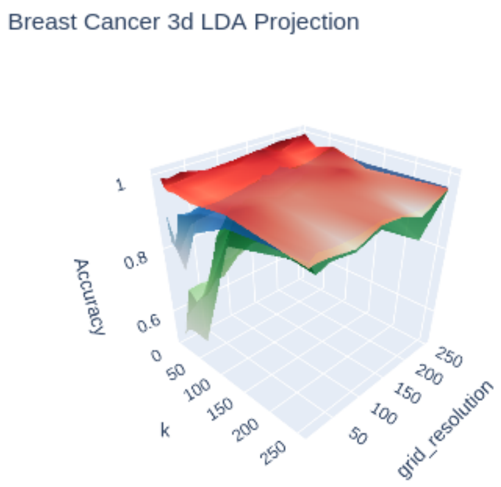


Figure 18: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on a 3d LDA projection of the Wisconsin Breast Cancer dataset.

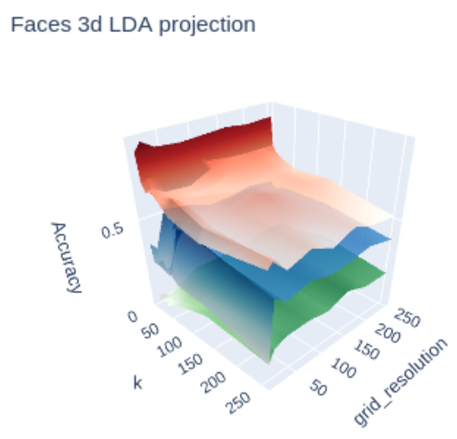


Figure 20: Performance of KNN (red), static$k$-ishNN (blue), and k-ishNN (green) on the faces dataset using a 3D LDA projection.

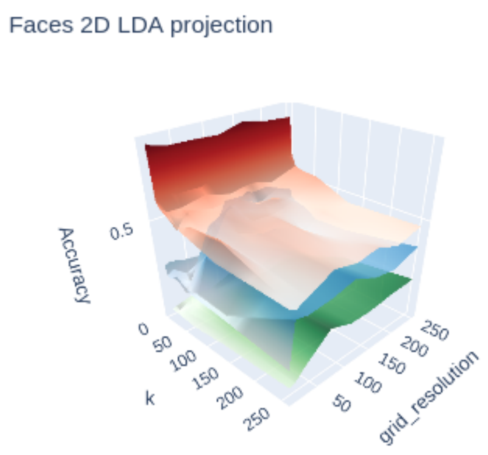


Figure 21: Performance of KNN (red), Static$k$-ishNN (blue), and k-ishNN(green) on the faces dataset using a 2D LDA projection.

Privacy Preserving SVM

We repeated the experiments to measure the performance of the privacy preserving SVM (protocol 3 – labeled "ppSVM"). Using the OpenFace algorithm as a benchmark, ppSVM has lower precision that improves as the number of dimensions increases. We also measured the effect of differing grid sizes, and though the grid impacted runtime, it did not impact accuracy for even relatively coarse grids.

In general, the SVM and KNN algorithms performed similarly in low dimensions. Because SVM scales with $\Omega(classes \cdot dimensions)$ and KNN scales with $\Omega(classes \cdot features)$, we were able to run experiments on SVMs on a higher number of dimensions without either exceeding a noise threshold on the computation or needing to execute multi-day runtimes.

The ability to scale to higher dimensions allows SVM based algorithms to take advantage of more information, and this is reflected in the results, which show significant improvements to precision as the number of dimensions increases.

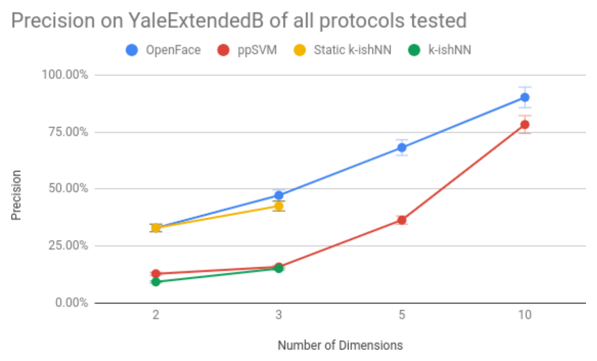


Figure 22: Average precision of selected algorithms on face dataset in multiple dimensions. The OpenFace implementation of FaceNet (blue) serves as a benchmark. The KNN approximation algorithms were only run on 2 and 3 dimensional datasets because the computation demands of those approaches rapidly become impractical for datasets with large numbers of training features, such as the facial recognition database used here.

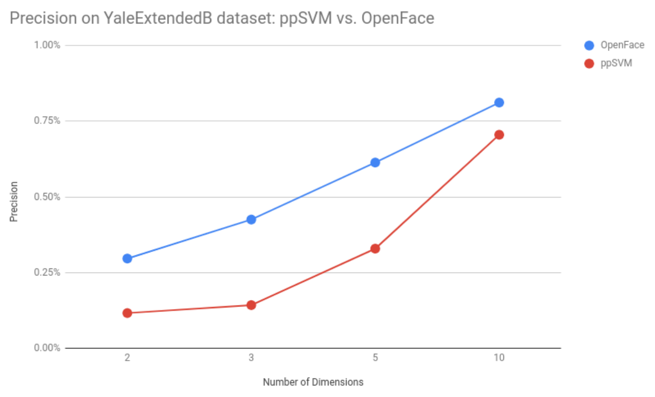


Figure 23 A one-to-one comparison of the OpenFace and ppSVM classifiers on vectors projected into an identical embedding space. OpenFace uses the scikit-learn [57] SVM classifier. The accuracy of the two algorithms begin to converge as the number of dimensions increases.

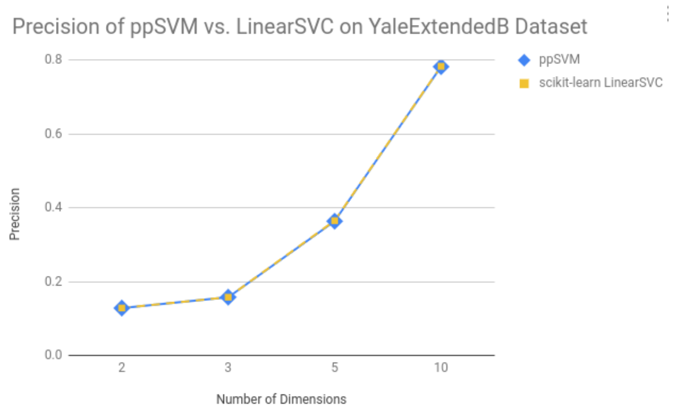


Figure 24 Precision of ppSVM and scikit-learn's wrapper of LibLINEAR [50], [57]. Though the two lines appear to overlap, the computation noise in ppSVM has caused a very slight decrease in precision on the test dataset. Both achieve nearly 80% accuracy in single element cross validation on a ten-dimensional projection of our facial recognition embedding space.

