Methodology Development for Improving the Performance of Critical Classification Applications

Sharmin Afrose

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Applications

December 5, 2022

Blacksburg, Virginia

Keywords: Software Security, Machine Learning Bias

Methodology Development for Improving the Performance of Critical Classification Applications

Sharmin Afrose

(ABSTRACT)

People interact with different critical applications in day-to-day life. Some examples of critical applications include computer programs, anonymous vehicles, digital healthcare, smart homes, etc. There are inherent risks in these critical applications if they fail to perform properly. In my dissertation, we mainly focus on developing methodologies for performance improvement for software security and healthcare prognosis. Cryptographic vulnerability tools are used to detect misuses of Java cryptographic APIs and thus classify secure and insecure parts of code. These detection tools are critical applications as misuse of cryptographic libraries and APIs causes devastating security and privacy implications. We develop two benchmarks that help developers to identify secure and insecure code usage as well as improve their tools. We also perform a comparative analysis of four static analysis tools. The developed benchmarks enable the first scientific comparison of the accuracy and scalability of cryptographic API misuse detection. Many published detection tools (CryptoGuard, CrySL, Oracle Parfait) have used our benchmarks to improve their performance in terms of the detection capability of insecure cases. We also examine the need for performance improvement for healthcare applications. Numerous prediction applications are developed to predict patients' health conditions. These are critical applications where misdiagnosis can cause serious harm to patients, even death. Due to the imbalanced nature of many clinical datasets, our work provides empirical evidence showing various prediction deficiencies in a typical machine learning model. We observe that missed death cases are 3.14 times higher than missed survival cases for mortality prediction. Also, existing sampling methods and other techniques are not well-equipped to achieve good performance. We design a double prioritized (DP) technique to mitigate representational bias or disparities across race and age groups. we show DP consistently boosts the minority class recall for underrepresented groups, by up to 38.0%. Our DP method also shows better performance than the existing methods in terms of reducing relative disparity by up to 88% in terms of minority class recall. Incorrect classification in these critical applications can have significant ramifications. Therefore, it is imperative to improve the performance of critical applications to alleviate risk and harm to people.

Methodology Development for Improving the Performance of Critical Classification Applications

Sharmin Afrose

(GENERAL AUDIENCE ABSTRACT)

We interact with many software using our devices in our everyday life. Examples of software usage include calling transport using Lyft or Uber, doing online shopping using eBay, using social media via Twitter, check payment status from credit card accounts or bank accounts. Many of these software use cryptography to secure our personal and financial information. However, the inappropriate or improper use of cryptography can let the malicious party gain sensitive information. To capture the inappropriate usage of cryptographic functions, there are several detection tools are developed. However, to compare the coverage of the tools, and the depth of detection of these tools, suitable benchmarks are needed. To bridge this gap, we aim to build two cryptographic benchmarks that are currently used by many tool developers to improve their performance and compare their tools with the existing tools. In another aspect, people see physicians and are admitted to hospitals if needed. Physicians also use different software that assists them in caring the patients. Among this software, many of them are built using machine learning algorithms to predict patients' conditions. The historical medical information or clinical dataset is taken as input to the prediction models. Clinical datasets contain information about patients of different races and ages. The number of samples in some groups of patients may be larger than in other groups. For example, many clinical datasets contain more white patients (i.e., majority group) than Black patients (i.e., minority group). Prediction models built on these imbalanced clinical data may provide inaccurate predictions for minority patients. Our work aims to improve the prediction accuracy for minority patients in important medical applications, such as estimating the likelihood of a patient dying in an emergency room visit or surviving cancer. We design a new technique that builds customized prediction models for different demographic groups. Our results reveal that subpopulation-specific models show better performance for minority groups. Our work contributes to improving the medical care of minority patients in the age of digital health. Overall, our aim is to improve the performance of critical applications to help people by decreasing risk. Our developed methods can be applicable to other critical application domains.

Dedication

To my parents (Mahmuda Khanam and Md. Zowadul Munir), my brother (Saeed Anwar)

& my husband (Majbah Uddin)

Acknowledgments

It was a hard decision for me to leave my parents in my home country and come alone to a new country to start a five-year journey. Now, I look back and thank Almighty Allah for following the right decisions. This experience makes me more observant, wiser, proactive, and stronger.

I would like to express my deepest gratitude to my parents. They respected every decision I made and encouraged me, talked to me, and motivated me along the way. I am also grateful to my brother who was always there for me whenever I needed him.

I would like to express my sincere appreciation to my advisor Dr. Danfeng (Daphne) Yao. After I got admission to Virginia Tech, she reached out to me and encouraged me to pursue Ph.D. under her supervision. I was amazed by her patience, her soothing attitude as well as guidance when I started writing my first first-authored research paper. Along the way, she taught me how to be a good researcher, to see the real-world problem that needs to be solved, and be proud of the impact I made through my solutions to these problems. I really admire her contribution in improving the environment for female professionals. She is a perfect role model for me to follow.

I would like to thank my Committee members, Dr. T. M. Murali, Dr. Na Meng, Dr. B. Aditya Prakash, and Dr. Sharon Xiaolei Huang. They gave lots of insightful suggestions that helped me for completing my Ph.D. research. I like to thank all our collaborators and lab mates for contributing to my research work. Specifically, I am grateful to Dr. Sazzadur Rahaman, Dr. Ya Xiao, and Miles Frantz for contributing to the work included in Chapter 3. Also, I am grateful to Wenjia Song, Dr. Chang Lu, and Dr. Charles B. Nemeroff for contributing to the work included in Chapter 4.

I express my gratitude to my dearest friends in Blacksburg including Nazia Munir, Tahmina Hossain Ahmed, Sadia Ahmed, Rezwana Islam Linda, Afroza Alam, Mahbubur Rahman Kaisar, Kazi Moshiur Rahman, and many more. I am very grateful for their support, inspiration, and friendship. I am so grateful to them for accompanying me through my joyful and challenging moments.

A very special thank goes to my husband, Majbah Uddin. His calmness, compassion, and constant support always brought peace to my mind during stressful situations and persuaded me to keep moving forward.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we define the research problems, describe our contributions, and specify the dissertation organization.

1.1 Problem Definition

Critical applications are applications that require special attention due to their risk and magnitude of harm for people if they fail to perform properly [10]. Examples of some critical applications are anonymous vehicles, healthcare prediction applications, computer programs, and application smart homes devices. Buggy computer programs can lead to data breaches as well as data leakage of sensitive information [26, 27, 28, 130]. Erroneous features of autonomous cars could lead to severe car crashes that can threaten human lives [90, 126]. Misdiagnosis using digital healthcare applications can lead to the death of patients [13, 66, 85, 138]. Hackers can gain unauthorized access to smart devices which can cause privacy and security issues [34, 52, 129]. In our dissertation, we mainly focus on software security and healthcare prognosis applications.