5.4.2 Timing

We also measure the runtime of these algorithms. Figures 25 and 26 show timing data collected during the experimental testing of the k-ishNN system described in Section 5.4.1. In addition to being more accurate, the pre-computation approach has significantly improved runtimes, especially for large grid sizes. This is likely because the approximation used in Algorithm 1b requires the computation of several additional polynomials. We did not measure the degree of parallelism used in these experiments, it may be possible to improve performance of one or both of these algorithms through improved optimization and parallelization.

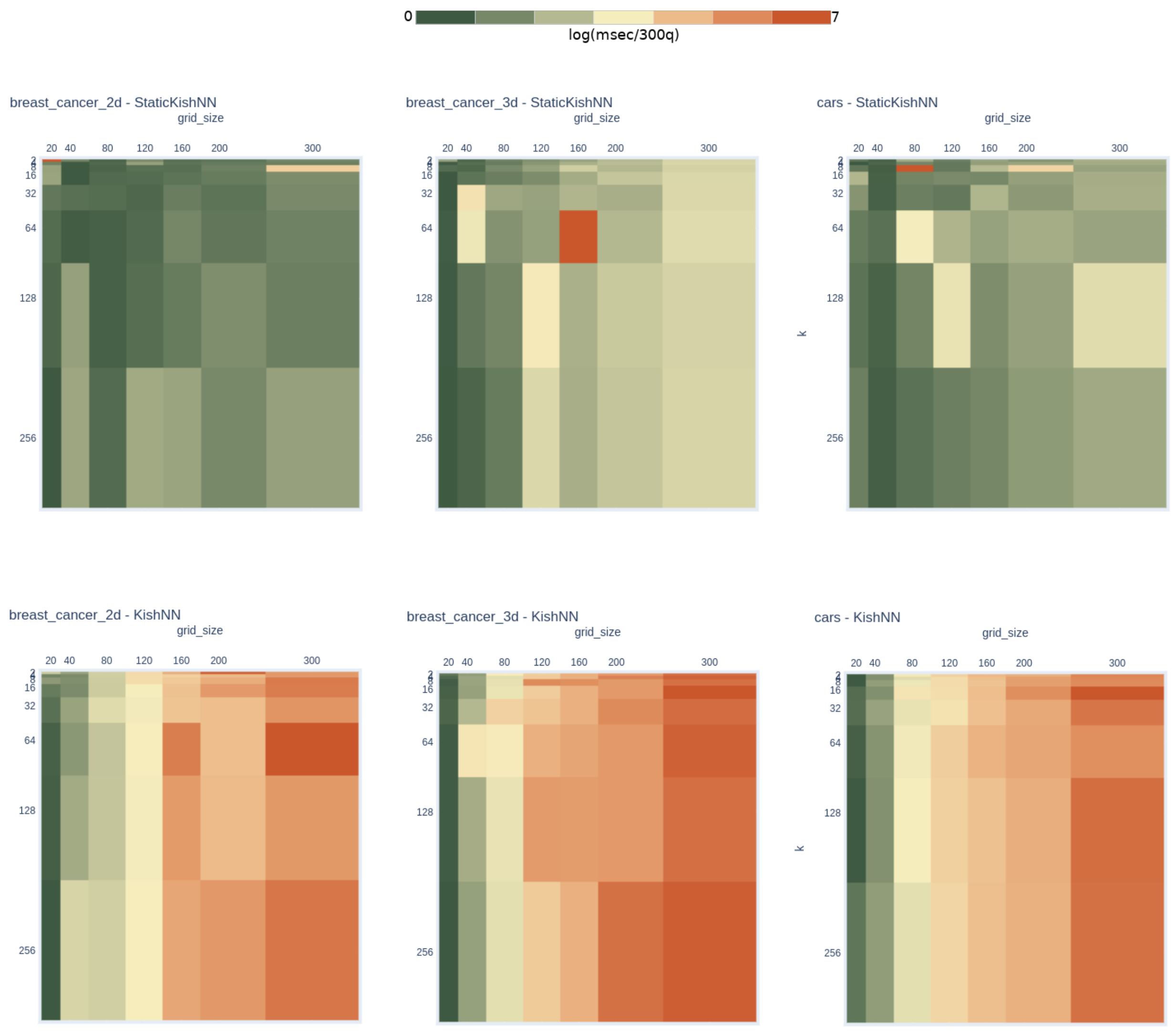


Figure 25 – The runtime of classification tasks using the k-ishNN algorithm both with and without precomputation. Colors are displayed based on a normalized log scale (dark green is a fraction of a second per query, dark red is over a minute per query). Timing was computed by taking the average time per query of 300 queries during cross-validation. Measurements were taken on a docker container running on a 2016 MacBook Pro with a 2.9 GHz Intel Core i5 processor and 8GB of LPDDR3 RAM. The data contains some clusters of outliers caused by the thermal throttling by the host OS. The algorithms that use precomputation can run in seconds instead of minutes on large grid sizes. As expected, runtime does not vary as a function of k.

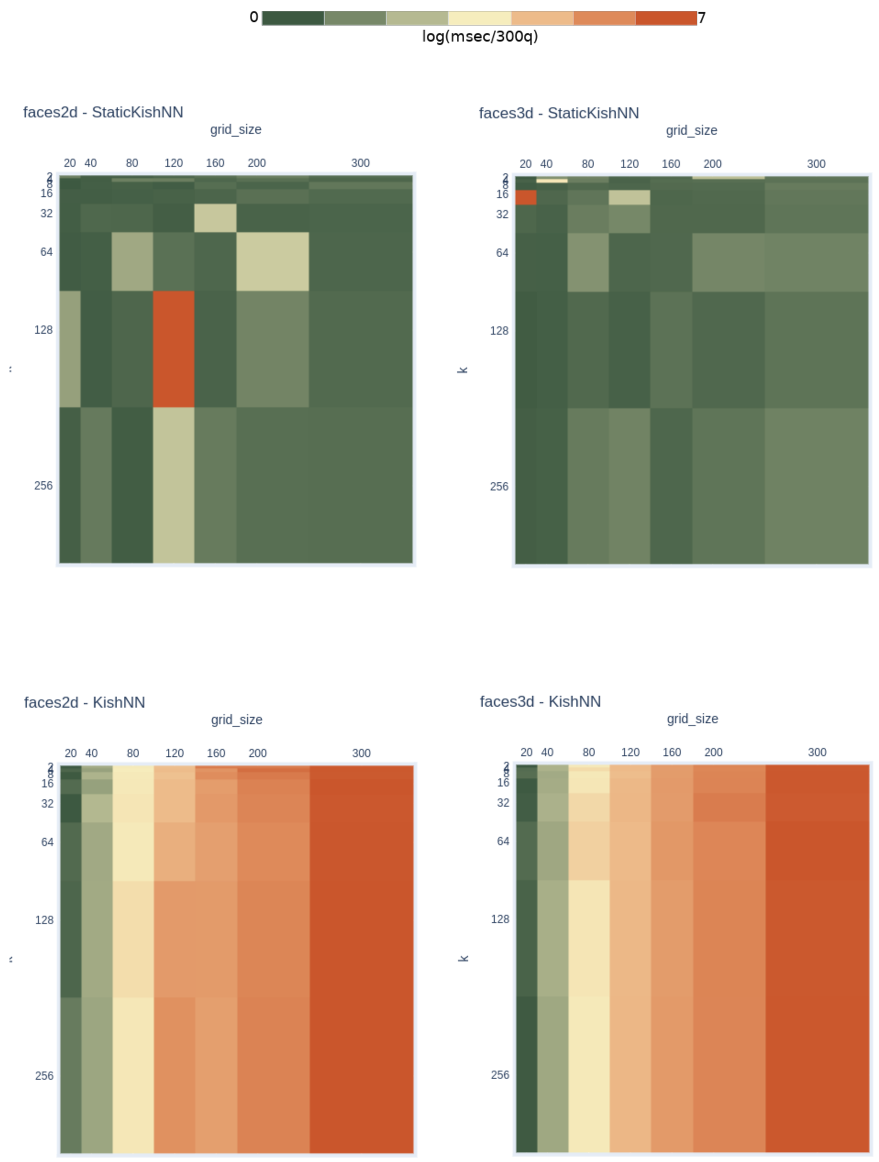


Figure 26 – The runtime of biometric classification tasks using the k-ishNN algorithm both with and without precomputation. Colors are plotted on the displayed based on a normalized log scale. The runtimes were collected using the same system as the one described in Figure 25. On this dataset the KishNN algorithm exceeded the available parallelism, causing significant slowdowns.

The implementation of the SVM algorithms ran on a different encryption library (SEAL [34] instead of HElib [33]) and no attempt was made to optimize any of the parameters in this system, resulting in a time per query of several minutes and making comparisons between that algorithm and the modified k-ishNN algorithm not particularly meaningful.

5.5 Discussion

We have shown that a naïve precomputation step can result in across-the-board performance and accuracy improvements to a homomorphically encrypted classification algorithm. This result suggests that exploring other pre-trained classification parameters or selecting from a family of functions based on the training database might yield even more improvements.

These experiments also demonstrated that classification tasks that scale with the number of classes rather than the number of feature sets are feasible given the current state of the art, though our implementation has high fixed overheads. Incorporating improved parallelism and more efficient comparison operators could increase the speed and utility of these algorithms in the future.

Chapter 6: Conclusion

This thesis presents two methods for moving most of the computation necessary to perform classification into pre-computation on the training set, allowing for efficient, homomorphically encrypted classification at sufficient resolution to enable biometric recognition. We demonstrate the efficacy of both of these methods on a binary classification task involving actual breast tumor data, on car desirability classification, and on a general facial recognition task.

These contributions demonstrate that it is both possible to perform practical server-side biometric recognition without exposing data to the classifier.

6.1 Next Steps / Future Work

Both algorithms rely on relatively naïve implementations and have several areas for improvement in future work.

ppSVM does not take advantage of recent innovations in the computation of hyperplane decision boundaries in BGV, and makes no effort to optimize the CKKS implementation it uses, or to implement comparisons in CKKS. Future work should apply these optimizations to sparse SVMs, which could potentially significantly improve the efficiency of the algorithm.

The k-ishNN extensions establish that even a constant function can outperform the originally implemented Gaussian approximation. Exploring alternative families of threshold generation functions might improve accuracy. For example, while preparing this thesis we used brute force to generate a polynomial that takes the average value of X as an input and returns the threshold as output, but ran into overfitting and approximation error issues; a more sophisticated approach to precomputing a threshold generation function or selecting the most appropriate generator from a family of candidate functions might improve the adaptability of the algorithm.

More generally, finding ways to adjust established algorithms to cache or precompute the portions of the algorithms that can be performed in plaintext is underexplored in the classification context. As more functions are well approximated in homomorphically encrypted contexts, this may be a fruitful area of inquiry.

6.2 Practical applications unlocked

The contributions of this thesis are sufficient to build a facial classifier that can identify people from a small community while giving those people strong privacy guarantees. Privacy preserving identification systems have clear utility in situations where identification is desirable but at odds with an expectation of privacy. For example, the algorithms in this thesis could be to build security systems that alert when an unfamiliar person enters an area (an important application for automating "bubbles" during the current COVID pandemic) while not building detailed logs that violate expectations of privacy.

This might also provide a practical “shield” from legal liability. Service providers could provide classification-based services without collecting sensitive or heavily regulated data. For example, these algorithms could be used by medical diagnostic classifiers to provide medical advice and insight without requiring access to the patient’s identity or diagnosis, which might be important for encouraging patient participation in the face of concerns about increased insurance premiums (an example originally considered in [1]).

6.3 Potential Interactions with Technology Policy

There are two categories of technology policy considerations where this might have immediate practical impact. The first is informing how regulators design policies relating to data privacy, use, storage and sharing. The second is how service providers design systems to minimize liability risk and exposure.