For applications in software security, usage of insecure applications can cause sensitive data leakage yielding privacy and security implications. Many software (Lyft, Airbnb, eBay, American Express, Dropbox, Twitter, etc.) incorporates Java cryptographic APIs [2]. Oftentimes, cryptographic API misuses are introduced in programs due to complex API designs [35, 111], the lack of cybersecurity training [104], insecure code generation tools [115] and insecure/misleading suggestions in Stack Overflow [36, 104]. These misuses attributes to many cyber attacks. To detect cryptographic API misuses, several cryptographic vulnerability detection tools are developed [30, 95, 122]. However, a lack of suitable benchmarks hinders scientific performance evaluation of cryptographic API misuses detection.

To address the aforementioned concerns, we first focus on detecting cryptographic vulnerabilities that are often misused by developers. First, we identify 18 Java cryptographic categories. Based on these cryptographic categories, we developed two benchmarks that facilitate first scientific accuracy and scalability comparison of cryptographic API misuses detection. These benchmarks contain unit test cases as well as real-world large codes with interesting program properties. The main challenge of these benchmarks is to include every possible interesting case. We obtain suggestions from reviewers as well as our benchmark users and update the benchmark at regular intervals. Our benchmark is currently actively used for improvement and comparison in Oracle bug checker Parfait, CryptoGuard, and CrySL.

Healthcare prognosis is another critical application where misdiagnosis can cost lives. Many hospitals [17, 18] use predictive analytics for monitoring patients' health status and preventing emergencies. Clinical datasets are intrinsically imbalanced due to the naturally occurring frequencies of data [89]. Data imbalance is a major cause of biased prediction results that may cause serious consequences for some patients [114, 119, 142]. To remove bias, several sampling techniques exist [65, 83, 91, 103]. However, they are not designed to address demographic subgroup biases, as they sample the entire minority class.

The second part of our dissertation focuses on reducing subpopulation-specific bias in healthcare prognosis. We first focus on analyzing clinical predictions from large medical datasets to show multiple types of disparities, including the metric disparity, and performance disparity among racial, and age groups. We also present a new technique, double prioritized (DP) bias correction that focuses on improving the prediction accuracy of specific demographic groups through sample enrichment. To show the effectiveness of DP, we systematically compare DP with other existing sampling techniques. The work is to aware researchers to be conscious of minority subpopulations and lessen life-threatening prediction mistakes.

1.2 Contributions

My dissertation work aims to measure two critical systems, point out the limitations and propose possible solution directions. To achieve this goal, our completed made contributions from two critical aspects. 1) *the real-world cryptographic benchmarks generation to point out the limitations of current vulnerability detection tools*, 2) *bias in subpopulation-specific machine learning prognosis for underrepresented patients.* The shared challenges of the research works are as follows.

- Interpretation of the output results from critical systems is a challenging task. The vulnerability detection reports generated from different existing detection tools are in various formats. Therefore, we had to manually check every single error in every report to understand whether an error is accurately captured or not. The machine learning prediction model's outcome is also challenging to interpret when an exceptional outcome arises. To solve this, we compute feature importance and other extensive analysis to interpret a specific outcome.

- Choosing the correct performance metric is another challenging task. We mainly focus on the rare event of interest (e.g., insecure API usage in code, death event of a patient in ICU). We compute rare event recall and precision as performance metrics for both critical applications. For healthcare disparity research work, we also focus on several other metrics, e.g., MCC (Matthew's correlation coefficient), F1 score, balanced accuracy, and precision-recall curve value, to more clearly understand the impact of prediction outcomes.

- To execute the existing works, we need to create the required environment (e.g., specific library versions). For codes in Python, we built a separate environment using Anaconda. If any library is not supported any longer, we fix the code to make it compatible with the existing library version.
- The MIMIC dataset is huge data containing over forty thousand patients' information. To run the prediction model using the MIMIC dataset, we used the CS Research GPU cluster as we needed more RAM to run the code. Still, the training time is very long. On average, the training time is around 10 minutes for each epoch to run the original model of the mortality prediction task.
- For the MIMIC dataset, our model input is time-series clinical data which is 3D data of a large number of patients. Therefore, it is challenging to use this dataset for several existing bias correction techniques.

Next, we will discuss our overall contribution. To assist the vulnerability detection tool developers, our contributions include:

- We provide a benchmark named CryptoAPI-Bench, which consists of 181 test cases covering 18 types of Cryptographic and SSL/TLS API misuse vulnerabilities. CryptoAPI-Bench utilized various interesting program properties (e.g., field-, context-, and path sensitivity) to produce a diverse set of test cases. Our benchmark is open-sourced and can be found on GitHub [37].

- We provide another benchmark named ApacheCryptoAPI-Bench for checking the scalability property of the cryptographic vulnerability detection tools. We document 121 test cases covering 12 types of Cryptographic and SSL/TLS API misuse vulnerabilities from 10 real-world Apache projects. Detailed information regarding ApacheCryptoAPI-Bench can be found on GitHub [38].

- We evaluate four static analysis tools that are capable of detecting cryptographic misuse vulnerabilities. Our experimental evaluation revealed some interesting insights. For complex cases, specialized tools (e.g., CryptoGuard, CrySL) detect more cryptographic misuses and cover more rules than general-purpose tools (e.g., SpotBugs, Tool A). Currently, none of these tools supports path-sensitive analysis.

To evaluate disparity and reduce bias in subpopulation-specific machine learning prognosis for underrepresented patients, our contribution includes:

- We provide empirical evidence showing severe racial and age prediction disparities using two large medical datasets (MIMIC III and SEER) and the deceptive nature of common conventional metrics such as overall accuracy and AUC-ROC.

- We evaluate the bias-correction ability of sampling methods (Gamma, ADASYN, SMOTE, replicated oversampling, Nearmiss-1, Nearmiss-3, Distant, random oversampling, stratified sampling, and sampling using a generative neural network), model reweighting method, model objective constraint method (Seldonian algorithm).

- We propose a new double prioritized (DP) bias correction technique and an equivalent prioritized reweighting technique for reducing disparity and increasing fairness among demographic population groups.

We discuss our developed methodology, analysis techniques, and insights in detail in the following chapters. Our developed methodologies can be applied to other critical application domains.