Some aspects of data use, such the quantity data preserved or mechanisms used in data storage, is difficult for regulators to observe or enforce. Because of this, data regulation tends to focus on the collection and use of data, or on the behavior of a firm after a public breach. These policies, because they focus on the nature of the data and its use, have recently been formulated around categories of data, of which personally identifiable and biometric data is particularly sensitive. This work demonstrates that application-based restrictions might be overly aggressive, and exceptions for encrypted data could allow regulators to protect user data without restricting useful services by creating exceptions for encrypted computation.

Service providers can approach this problem from the other side. Breach notification and security and compliance rules are tuned explicitly for the nature of the data being stored [58]–[60]. Data that is irrecoverable or uninterpretable (such as encrypted data with no key) is in many jurisdictions subject to much lower standards of care. Using homomorphic encryption for simple classification tasks using a method like the one outlined in Section 4.2 (which does not require the storage of reference features), might avoid many of the more stringent regulations by never collecting the most sensitive classes of information.

6.4 Lessons Learned

Cryptographic algorithms should be run in simulation first. Similar to machine learning algorithms, cryptographic computations effectively run in a black box. Building monitoring systems to keep track of noise levels, decrypt and log intermediate values, and generally provide a higher level of visibility was critical to exploring and debugging these algorithms. Discovering the overflow problem with the k-isnNN algorithm took days, both because intermediate values in an encrypted system are obfuscated, and because running a test of an HE algorithm can take hours to days to run across a permutation of possible inputs. Building an “unencrypted mode” that tracked noise levels but did not actually perform the expensive encrypted computations was a massive productivity boost.

Heterogeneous and hetero-temporal execution environments require heterogeneous system design. Portions of our algorithm necessarily ran in a computing environment with vastly different properties and constraints than a typical computing model. Some operations (those responding to a query) needed to happen in real time, while others could be precomputed. Good system design requires careful consideration of where and when computation can be performed most effectively. In a typical computing environment, caching the results of an asymptotically fast operation is usually unnecessary; in a partially homomorphically encrypted environment, it is worthwhile to create somewhat awkward divisions between different parts of algorithms in order to move as much computation into a cache or at least into plaintext. The constant factors are high enough that this often remains true even if the move to plaintext or a cache results in worse asymptotic complexity.

Cryptography needs better interdisciplinary communication. The author worked on this project while also involved in regulatory policy research and had a chance to discuss the work with both his technical and non-technical colleagues. As a rule, technical experts are less likely to trust (unprovable) policy-based approaches to privacy and fairness and legal experts are less likely to trust (difficult to understand or audit) technical measures to protecting privacy and fairness. Especially under-communicated are the various security threat models and how those threat models are designed and used.

6.5 Final Thoughts

Biometric recognition is a sensitive topic. It's become a hot-button issue as surveillance concerns play a growing role in civil and political discourse. This thesis describes and demonstrates the feasibility of biometric recognition systems that operate entirely on encrypted data, establishing an importing first step in providing technical tools that mitigate many of the social concerns raised by modern recognition systems.

Bibliography

[1] H. Shaul, D. Feldman, and D. Rus, "Secure \$k\$-ish Nearest Neighbors Classifier," in *PoPETS*, 2020, pp. 1–20, Accessed: 07-Aug-2020. [Online]. Available: http://arxiv.org/abs/1801.07301.

[2] P. Jayapal, "H.R.7356 - 116th Congress (2019-2020): Facial Recognition and Biometric Technology Moratorium Act of 2020," 2020, Accessed: 17-Aug-2020. [Online]. Available: https://www.congress.gov/bill/116th-congress/house-bill/7356.

[3] E. J. Markey, "S.4084 - 116th Congress (2019-2020): Facial Recognition and Biometric Technology Moratorium Act of 2020," 2020, Accessed: 17-Aug-2020. [Online]. Available: https://www.congress.gov/bill/116th-congress/senate-bill/4084/text.

[4] J. Zou and L. Schiebinger, "Design AI so that its fair," *Nature*, vol. 559, no. 7714. Nature Publishing Group, pp. 324–326, 19-Jul-2018, doi: 10.1038/d41586-018-05707-8.

[5] A. Amini, A. P. Soleimany, W. Schwarting, D. Rus, and S. N. Bhatia, "Uncovering and Mitigating Algorithmic Bias through Learned Latent Structure," vol. 19, 2019, doi: 10.1145/3306618.3314243.

[6] *Privacy and Technologies of Identity.* Springer-Verlag, 2006.

[7] "About Face ID advanced technology - Apple Support." https://support.apple.com/en-us/HT208108 (accessed Aug. 18, 2020).

[8] "Sidewalk Labs | The next-generation intersection helps all modes share the street." https://www.sidewalklabs.com/blog/the-next-generation-intersection-helps-all-modes-share-the-street/ (accessed Aug. 18, 2020).

[9] "The Future of AI in Law Enforcement - Intel." https://www.intel.com/content/www/us/en/analytics/artificial-intelligence/article/ai-helps-find-kids.html (accessed Aug. 18, 2020).

[10] H. A. Abbass, "An evolutionary artificial neural networks approach for breast cancer diagnosis," *Artif. Intell. Med.*, vol. 25, no. 3, pp. 265–281, Jul. 2002, doi: 10.1016/S0933-3657(02)00028-3.

[11] ISACA, "State of Cybersecurity 2020 (part 1)," *Isaca*, no. November 2018, p. 22, 2020.

[12] R. Rivest, A. Shamir, and L. Adleman, "On Data Systems and Privacy Homomorphisms," *Found. Secur. Comput.*, vol. 4(11), pp. 169–180, 1978, Accessed: 15-Aug-2020. [Online]. Available: https://pdfs.semanticscholar.org/3c87/22737ef9f37b7a1da6ab81b54224a3c64f72.pdf%0Ahttp://files/834/22737ef9f37b7a1da6ab81b54224a3c64f72.pdf.

[13] C. Gentry, "A Fully Homomorphic Encryption Sch," 2009.

[14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ACM Transactions on Computation Theory*, 2014, vol. 6, no. 3, doi: 10.1145/2633600.

[15] S. Halevi and V. Shoup, "Algorithms in HElib," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8616 LNCS, no. PART 1, pp. 554–571, doi: 10.1007/978-3-662-44371-2_31.

[16] "{M}icrosoft {SEAL} (release 3.5)." Apr-2020.

[17] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin, "LIBLINEAR: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, 2008, doi: 10.1145/1390681.1442794.

[18] J. K. Winn, "Are 'Better' Security Breach Notifications Possible?," *Berkeley Technol. Law J.*, vol. 24, pp. 1009–1018, 2009.

[19] M. Burdon, "Contextualizing the tensions and weaknesses of information privacy and data breach notification laws," *St. Cl. Comput. High Technol. Law J.*, vol. 27, no. 1, pp. 63–129, 2010, [Online]. Available: http://digitalcommons.law.scu.edu/chtlj/vol27/iss1/3/%5Cnhttp://heinonline.org/HOL/Page?handle=hein.journals/sccj27&id=65&div=&collection=.

[20] "NSA collecting phone records of millions of Verizon customers daily | US news |

The Guardian.” https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order (accessed Aug. 18, 2020).

[21] “U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program - The Washington Post.” https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html (accessed Aug. 18, 2020).

[22] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, 1997, doi: 10.1109/34.598228.

[23] “Yale Face Database.” http://vision.ucsd.edu/~leekc/ExtYaleDatabase/Yale Face Database.htm (accessed Aug. 17, 2020).

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks.” Accessed: 18-Aug-2020. [Online]. Available: http://code.google.com/p/cuda-convnet/.

[25] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June, pp. 815–823, doi: 10.1109/CVPR.2015.7298682.

[26] "PRIVACY AND TECHNOLOGIES OF IDENTITY."

[27] J. Heurix, P. Zimmermann, T. Neubauer, and S. Fenz, "A taxonomy for privacy enhancing technologies," *Comput. Secur.*, vol. 53, pp. 1–17, Jun. 2015, doi: 10.1016/j.cose.2015.05.002.

[28] "Google AI Blog: Federated Learning: Collaborative Machine Learning without Centralized Training Data." https://ai.googleblog.com/2017/04/federated-learning-collaborative.html (accessed Aug. 18, 2020).

[29] K. Bonawitz *et al.*, "Practical Secure Aggregation for Privacy-Preserving Machine Learning."

[30] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "ON DATA BANKS AND PRIVACY HOMOMORPHISMS," 1978.

[31] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2009, pp. 169–178, doi: 10.1145/1536414.1536440.

[32] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10624 LNCS, pp. 409–437, doi: 10.1007/978-3-319-70694-8_15.

[33] S. Halevi, “An Implementation of homomorphic encryption,” p. 2015, 2015, Accessed: 15-Aug-2020. [Online]. Available: https://github.com/shaih/HElib.

[34] “microsoft/SEAL: Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library.” https://github.com/Microsoft/SEAL#citing-microsoft-seal (accessed Aug. 07, 2020).

[35] “HayimShaul/liphe: Library for Practical Homomorphic Encryption.” https://github.com/HayimShaul/liphe (accessed Aug. 16, 2020).

[36] “ibarrond/Pyfhel: PYthon For Homomorphic Encryption Libraries, perform encrypted computations such as sum, mult, scalar product or matrix multiplication in Python, with NumPy compatibility. Uses SEAL/HElib/PALISADE as backends, implemented using Cython.” https://github.com/ibarrond/Pyfhel (accessed Aug. 07, 2020).

[37] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, “Private Machine Learning Classification Based on Fully Homomorphic Encryption,” *IEEE Trans. Emerg. Top. Comput.*, vol. 8, no. 2, pp. 352–364, Apr. 2020, doi: 10.1109/TETC.2018.2794611.

[38] V. Naresh Boddeti, “Secure face matching using fully homomorphic encryption,” in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems, BTAS 2018*, 2018, doi: 10.1109/BTAS.2018.8698601.

[39] A. R. Sadeghi, T. Schneider, and I. Wehrenberg, “Efficient privacy-preserving face

recognition," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 5984 LNCS, pp. 229–244, doi: 10.1007/978-3-642-14423-3_16.

[40] V. Naresh Boddeti, "Secure face matching using fully homomorphic encryption," in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems, BTAS 2018*, 2018, doi: 10.1109/BTAS.2018.8698601.

[41] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "OpenFace: A General-Purpose Face Recognition Library with Mobile Applications," *Tech. Rep. C. C. Sch. Comput. Sci.*, vol. 16, no. 118, pp. 1–18, 2016, Accessed: 26-Jul-2020. [Online]. Available: http://cmusatyalab.github.io/openface/.

[42] "Consensus-Driven Propagation in Massive Unlabeled Data for Face Recognition." Accessed: 17-Aug-2020. [Online]. Available: https://github.com/XiaohangZhan/cdp/.

[43] "HayimShaul/ppknn: Privacy Preserving k-ish earest Neighbors." https://github.com/HayimShaul/ppknn (accessed Aug. 15, 2020).

[44] V. N. Boddeti, "Secure Face Matching Using Fully Homomorphic Encryption," May 2018, Accessed: 18-Aug-2020. [Online]. Available: http://arxiv.org/abs/1805.00577.

[45] S. Arita and S. Nakasato, "Fully Homomorphic Encryption for Classification in Machine Learning," in *2017 IEEE International Conference on Smart Computing,*

SMARTCOMP 2017, 2017, doi: 10.1109/SMARTCOMP.2017.7947011.

[46] A. Al Badawi, L. Hoang, C. F. Mun, K. Laine, K. Mi, and M. Aung, “PrivFT: Private and Fast Text Classification with Homomorphic Encryption.”

[47] “Design Patterns: Elements of Reusable Object-Oriented Software [Book].” https://www.oreilly.com/library/view/design-patterns-elements/0201633612/ (accessed Aug. 17, 2020).

[48] S. S. Keerthi, S. Sundararajan, K. W. Chang, C. J. Hsieh, and C. J. Lin, “A sequential dual method for large scale multi-class linear svms,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 408–416, doi: 10.1145/1401890.1401942.

[49] K.-W. Chang, C.-J. Hsieh, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, “A sequential dual method for large scale multi-class linear SVMs,” 2014, doi: 10.1145/1401890.1401942.

[50] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A Library for Large Linear Classification,” 2008. Accessed: 12-Aug-2020. [Online]. Available: http://www.csie.ntu.edu.tw/.

[51] J. H. Cheon, D. Kim, D. Kim, H. Lee, and K. Lee, “Numerical Method for Comparison on Homomorphically Encrypted Numbers.”

[52] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” 2011. Accessed: 16-

Aug-2020. [Online]. Available: http://scikit-learn.sourceforge.net.