1.3 Dissertation Organization

The structure of this dissertation is as follows. Chapter 2 is the literature review about the related studies. Chapter 3 introduces scientific comparisons of the accuracy and scalability of cryptographic API misuse detection using two benchmarks. Chapter 4 focuses on empirical evidence of severe racial and age prediction bias in healthcare prognosis and the bias-correction ability of proposed double prioritized (DP) sampling methods compared to other state-of-the-art sampling methods. Chapter 5 concludes the dissertation and discusses future works.

Chapter 2

Review of Literature

In this chapter, we discuss existing literature on cryptographic vulnerability detection benchmarks and machine learning prediction bias.

2.1 Cryptographic Vulnerability Detection Benchmarks

AndroZoo++ [99] is a collection of over eight million Android apps [8] that drives a lot of security, software engineering, and malware analysis research. However, vulnerabilities in these apps are not documented, hence not suitable for vulnerability detection benchmarking purposes.

DroidBench [46], a benchmark containing vulnerable android apps, fills the gap by providing specific vulnerability locations within the benchmark. To date, DroidBench is one of the most popular benchmarks to evaluate the performance of vulnerability detection tools in Android literature. In total, DroidBench has 119 APKs from 13 categories (Commit id 0fe281b). Categories include vulnerabilities that use field and object sensitivity, inter-app communication, inter-component communication, android life-cycle, reflection, etc. However, DroidBench *i)* does not cover cryptographic misuse vulnerabilities and *ii)* does not have source code. To the best of our knowledge, Ghera [107] is the only Android app benchmark that contains app source code. Like DroidBench, most of the vulnerabilities in Ghera are specific to Android apps and barely contain any cryptographic misuse vulnerabilities.

To be specific, CryptoAPI-Bench and Ghera have only 2 types of vulnerabilities in common.

OWASP benchmark [64] is fundamentally designed to capture eleven cybersecurity vulnerabilities. However, among the detected vulnerabilities, it builds to address only three Java cryptographic vulnerabilities, i.e., weak encryption algorithms, weak hash algorithms, and a weak random number. SonarSource [29] released a set of vulnerability samples that can be useful to check for coverage of vulnerability categories. A verification tool for five common audit controls is proposed for ensuring continuous compliance [92]. MASC framework [44] is designed to evaluate static analysis tools using mutation testing. However, it considers limited complex cases.

The DaCapo benchmarks [59] are designed to evaluate the performance of various components of Java virtual machine (JVM), Garbage collection (GC), Just-in-time (JIT) compiler itself. BugBench [100] is a benchmark to find C/C++ bugs that contain 17 real-world applications. BugBench mostly covers various memory, concurrency, and semantic bugs. To detect bugs in the multi-threaded Java programs, a benchmark and framework have been proposed [74, 82]. Coding practice and recommendations are provided for 28 enterprise applications that use Spring security framework [86]. ManyBugs and IntroClass benchmarks are designed to evaluate various C/C++ code repair techniques [97]. Most of the defects in ManyBugs and IntroClass do not impact security, e.g., in the ManyBugs benchmark, more than half of the instances impact correctness, not necessary security.

2.2 Bias in Machine Learning

A widely used bias-correction approach to the data imbalance problem is sampling. Oversampling, e.g., replicated oversampling (ROS), is to balance a dataset by adding samples of the minority class; undersampling, e.g., random undersampling (RUS), is to balance a dataset by removing samples of the majority class [137]. An improvement is the K–nearest neighbor (K–NN) classifier–based undersampling technique [103] (e.g., NearMiss1, NearMiss2, NearMiss3, Distant) that selects samples from the majority class based on distance from minority class samples. State-of-the-art solutions are all oversampling methods, including Synthetic Minority Over-sampling Technique (SMOTE) [65], Adaptive Synthetic Sampling (ADASYN) [83], and Gamma [91]. All three methods generate new minority points based on existing minority samples, namely using linear interpolation [65], gamma distribution [91], or at the class border [83]. However, existing sampling techniques are not designed to address subgroup biases, as they sample the entire minority class. These methods do not differentiate demographic subgroups (e.g., Black patients or young patients under 30). Thus, it is unclear how well existing sampling solutions reduce accuracy disparity.

Besides sampling, reweighting is another existing method for mitigating the data imbalance and correcting the prediction bias. Existing studies showed that sampling performance is more effective than reweighing from both theoretical and experimental perspectives for neural networks [45, 128].

Ribeiro et al. [124] evaluate the consistency of the question-answer model built on natural language processing. They generate implications and ask questions differently in order to see whether the model can answer the implicated question. Ribeiro et al. [125] propose behavioral testing of the NLP models using different templates. They used different words using the template and see whether the prediction changes from the expected prediction.

Several works are done on fairness testing [42, 78] where changing sensitive attribute (e.g., gender, age, race) information should not change the outcome. These approaches can be more applicable to specific prediction tasks such as hiring decisions [60], loan approval [109] prediction, etc. However, in the medical field, sensitive attributes are important in decision-making.

Chapter 3

Cryptographic API Benchmarks

In this chapter, we specify our developed cryptographic API benchmarks that facilitate scientific in-depth comparisons among existing tools.

3.1 Introduction

Various studies have shown that a vast majority of Java and Android applications misuse cryptographic libraries and APIs, causing devastating security and privacy implications. The most pervasive cryptographic misuses include exposed secrets (e.g., secret keys and passwords), predictable random numbers, use of insecure crypto primitives, vulnerable certificate verification [73, 75, 79, 104, 121, 122].

Several studies showed that the prominent causes for cryptographic misuses are the deficiency in understanding of security API usage [35, 104], complex API designs [35, 111], the lack of cybersecurity training [104], insecure code generation tools [115] and insecure/misleading suggestions in Stack Overflow [36, 104]. The reality is that most developers, with tight project deadlines and short product turnaround time, spend little effort on improving their knowledge or hardening their code for long-term benefits [49]. Recognizing these practical barriers, automatic cryptographic code generation [96], and misuse detection tools [122] play a significant role in assisting developers with writing and maintaining secure code.

The security community has produced several impressive static (e.g., CryptoLint [73], CrySL [95], FixDroid [113], MalloDroid [75], CryptoGuard [122]) and dynamic code screening tools (e.g., Crylogger [118], SMV-Hunter [131], and AndroSSL [76]) to detect API misuses in Java. The static analysis does not require a program to execute, rather it is performed on a version of the code (e.g., source code, intermediate representations or binary). Many abstract security rules are reducible to concrete program properties that are enforceable via generic static analysis techniques [47, 122]. Consequently, static analysis tools have the potential to cover a wide range of security rules. In contrast, dynamic analysis tools require one to execute a program and spend a significant effort to trigger and detect specific misuse symptoms at runtime. Hence, dynamic analysis tools may be limited in their coverage. A code screening tool needs to be scalable with wide coverage. Thus, static analysis-based tools are usually more favorable than their dynamic counterparts.

However, a major weakness of static analysis tools is their tendency to produce false alerts. False alerts substantially diminish the value of a tool. To reduce the number of false positives, most of the static analysis tools offer a trade-off between completeness and scalability [101]. We define *completeness* as the ability to detect all the misuse instances and *scalability* as the ability to induce low computational overhead to analyze large code bases. Designing tools that would produce fewer false positives and false negatives with smaller computational overhead help real-world deployment.

To advance and monitor the scientific progress of domains to produce effective tools, a mechanism for comparative studies is required. Unfortunately, for the automatic detection of cryptographic API misuses, no suitable mechanism or benchmark exists. Such a benchmark needs to have several requirements: *i)* it should cover a wide range of misuse instances. *ii)* it should cover interesting program properties (e.g., flow-, context-, field-, path-sensitivity, etc.) [132, 139]. These are different detection capabilities required for capturing certain vulnerabilities. *iii)* Test cases should be written in easily compilable source codes so that both source code and binary code analysis tools can be easily evaluated.

None of the existing benchmarks follows these criteria (e.g., DroidBench [46], Ghera [107]). For example, DroidBench [46] only contains binaries. Ghera [107] has sources of provided Android apps. However, both DroidBench and Ghera barely cover cryptographic API misuses.

We present two benchmarks for cryptographic API misuses. The first one is CryptoAPI-Bench, a comprehensive benchmark for comparing the quality of cryptographic vulnerability detection tools. It consists of 181 unit test cases covering 18 types of cryptographic misuses. Several test cases include interesting program properties [132, 139]. Flow-sensitive correctly computes and analyzes the order of statements in a program. Path-sensitivity analysis computes different dataflow analysis information dependent on conditional branch statements. The field-sensitive analysis distinguishes two fields containing the same object in a class. A context-sensitive analysis is any interprocedural analysis that analyzes the target of a function call.

The second one is ApacheCryptoAPI-Bench which is built upon 10 real-world Apache projects. It contains early versions of activemq-artemis, deltaspike, directory-server, manifoldcf, meecrowave, spark, tika, tomee, wicket projects. We identify 121 crypto cases in them, including 82 basic cases and 39 advanced cases.

We run CryptoAPI-Bench and ApacheCryptoAPI-Bench on four static analysis tools (i.e., SpotBugs [30], CryptoGuard, CrySL, and Tool A (anonymous) and perform a comparative analysis of these tools. These tools are *i)* capable of detecting cryptographic misuse vulnerabilities and *ii)* open-sourced and/or provide free evaluation license. CrySL and CryptoGuard have open-sourced research prototypes that are actively being maintained to improve their accuracy and coverage. SpotBugs is also an actively maintained open-source project, which is the successor of FindBugs. Tool A is one of the most popular static analysis platforms for decades.

Our main technical contributions are summarized as follows.

- We provide a benchmark named CryptoAPI-Bench, which consists of 181 test cases covering 18 types of Cryptographic and SSL/TLS API misuse vulnerabilities. CryptoAPI-Bench utilized various interesting program properties (e.g., field-, context-, and path sensitivity) to produce a diverse set of test cases. Our benchmark is open-sourced and can be found on GitHub [37].
- We provide another benchmark named ApacheCryptoAPI-Bench for checking the scalability property of the cryptographic vulnerability detection tools. We document 121 test cases covering 12 types of Cryptographic and SSL/TLS API misuse vulnerabilities from 10 real-world Apache projects. Detailed information regarding ApacheCryptoAPI-Bench can be found on GitHub [38].
- We evaluate four static analysis tools that are capable of detecting cryptographic misuse vulnerabilities. Our experimental evaluation revealed some interesting insights. For complex cases, specialized tools (e.g., CryptoGuard, CrySL) detect more cryptographic misuses and cover more rules than general-purpose tools (e.g., SpotBugs, Tool A). Currently, none of these tools supports path-sensitive analysis.

3.2 Background

In this section, we describe Java cryptographic API misuses that are often misused by developers and existing static vulnerability detection tools.

3.2.1 Java Cryptographic API Misuses

We consider 18 Java cryptographic API misuse categories for our benchmarks. We got the insights of these misuse categories from previous literature [95, 113, 122], NIST documents [19, 53, 54], and other blogs [29]. We describe reasons for vulnerability and possible secure solutions for these misuse categories.

1) Cryptographic Keys: For encryption, it is expected to use an unpredictable key using `javax.crypto.spec.SecretKeySpec` API that takes a byte array as input. If the Byte array is constant or hardcoded inside the code, the adversary can easily read the cryptographic key and may obtain sensitive information. Therefore, an unpredictable byte array should be used as a parameter in SecretKeySpec to generate a secure key.

2) Passwords in Password-based Encryption: Password-based Encryption (PBE) is a popular technique of generating a strong secret key using `javax.crypto.spec.PBEKeySpec` API. It takes three parameters (i.e., password, salt, and iteration count). However, if a hardcoded or constant password is used in the code, then malicious attackers may obtain the password and predict the key [73]. Therefore, an unpredictable password should be used as a parameter in PBEKeySpec.

3) Passwords in KeyStore: Cryptographic keys and certificates are sometimes stored using `java.security.KeyStore` API. The KeyStore employs a password to get access to the stored keys and certificates. However, if a hardcoded or constant password is used for KeyStore in the code, it poses a security threat of revealing keys and certificates stored in the KeyStore. Therefore, an unpredictable random password should be used in KeyStore.

4) Credentials in String: Credentials (passwords, secret keys, etc) should not be stored in the String variable. In Java, String is a final and immutable class stored in the heap. More specifically, it exists in the memory until garbage collection. Therefore, sensitive information should not be stored in String[25, 31]. Compared with String, it is highly recommended to use mutable data structures (e.g., byte or char array) for sensitive information and clear it immediately after use. This reduces the window of opportunity for an adversary. [21].