[53] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical Method for Comparison on Homomorphically Encrypted Numbers," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11922 LNCS, pp. 415–445, doi: 10.1007/978-3-030-34621-8_15.

[54] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," 2019.

[55] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2625, 2008.

[56] A. Gisbrecht, W. Lueks, B. Mokbel, and B. Hammer, "Out-of-sample kernel extensions for nonparametric dimensionality reduction," in *ESANN 2012 proceedings, 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2012, pp. 531–536, Accessed: 23-Jul-2020. [Online]. Available: http://www.i6doc.com/en/livre/?GCOI=28001100967420.

[57] F. Pedregosa FABIANPEDREGOSA *et al.*, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot," 2011.

Accessed: 16-Aug-2020. [Online]. Available: http://scikit-learn.sourceforge.net.

[58] H. Office of Civil Rights, “HIPAA Administrative Simplification Regulation Text.”

[59] California State Legislature, *AB-375 Privacy: personal information: businesses. (CCPA)*, no. 375. California, USA, 2018.

[60] T. Zarsky, “Incompatible: The GDPR in the Age of Big Data,” *Seton Hall Law Rev.*, vol. 47, no. 4, p. 2, 2017.

Appendix A – Source Code

Any modification to existing projects has been sent to the maintainers of the open-source projects referenced in this thesis and can be found incorporated into those codebases ([36], [43]).