5) Hostname Verifier: HostnameVerifier in `javax.net.ssl.HostnameVerifier` API verifies the hostname by checking the hostname's authentication and identification. In some cases, verify() method of HostnameVerifier class is set to return true by default so that the verification method can quickly get past an exception. However, this arrangement causes a security threat, where URL spoofing [7] attacks can be possible. URL spoofing makes it simpler for numerous cyber-attacks (e.g., identity theft, phishing). In Fig. 3.1, Line 3 returns true without verifying the hostname which is a major source of vulnerability.

```
1 public boolean verify(String hostname, SSLSession sslSession)
2 {
3     return true
4 }
```

Listing 3.1: Skipping hostname verification in the `verify` method of `javax.net.ssl.HostnameVerifier` is insecure

6) Certificate Validation: Empty methods are often implemented to connect quickly and easily with clients while using `javax.net.ssl.X509TrustManager` interface without any certificate validation. In that case, the TrustManager accepts and trusts every entity including the entity that is not signed by a trusted certificate authority. It may cause Man-in-the-middle (MitM) attacks [3, 75].

7) SSL Sockets: `javax.net.ssl.SSLSocket` connects a specific host to a specific port. However, before the connection, the hostname of the server should be verified and authenticated using `javax.net.ssl.HostnameVerifier` API. However, incorrect implementation omits the hostname verification when the socket is created [4, 79].

8) Hypertext Transfer Protocol: HyperText Transfer Protocol (HTTP) sends a request to a server to retrieve a web page. However, HTTP allows hackers to intercept and read sensitive information [55]. Therefore, it is recommended to use HyperText Transfer Protocol Secure (HTTPS) which utilizes a secured socket layer to encrypt sensitive information. In Listing. 3.2, a code snippet of secure and insecure URL usage using `java.net.URL` API is presented.

```
🐞 URL urlInsecure = "http://time.com/"
   URL urlSecure = "https://www.google.com";
```

Listing 3.2: Use of HTTP URL in `java.net.URL` API is inherently insecure

9) Pseudorandom Number Generator (PRNG): The generation of a pseudorandom number using `java.util.Random` is vulnerable as the generated random number is not completely random, because it uses a definite mathematical algorithm (Knuth's subtractive random number generator algorithm [93]) that is proven to be insecure. To solve the problem, `java.security.SecureRandom` provides non-deterministic and unpredictable random numbers.

```
🐞 Random r = new Random();
   SecureRandom sr = new SecureRandom();
   int insecureSeed = r.nextInt();
   int secureSeed = sr.nextInt();
   byte [] bytes = {(byte) 100};
🐞 sr.setSeed(bytes);
   int insecureSeed2 = sr.nextInt();
```

Listing 3.3: Generating seeds using `java.util.Random` is insecure. Random secure seeds can be generated using `java.security.SecureRandom` API

10) Seeds in Pseudorandom Number Generator (PRNG) A constant or static seed in `java.security.SecureRandom` can cause same outcome on every run. Therefore, developers should use a non-deterministic random seed.

11) Salts in Password-based encryption: `javax.crypto.spec.PBEParameterSpec` API takes salt as one of the parameters for Password-based encryption. Using constant or static salts increases the possibility of a dictionary attack. The salt should be a random number that produces a random and unpredictable key. In Fig. 3.4, Line 2 takes a static/constant salt that is insecure to be used in PBEParameterSpec.

```
1    PBEParameterSpec pbeParamSpec = null;
2    byte[] salt = {(byte) 0xa2}
3    int count = 20;
4    pbeParamSpec = new  PBEParameterSpec(salt, count);
```

Listing 3.4: `javax.crypto.spec.PBEParameterSpec` API usage is insecure if iteration count is less than 1000 and salt is constant or predictable

12) Mode of Operation: The Electronic Codebook (ECB) mode of operation is insecure to use in `javax.crypto.Cipher` as ECB-encrypted ciphertext can leak information about the plaintext. Instead of ECB, Cipher Block Chaining (CBC) or Galois/Counter Mode (GCM) is more secure to use. Table 3.1 provides a list of insecure and secure modes of operation.

13) Initialization Vector (IV): The initialization vector (IV) is used during encryption and decryption with several modes of operation. Static/constant initialization vector introduces vulnerabilities for CBC mode of operation. Therefore, it is suggested to use an unpredictable random initialization vector in `crypto.spec.IvParameterSpec API`. Note that, for several modes of operation (e.g., CTR, CBC-MAC), unpredictable random IV is not required.

14) Iteration Count in Password-based Encryption (PBE): The iteration count is one of the parameters in `javax.crypto.spec.PBEParameterSpec` API. In PKCS #5 [108], it is suggested that the number of iterations should be more than 1000 to provide a reasonable security level. In Fig. 3.4, Line 3 takes an iteration count of 20 which is insecure to be used in PBEParameterSpec.

15) Symmetric Ciphers: Symmetric ciphers use the same key for encryption and decryption. There are a couple of vulnerable symmetric cipher algorithms, e.g., DES, Blowfish, RC4, RC2, and IDEA. For example, DES is a broken block cipher because it uses an outdated block size (64 bits) that allows brute-force attack. RC4 is a flawed stream cipher that produces a biased keystream while a pseudo-random keystream is required for security, thus leading to several attacks (e.g., bit-flipping attack). To overcome the attacks, developers need to use a secure alternative AES which can support a block length of 128 bits and key lengths of 128, 192, and 256 bits [1]. Table 3.1 provides a list of insecure and secure symmetric ciphers. In Fig. 3.5, Line 1 is an insecure implementation of a symmetric cipher.

```
1 🐞 Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
2     cipher.init(Cipher.ENCRYPT_MODE, key)
```

Listing 3.5: Use case of `javax.crypto.Cipher` API is insecure if DES symmetric cipher and ECB mode of operation are used

16) Asymmetric Ciphers: In asymmetric cryptography, two keys, i.e., a public key and a private key are used for encryption and decryption. RSA is considered insecure for 1024-bit ciphers [54]. For this reason, developers are recommended to use RSA with a key size of 2048 bits or higher.

17) Cryptographic Hash Functions: A cryptographic hash function generates a fixed-length alphanumeric hash value or message digest which is commonly used in verifying

Table 3.1: Secure and insecure use of the mode of operation (12), symmetric cipher (15), cryptographic hash function (17)

Cryptograhic API	**Secure**	**Insecure**
Mode of Operation	CBC, GCM	ECB
Symmetric Cipher	AES	DES, Blowfish, RC4, RC2, IDEA
Hash Function	SHA-256	SHA1, MD5, MD4, MD2

message integrity, digital signature, and authentication. A cryptographic hash function is contemplated as broken if a collision can be observed, i.e., the same hash value is generated for two different inputs. The list of broken hash functions includes SHA1, MD4, MD5, and MD2. Therefore, developers need to use a strong hash function, e.g., SHA-256. Table 3.1 provides a list of insecure and secure hash functions. In Fig. 3.6, a code snippet of the broken hash function test case is shown.

```
🐞 MessageDigest md=MessageDigest.getInstance("MD5");
   md.update(message.getBytes());
```

Listing 3.6: Use case of java.security.MessageDigest API is insecure if MD5 is used. Hash function SHA-256 is secure

18) Cryptographic MAC: A MAC algorithm HmacMD5 and HmacSHA1 are considered insecure as these are susceptible to collision attacks [57]. Therefore, the developers need to use a strong MAC algorithm, e.g., HmacSHA256.