A.1 – Simulation Code for ppKNN and simulation and implementation of ppSVM

```
from collections import Counter, namedtuple, defaultdict
import csv

from scipy import stats

import sympy

import random
import time

from math import prod
import json
from tqdm import tqdm

from sklearn.svm import LinearSVC, SVC
from sklearn import svm
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import numpy as np

import Pyfhel

VERBOSE = False

BaseP = namedtuple('BaseP', ['high', 'low'])
def _basep(n, p):
    return BaseP(int(n//p), int(n%p))


# simulation code
```

```
def KNN(S, q, classes, k):
    """
    returns the most common class in the k elements with the smalled l1 distance to `q`
    """
    return Counter(
        c for s, c in
        sorted(
            ((s, c) for s, c in zip(S, classes)),
            key=lambda s: sum(abs(ss - qq) for ss, qq in zip(s[0], q))
        )[:k]
    ).most_common(1)[0][0]

def KishNN(S, q, classes, k, p):
    n = len(S)
    mult(n)
    X = [sum(abs(ss - qq) for ss, qq in zip(s, q)) % p for s in S]    # l1 norm distances
for each X
    # approximate mu
    mu = (sum(X) // len(X)) % p # assume an arbitrarily low delta, achieves best theoretical
accuracy.

    # base-p rep of mu*2
    poly(p, len(X)/64) + mult(len(X))
    bp_mu_sq = _basep(mu ** 2, p)

    # base-p rep of avg(mu*2)
    poly(p, len(X)/64) + mult(len(X))
    bp_avg_sq = _basep(sum((x**2 for x in X))/n, p)

    # sigma = approximate
    _highdiff = (bp_mu_sq.high - bp_avg_sq.high)
    add()
    mult(2) + add(2)
    if _highdiff == 0:
        sigma = abs(bp_mu_sq.low - bp_avg_sq.low) ** .5
    elif _highdiff == 1:
        sigma = abs(bp_mu_sq.low - bp_avg_sq.low + p) ** .5
    else:
        sigma = abs(p*(bp_mu_sq.high - bp_avg_sq.high)) ** .5
    sigma = int(sigma)
    sigma %= p

    # Threshold
    invnorm = stats.norm.ppf(k/n)
    mult(1) + add(1)
    T = int(mu + sigma * invnorm)
    T %= p

    # class = class if within threshold
    C = [0 for _ in range(len(set(classes)))]
    for x, cls in zip(X, classes):
        poly(p, n=len(S))
        C[cls] += x < T
    if VERBOSE:
        print("""
        q: {q}
        k: {k},
        n: {n},
        mu: {mu},
        bp_mu_sq: {bp_mu_sq},
        bp_avg_sq: {bp_avg_sq}
        sigma: {sigma}
        invnorm: {invnorm}
        T: {T}
        C: {C}""".format(**locals()))

    # argmax
    poly(p, n=len(C))
    return C.index(max(C))

def _compute_Ts(S, p, k):
    """
    precompute the min, max and average value of T
    """
    q = S[0]
    dists = sorted([sum(abs(ss - qq) for ss, qq in zip(s, q)) % p for s in S])
    minT, maxT = dists[k+1:k+3]
    highest_low = minT
    lowest_high = maxT
    average_Ts = []
    for q in tqdm(S):
        minT, maxT = sorted([sum(abs(ss - qq) for ss, qq in zip(s, q)) % p for s in
S])[k+1:k+3]
        average_Ts.append(minT + maxT / 2)
        highest_low = max(highest_low, minT)
        lowest_high = min(lowest_high, maxT)
    return tuple(map(int, (highest_low % p, average_Ts[len(S)//2]%p, (lowest_high % p) +
1)))

def StaticKishNN(S, q, classes, k, p, cached_Ts=None, nullable=False):
    n = len(S)
    mult(n)
    X = [sum(abs(ss - qq) for ss, qq in zip(s, q)) % p for s in S]    # l1 norm distances
for each X
    # approximate mu
    mu = (sum(X) // len(X)) % p # assume an arbitrarily low delta, maximizes accuracy

    T1, T2, T3 = cached_Ts# or _compute_Ts(S, p, k)

    # class = class if within threshold
    C1 = [0 for _ in range(len(set(classes)))]
    poly(p, n)
    for x, cls in zip(X, classes):
        C1[cls] += x < T1
    # class = class if within threshold
    C2 = [0 for _ in range(len(set(classes)))]
    for x, cls in zip(X, classes):
        C2[cls] += x < T2
    # class = class if within threshold
    C3 = [0 for _ in range(len(set(classes)))]
    for x, cls in zip(X, classes):
        C3[cls] += x < T3

    # argmax
    candidates = C1.index(max(C1)), C2.index(max(C2)), C3.index(max(C3))
    mult(3)
    val, count = Counter(candidates).most_common(1)[0]

    if VERBOSE:
        print("""
        q: {q}
        k: {k},
        n: {n},
        T: {T1}, {T2}, {T3}
        C1: {C1}
        C2: {C2}
        """.format(**locals()))

    if count > 1:
        return val
    else:
        return None if nullable else candidates[1]

def _compute_SVM_vectors(S, classes):
    pass # return vectors, classMasks, classXORs


# # Experiments
# ## Helper

def gen_grid_fn(S, res):
    d = len(S[0])
    ranges = [None for _ in range(d)]
    for i in range(d):
        ranges[i] = min(s[i] for s in S), max(s[i] for s in S)
    def grid_fn(x):
        return [
            round((res * (x[i] - ranges[i][0]))/(ranges[i][1] - ranges[i][0]))
            for i in range(d)
        ]
    return grid_fn

def gen_p(S, res):
    d = len(S[0])
    return sympy.nextprime(res * d)

def grid_and_p(S, q, res):
    grid_fn = gen_grid_fn(S, res)
    grid = list(map(grid_fn, S))
    grid_q = grid_fn(q)
    p = gen_p(S, res)
    return grid, grid_q, p

def get_results(res_list):
    correct_counts = defaultdict(int)
    incorrect_counts = defaultdict(int)
    skip_counts = defaultdict(int)
    for res in res_list:
        for key, value in res.items():
            correct_counts[key] += value == res['real_class']
            incorrect_counts[key] += value is not None and value != res['real_class']
            skip_counts[key] += value is None
    return correct_counts, incorrect_counts, skip_counts

def print_results(res_list):
    correct_counts, incorrect_counts, skip_counts = get_results(res_list)
    n = len(res_list)
    for key in correct_counts.keys():
        print("{key:20s}:   \t{perc:03.2%}   correct      \t   {incperc:03.2%}   error   \t
{skipperc:03.2%} skip".format(
            key=key,     perc=correct_counts[key]/n,     incperc=incorrect_counts[key]/n,
skipperc=skip_counts[key]/n))

def load_data(dataset):
    path = dataset_paths[dataset]
    with open(path) as f:
        raw = list(csv.reader(f))
        S = [list(map(float, s[:-1])) for s in raw]
        classes = list(map(int, (s[-1] for s in raw)))
    if VERBOSE:
        print("len: ", len(S), " - d: ", len(S[0]), " - classes: ", len(set(classes)))
    return raw, S, classes
dataset_paths = {
    'breast_cancer_2d':
'/~/_M.Eng/data/breast_cancer/breast_cancer_classification_2d.csv',
    'breast_cancer_3d':
'/~/_M.Eng/data/breast_cancer/breast_cancer_classification_3d.csv',
    'breast_cancer_30d':
'/~/_M.Eng/data/breast_cancer/breast_cancer_classification_30d.csv',
    'cars': '/~/_M.Eng/data/cars/car.csv',
    'cars2': '/~/_M.Eng/data/cars/car2.csv',
    'faces2d': '/~/_M.Eng/data/yaleExtendedB/YaleExtendedB_LDA_2d_with_classlabels.csv',
    'faces3d': '/~/_M.Eng/data/yaleExtendedB/YaleExtendedB_LDA_3d_with_classlabels.csv',
    'faces128d': '/~/_M.Eng/data/yaleExtendedB/YaleExtendedB_128d_with_classlabels.csv',
}

def run_all_KNN(grid_S, grid_q, S, q, classes, p, k, correct_class, cached_Ts):
    to_ret = run_all_KNN_raw(S, q, classes, k, correct_class)
    to_ret.update(run_all_KNN_grid(grid_S, classes, grid_q, p, k, cached_Ts))
    return to_ret

def run_all_KNN_raw(S, q, classes, k, correct_class):
    return {
        "real_class": correct_class,
        "KNN": KNN(S, q, classes, k),
    }
def run_all_KNN_grid(grid_S, classes, grid_q, p, k, cached_Ts):
    return {
        "KishNN": KishNN(grid_S, grid_q, classes, k, p),
        "StaticKishNN": StaticKishNN(grid_S, grid_q, classes, k, p, cached_Ts=cached_Ts),