3.2.2 Java Cryptographic API Vulnerability Detection Tools

We summarize the vulnerability detection tools that we choose to run on CryptoAPI-Bench and ApacheCryptoAPI-Bench. We consider three criteria while choosing the analysis tools. (1) Open-sourced tools: The open-sourced vulnerability detection tools, i.e., CrySL [95, 141], CryptoGuard [122], SpotBugs [30] are convenient to use as we are able to analyze their codes and understand the reason for their lack of performance. (2) Static analysis tools: We choose static analysis tools that can examine and detect vulnerability without executing the code. SpotBugs, CryptoGuard, CrySL, and Tool A are static analysis tools. (3) Free cryptographic vulnerability detection services: We consider Tool A as a provider of free cryptographic vulnerability detection services. Tool A is not open-sourced. However, Tool A provides online services to detect vulnerabilities.

We also consider GrammaTech [16], QARK [23] and FixDroid [113]. However, GrammaTech is a commercial tool. We were unable to access its trial version. The online SWAMP [68] contains GrammaTech tool to use that only supports vulnerability detection for C and C++. Therefore, we excluded GrammaTech from our list of tools. QARK is a tool that is mainly designed to capture security vulnerabilities in Android applications. FixDroid is built as a research prototype that is embedded as a plugin in Android Studio to conduct a usability study. Our investigation shows that the detection capability of FixDroid and QARK is limited. Though QARK has been maintained and updated, FixDroid has not been updated since 2017. Therefore, we mainly focus on four tools, i.e., SpotBugs, CryptoGuard, CrySL, and Tool A to evaluate on CryptoAPI-Bench. We choose to anonymize Tool A's name. Tool A has an educational license that generally does not allow publishing comparison with other tools.

SpotBugs

SpotBugs is a static analysis tool also for capturing deficiencies in Java code. The tool is built based on a plugin structure. The tools detect defects by utilizing visitor patterns in class files or bytecodes of Java, state machine, and flags. We use the SpotBugs tool (version 3.1.12) available online in SWAMP [68]. However, currently, SWAMP is in the transition to a new host service [33].

CryptoGuard

CryptoGuard [122] is a static analysis tool that is operated based on program slicing with novel language-based refinement algorithms. It significantly reduces the false positive rate which is a typical problem for static analysis. Furthermore, CryptoGuard covers 16 cryptographic rules and achieves high precision. The authors showed screening a large number of Apache projects and Android apps to present their high precision rate and low false positive rate. We run the experiment on CryptoGuard (commitID: 97b220) available on GitHub [11].

CrySL

CrySL [95] is a domain-specific language for cryptographic libraries. The static analysis $CogniCrypt_{SAST}$ takes the rules provided in the specification language CrySL as input and performs a static analysis based on the specification of the rules. CrySL is open-sourced and we run the experiment on CrySL (commit ID: 004cd2) available on GitHub [12].

3.3 Design of Benchmarks

We developed two benchmarks, CryptoAPI-Bench and ApacheCryptoAPI-Bench. In this section, we explain how we design these benchmarks to help developers to identify vulnerabilities in their tools. We include 18 cryptographic misuse categories (discussed in the Background Section) in these benchmarks.

3.3.1 Design of CryptoAPI-Bench

In CryptoAPI-Bench, we manually generate 181 unit test cases guided by 18 types of misuses presented in Section 3.2.1. We divide all test cases into two types, i.e., basic cases and advanced cases. These test cases incorporate the majority of possible variations in the perspective of program analysis to detect cryptographic vulnerability.

Basic Cases

Basic test cases are simple ones where the probable source of vulnerability for Crypto API exists within the same method. For example, Listing 1 shows that Cipher API takes `cryptoAlgo` as an argument. Note that, `cryptoAlgo` contains an insecure cipher algorithm that is defined within the same method (`method1`). In CryptoAPI-Bench, we create 45 basic test cases covering all 18 misuse categories. Among these test cases, 30 test cases contain cryptographic vulnerability (i.e., true positive), and 15 test cases do not contain any cryptographic vulnerability (i.e., true negative). These test cases identify a tool's capability to detect a specific misuse category. An example code snippet of a basic test case is presented in Listing 3.7.

```
public void method1 ()
{   ...
    cryptoAlgo = "DES/ECB/PKCS5Padding"
    Cipher cipher = Cipher.getInstance(cryptoAlgo)
    ...
}
```

Listing 3.7: Example code snippet of a basic test case

Table 3.2: CryptoAPI-Bench: Summary of unit test cases. There are 181 unit test cases with 45 basic cases and 136 advanced cases (interprocedural, field sensitive, combined case, path sensitive, miscellaneous, and multiple class test cases). Total test cases per group and misuse categories are summarized here. Details information are presented in Section 3.3.1.

No.	**Misuse Categories**	**Basic Cases**	**Two-Interproc.**	**Three-Interproc.**	**Field Sen.**	**Comb. Case**	**Path Sen.**	**Misc.**	**Multi. Class**	**Total Cases per Categ.**
1	Cryptographic Key	2	1	1	1	1	1	1	1	**9**
2	Password in PBE	3	1	1	1	1	1	2	1	**11**
3	Password in KeyStore	2	1	1	1	1	1	2	1	**10**
4	Hostname Verifier	2	0	0	0	0	0	0	0	**2**
5	Certificate Validation	3	0	0	0	0	0	0	0	**3**
6	SSL Socket	1	0	0	0	0	0	0	0	**1**
7	HTTP Protocol	2	1	1	1	1	1	0	1	**8**
8	PRNG	2	0	0	0	0	0	0	0	**2**
9	Seed in PRNG	3	2	2	2	2	2	2	2	**17**
10	Salt in PBE	2	1	1	1	1	1	1	1	**9**
11	Mode of Operation	2	1	1	1	1	1	0	1	**8**
12	Initialization Vector	2	1	1	1	1	1	2	1	**10**
13	Iteration in PBE	2	1	1	1	1	1	1	1	**9**
14	Symmetric Ciphers	6	5	5	5	5	5	0	5	**36**
15	Asymmetric Ciphers	1	1	1	0	1	1	0	1	**6**
16	Cryptographic Hash	5	4	4	4	4	4	0	4	**29**
17	Cryptographic MAC	3	0	0	0	0	0	0	0	**3**
18	Credentials in String	2	1	1	1	1	0	1	1	**8**
Total Cases per Group		**45**	**21**	**21**	**20**	**21**	**20**	**12**	**21**	**181**

Advanced Cases

The advanced cases are more complex compared to basic cases where the probable source of vulnerability of a Crypto API appears from other methods, classes, field variables, or conditional statements. In CryptoAPI-Bench, we include 136 advanced cases. The distribution of advanced cases is presented from the fourth to tenth columns of Table 3.2.