        "StaticKishNNNullable": StaticKishNN(grid_S, grid_q, classes, k, p, nullable=True,
cached_Ts=cached_Ts),
    }

def timing(fn, args=[], kwargs={}):
    measurements = []
    for _ in range(2):
        start = time.time()
        fn(*args, **kwargs)
        end = time.time()
        measurements.append(end - start)
    return measurements

def time_all_KNN(grid_S, grid_q, S, q, classes, p, k, correct_class, cached_Ts):
    to_ret = run_all_KNN_raw(S, q, classes, k, correct_class)
    to_ret.update(run_all_KNN_grid(grid_S, classes, grid_q, p, k, cached_Ts))
    return to_ret

def time_all_KNN_raw(S, q, classes, k, correct_class):
    return {
        "real_class": timing(lambda x: x, [correct_class]),
        "KNN": timing(KNN, [S, q, classes, k]),
    }
def time_all_KNN_grid(grid_S, classes, grid_q, p, k, cached_Ts):
    return {
        "KishNN": timing(KishNN,[grid_S, grid_q, classes, k, p]),
        "StaticKishNN":   timing(StaticKishNN,[grid_S,   grid_q,   classes,   k,   p],
dict(cached_Ts=[1,2,3])),
        "StaticKishNNNullable":  timing(StaticKishNN,[grid_S,  grid_q,  classes,  k,  p],
dict(nullable=True, cached_Ts=[1,2,3])),
    }

def try_KNN(k, res, S, n=None):
    p = gen_p(S, res)
    grid_fn = gen_grid_fn(S, res)
    grid_S = list(map(grid_fn, S))
    res_list = []
    VERBOSE = 0
    for ii in range(n or len(S)):
        i = random.randrange(len(S)) if n else ii
        S_sample = S[:i] + S[i+1:]
        classes_sample = classes[:i] + classes[i+1:]
        q = S[i]
        grid_q = grid_fn(q)
        grid_S = grid_S[:i] + grid_S[i+1:]
        correct_class = classes[i]
        res = time_all_KNN(grid_S, grid_q, S, q, classes, p, k, correct_class, [1,2,3])
        res_list.append(res)
        if VERBOSE:
            print(res)
    return res_list

def backup(obj, name):
    with                          open("/Users/stein/Dropbox/_M.Eng/data/experiments/{}-
macbook.json".format(name), 'w') as f:
        json.dump(obj, f)


ts = time.time()
results = dict()
#'breast_cancer_2d', 'breast_cancer_3d',  'cars',
for dataset in ['cars2', 'faces2d', 'faces3d']:
    for res in [20, 120, 200, 300]:
        for k in [4,16,64,256]:
            print(dataset,res,k)
            raw, S, classes = load_data(dataset)
            res_list = try_KNN(k, res, S, n=1)
            results["{},{},{}".format(dataset, res, k)] = res_list
            backup(results, 'KNN_experiments{}'.format(ts))


# ### SV

def generate_SVM_vectors(X, classes, d=None):
    # return vectors, coef
    if d and d < len(X[0]):
        X = LDA(n_components=d).fit_transform(X, classes)
    classifier = LinearSVC(C=1, dual=False, max_iter=10000)
    classifier.fit(X, classes)
    return classifier.coef_, len(classifier.coef_[0]), X, classifier

def gen_p(S, res):
    d = len(S[0])
    return sympy.nextprime(res * d)

def gen_grid_scalefn(S, res):
    bound = max(abs(ss) for s in S for ss in s)
    p = gen_p(S, res)
    def grid_fn(x):
        return [round(res * ((xx/bound))) for xx in x]
    return grid_fn

def ppSVM(W, q, p):
    class_idx = np.dot(q, (np.array(W)//2).T)
    return (class_idx + p//2).argmax()

def encrypted_ppSVM(W, q):
    enc_W = [[HE.encryptInt(ww) for ww in w] for w in W]
    enc_q = [HE.encryptInt(qq) for qq in q]
    class_idx = []
    for w in enc_W:
        class_idx.append(encrypted_dot(w, enc_q))
    return encrypted_ArgMax(class_idx)

def encrypted_dot(w, q):
    output = []
    for ww, qq, in zip(w, q):
        output.append(debuggify(ww * qq))
    to_ret = HE.encryptInt(0)
    for o in output:
        to_ret = debuggify(to_ret + o)
    return to_ret

def encrypted_ArgMax(C):
    to_ret = HE.encryptInt(0)
    for j in range(len(C)):
        cur_prod = HE.encryptInt(j)
        smallest = isSmallest(C, j)
        to_ret = debuggify(to_ret + cur_prod * smallest)
    return to_ret

def ep(ctxt, name=""):
    print(name, HE.decryptInt(ctxt), HE.noiseLevel(ctxt))

def debuggify(ctxt):
    if DEBUGGING:
        # overagressive relinearization to fix weird PyFel thing. Do not use in
        # actual test, this both removes noise and adds time.
        tmp = HE.decryptInt(ctxt)
        print(tmp)
        return HE.encryptInt(tmp)
    return ctxt

def isSmallest(C, j):
    vals = [stubbed_isSmaller(c, C[j]) for i, c in enumerate(C) if i != j]
    cur = HE.encryptInt(vals[0])
    for v in vals[1:]:
        cur = debuggify(cur * v)
    return cur

def stubbed_isSmaller(ci, cj):
    x = HE.decryptInt(cj) - HE.decryptInt(ci)
    coefs = _IS_NEGATIVE_MACLAUREN
    return HE.encryptInt(1 if x > 0 else 0)

_IS_NEGATIVE_MACLAUREN                                                          =
json.load(open("~/_M.Eng/data/experiments/is_smaller_poly.json"))
def encrypted_isSmaller(ci, cj):
    print((cj-ci)._encoding)
    return HE.polyEval_double(cj - ci, _IS_NEGATIVE_MACLAUREN, in_new_ctxt=True)

def run_all_SVM(W_grid, q_grid, W, q, p, cls, correct_class):
    to_ret = run_all_SVM_raw(q, correct_class, clf)
    to_ret.update(run_all_SVM_grid(W_grid, q_grid, p))
    return to_ret

def run_all_SVM_raw(q, correct_class, clf):
    return {
        "real_class": int(correct_class),
        "SVM": int(clf.predict([q])[0]),
        "continuousSVM": int(np.dot(q, np.array(W).T).argmax())
    }

def run_all_SVM_grid(W, q, p):
    return {
        "ppSVM": int(ppSVM(W, q, p)),
        "enc_ppSVM": int(HE.decryptInt(encrypted_ppSVM(W, q)))
    }

def time_all_SVM(W_grid, q_grid, W, q, p, cls, correct_class):
    to_ret = run_all_SVM_raw(q, correct_class, clf)
    to_ret.update(run_all_SVM_grid(W_grid, q_grid, p))
    return to_ret

def time_all_SVM_raw(q, correct_class, clf):
    return {
        "real_class": timing(int(correct_class)),
        "SVM": timing(int(clf.predict([q])[0])),
        "continuousSVM": timing(int(np.dot(q, np.array(W).T).argmax()))
    }

def time_all_SVM_grid(W, q, p):
    return {
        "ppSVM": timing(int(ppSVM(W, q, p))),
        "enc_ppSVM": timing(int(HE.decryptInt(encrypted_ppSVM(W, q))))
    }
```

```
HE = Pyfhel.Pyfhel()
HE.contextGen(p=2)
HE.keyGen()
import datetime
ts = datetime.datetime.now()
dataset_list = ['faces2d', 'breast_cancer_30d', 'cars2']
with tqdm(total=len(dataset_list) * 2 * 3) as pbar:
    for dataset in dataset_list: # 'faces128d'
        for d in [2, 3]:
            for grid_size in [10, 120, 1000]:
                pbar.update(1)
                pbar.set_description("dataset: {}, d: {}, grid: {}".format(dataset, d,
grid_size))

                raw, S, classes = load_data(dataset)
                n_classes = len(set(classes))
                if d > n_classes - 1:
                    continue
                W, d, X, clf = generate_SVM_vectors(S, classes, d=d)
                W = [[ww for ww in w] for w in W]
                grid_fn = gen_grid_scalefn(S+W, grid_size)
                p = gen_p(S+W, grid_size)
                W_grid = list(map(grid_fn, W))
                res_list = []

                for _ in range(10):
                    i = random.randrange(len(S))
                    q = X[i]
                    correct_class = classes[i]

                    svm_results  =  time_all_SVM(W_grid,  grid_fn(q),  W,  q,  p,  clf,
correct_class)

                    res_list.append(res)
                    if VERBOSE:
                        print(res)
                    svm_results["{},{},{}".format(dataset, d, grid_size)] = res_list
                    backup(svm_results, "svm_results{}".format(ts))
```