Interprocedural Cases

In interprocedural cases, the probable source of vulnerability in a Crypto API comes from other methods (i.e., procedures). We create two types of interprocedural cases: two-interprocedural (i.e., involving two methods) and three-interprocedural (i.e., involving three methods). In a two-interprocedural test case, the probable source of vulnerability comes from another method as a parameter. Listing 3.8 shows the code snippet of a two-interprocedural test case. In `method2`, Cipher API takes `cryptoAlgo` as an argument, and `cryptoAlgo` is not defined in `method2`, rather, it comes from another method `method1`. The assigned value of `cryptoAlgo` in `method1` shows that the test case is insecure.

```java
public void method1 ()
{  ...
   cryptoAlgo = "DES/ECB/PKCS5Padding"
   method2(cryptoAlgo)
   ...
}
public void method2 (String cryptoAlgo)
{  ...
   Cipher cipher = Cipher.getInstance(cryptoAlgo)
   ...
}
```

Listing 3.8: Example code snippet of a two-interprocedural test case

In three-interprocedural test cases, the probable source of vulnerability comes from two consecutive methods (i.e., source defined in one method, passes to another method, and then passes again to be used in Cipher API). CryptoAPI-Bench contains a total of 42 interprocedural test cases. Among them, 21 are two-interprocedural test cases, and 21 are three-interprocedural test cases. The purpose of having the interprocedural test cases is to check the detection tool's interprocedural data flow handling capability.

Field Sensitive Cases

In field-sensitive cases, the probable source of cryptographic vulnerabilities can be detected by the analysis tools if the tools are capable of performing field-sensitive data flow analysis. Field-sensitive refers to an analysis that is able to differentiate multiple fields or variables with the same object [132]. In Listing 3.9, `algo` is an instance or field variable in the `Crypto` class. The constructor `Crypto()` stores `algo` with defAlgo object. A class member function `encrypt()` uses this `algo` value in Cipher API. Both `algo` and `defAlgo` contain the same object, i.e., a secure or insecure cipher algorithm. This is a field-sensitive case as the tools need to trace the field variable `algo` as the probable source of vulnerability. CryptoAPI-Bench contains 20 field-sensitive test cases.

```
class Crypto {
    String algo
    public Crypto (String defAlgo) {
        algo = defAlgo;
    }
    public void encrypt(... ) {
        ...
        Cipher cipher = Cipher.getInstance(algo);
        ...
    }
 }
```

Listing 3.9: Example code snippet of a field sensitive test case

Combined Cases

The combined cases are a bit more complex where both interprocedural and field sensitivity properties are combined, i.e., both Listing 3.8 and Listing 3.9 are incorporated to generate complicated test cases. CryptoAPI-Bench has 21 combined test cases.

Path-Sensitive Cases

In path-sensitive test cases, conditional branch instructions are included in the test cases containing the definition of the probable source of a vulnerability. In Listing 3.10, an example code snippet of a path sensitivity case is given. Depending on the `choice` variable, the `Cipher` is getting the instance from a secure or an insecure cryptographic algorithm. There are 20 path-sensitive test cases in CryptoAPI-Bench.

```java
public void method1 (int choice) {
    ...
    Cipher ch = Cipher.getInstance ("DES/ECB/...") ;
    if (choice > 1) {
        ch = Cipher.getInstance ("AES/CBC/...") ;
    }
    ch.init (Cipher.ENCRYPT_MODE, key) ;
    ...
 }
```

Listing 3.10: Example code snippet of a path sensitive test case

Miscellaneous Cases

Miscellaneous test cases evaluate the tool's abilities to recognize irrelevant constraints and other interfaces, e.g., Map. In Listing 3.11, the Map interface of Line 3-6 provides a secure key or insecure key depending on the choice variable. The Map indices (e.g., "a", "b") represent only index values, not security-relevant values. Similarly, in Line 8, the "UTF-8" represents byte encoding, not any constant or hard-coded value. CryptoAPI-Bench contains 12 miscellaneous test cases.

```java
public void method1 (String choice) {
    ...
    Map<String,String> hm = new HashMap<String, String>();
    hm.put( ``a", secureKeyString);
    hm.put( ``b", insecureKeyString);
    String keyString = hm.get(choice);

    byte [] b = secureKeyString.getBytes("UTF-8");
    IvParameterSpec ivSpec = new IvParameterSpec(b);
    ...
}
```

Listing 3.11: Example code snippet of a miscellaneous test case

Multiple Class Cases

In multiple class test cases, the probable source of vulnerabilities comes from another Java class. An example code snippet of a multiple class case is presented in Listing 3.12. It is necessary to detect whether a secure or an insecure algorithm is passed in Line 4 in `MultipleClass1` and used in Line 9 in `MultipleClass2`. CryptoAPI-Bench has 21 multiple-class test cases.

3.3.2 Design of ApacheCryptoAPI-Bench

We include the early version of real-world large 10 Apache projects to check the scalability property of different tools. The second and third columns of Table 3.3 show the number of Java files and lines of Java Code in Apache projects. The spark project is the largest among 10 considered projects containing 2,005 Java files with 311,856 lines of code. The meecrowave project contains the lowest number of Java files (40 Java files) and deltaspike contains the lowest number of lines of code (i.e., 5,116 LoC).

```java
public class MultipleClass1 {
    public void method1 (String passedAlgo) {
        MultipleClass2 mc = new MultipleClass2 ();
        mc.method2 (passedAlgo);
    }
}
public class MultipleClass2 {
    public void method2 (String cryptoAlgo) {
        Cipher c = Cipher.getInstance (cryptoAlgo);
    }
}
```

Listing 3.12: Example code snippet of a multiple class test case

We enlist 121 test cases in our ApacheCryptoAPI-Bench [41]. Among them, 82 test cases are basic cases, i.e., the vulnerability rise within the same method. There are 39 advanced test cases where probable source vulnerability comes from other methods (interprocedural cases), other classes (multiple class cases), class variables (field sensitive cases), etc. We detect 64 cryptographic misuses, i.e., true positive alerts. Regarding true negatives, we consider only the cases where a tool shows the case as a false alert. With this consideration, we show 57 true negative cases.

We look into the Apache projects in the Benchmark and made detailed documentation. The documentation consists of cryptographic vulnerabilities the project contains, an explanation of the error, and the location (file name, method name, line number) of the vulnerabilities. The documentation and corresponding ApacheCryptoAPI-Bench benchmark are available in the GitHub repository [38].

Table 3.3: ApacheCryptoAPI-Bench: Summary of unit test cases. Contents (number of Java files and lines of code) of the considered Apache projects are summarized here. There is a total number of 121 unit test cases with 82 basic cases and 39 advanced cases. Details information are presented in Section 3.3.2.

Apache Project	**Number of Java Files**	**Lines of Code**	**Test Cases**				
			Total Case	**Basic Case**	**Advanced Cases**	**TP**	**TN**
deltaspike	87	5116	8	5	3	2	6
directory-server	468	20780	36	15	21	19	17
incubator-taverna-workbench	45	9919	8	5	3	8	0
manifoldcf	126	16998	7	4	3	3	4
meecrowave	40	5646	3	3	0	3	0
spark	2005	311856	26	25	1	12	14
tika	225	16558	2	1	1	0	2
tomee	1029	118661	9	7	2	7	2
wicket	204	13442	9	7	2	7	2
artemis-commons	126	8915	15	12	3	7	8
		Total	**121**	**82**	**39**	**64**	**57**

3.4 Evaluation and Findings

In this section, we evaluate the results for four cryptographic misuse detection tools, i.e., SpotBugs, CryptoGuard, CrySL, and Tool A. We show the experimental setup, evaluation criteria, and analysis results of both CryptoAPI-Bench and ApacheCryptoAPI-Bench.

3.4.1 Experimental Setup

We evaluate mainly four cryptographic analysis tools, i.e., SpotBugs [30], CryptoGuard [122], CrySL [95], Tool A on both Benchmarks. We follow the instructions from GitHub to set up the environment of CryptoGuard and CrySL in our machine to perform the analysis. We upload JAR files from CryptoAPI-Bench and Apache projects into the SpotBugs tool available in SWAMP. Tool A is an online tool that takes GitHub links and compressed code files in order to start analysis.

Table 3.4: Generated alert keywords for each misuse category from cryptographic vulnerability detection tools (SpotBugs, CryptoGuard, CrySL, and Tool A). For example, for misuse category 17 (i.e., Cryptographic Hash), the generated alert keywords in tools are WEAK_MESSAGE_DIGEST, broken hash scheme, ConstraintError, RISKY_CRYPTO, respectively.

No.	SpotBugs	CryptoGuard	CrySL	Tool A
1	HARD_CODE_PASSWORD	Constant keys	RequiredPredicateError	HARDCODED_CREDENTIALS
2	HARD_CODE_PASSWORD	Constant keys	HardCodedError	HARDCODED_CREDENTIALS
3	HARD_CODE_PASSWORD	Predictable password	HardCodedError	HARDCODED_CREDENTIALS
4	—	—	RequiredPredicateError	—
5	WEAK_HOSTNAME_VERIFIER	Manually verify hostname	—	BAD_CERT_VERIFICATION
6	WEAK_TRUST_MANAGER	Untrusted TrustManager	—	BAD_CERT_VERIFICATION
7	–	Does not manually verify socket	—	RESOURCE_LEAK
8	—	HTTP protocol	—	—
9	PREDICTABLE_RANDOM	Untrusted PRNG	—	—
10	—	Predictable Seed	RequiredPredicateError	PREDICTABLE_RANDOM_SEED
11	—	Constant Salt	RequiredPredicateError	—
12	CIPHER_INTEGRITY	Broken crypto scheme	ConstraintError	RISKY_CRYPTO
13	STATIC_IV	Constant IV	RequiredPredicateError	—
14	—	<1000 iteration	ConstraintError	—
15	CIPHER_INTEGRITY	Broken crypto scheme	ConstraintError	RISKY_CRYPTO
16	—	Export grade public key	ConstraintError	—
17	WEAK_MESSAGE_DIGEST	Broken hash scheme	ConstraintError	RISKY_CRYPTO
18	—	—	ConstraintError	—

3.4.2 Evaluation Criteria

We evaluate the vulnerability detection tools by running these tools on our benchmarks. After performing the analysis, we capture true positives, false positives, and false negatives from the corresponding tool's result log. As our purpose is to detect cryptographic vulnerability detection, we consider only cryptographic misuse alerts and discard others. In Table 3.4, we present the alert keywords that detection tools use while showing a specific cryptographic misuse. This can assist developers to understand which keyword they should search in the result log to find a specific type of vulnerability. In the following, we provide a brief description of our process of identification of true positive, false positive, and false negative alerts.

- **True positive (TP)**: If a tool generates an alert due to the correct reason while screening any specific vulnerable unit test case in CryptoAPI-Bench, then the event is considered a true positive.

- **False positive (FP)**: The false positive alert can be captured from two different scenarios. If an alert raised by a tool is unexpected (i.e., does not exist in a specific unit test case), then the alert is a false positive. In addition, if a tool gives an inaccurate reason for an expected alert, then it is also considered a false positive.

- **False negative (FN)**: A vulnerable test case may not be detected by the evaluation tools. This missed detection is characterized as a false negative.

After analyzing the results by determining the true positive (TP), false positive (FP), and false negative (FN) values, we compute the recall and precision to determine the performance of the tools.

3.4.3 Evaluation on CryptoAPI-Bench

In this section, we describe CryptoAPI-Bench evaluation findings on each detection tool based on the result log and performance analysis. Table 3.5 presents the number of true positive and false positive vulnerability threat detection captured by the tools for 18 cryptographic misuse categories. There are only 6 common cryptographic misuse categories detected by all tools. To ensure fairness in comparison, we consider only these 6 common cryptographic misuses while finding the comparative analysis results of tools based on the basic and advanced benchmark in Table 3.6 and Table 3.7, respectively. The analysis results are presented in terms of false positive rate (FPR), false negative rate (FNR), recall, and precision.

Analysis Overview: Table 3.5 shows that among the 18 specified high-impact cryptographic misuse categories in Section 3.2.1, the cryptographic vulnerability detection tools are able to detect a subset of rules.

- SpotBugs, CryptoGuard, CrySL, and Tool A cover 9, 16, 14, and 10 cryptographic misuse categories, respectively.

Table 3.5: CryptoAPI-Bench comparison of SpotBugs, CryptoGuard, CrySL, and Tool A on all 18 rules with CryptoAPI-Bench's 181 test cases. There are 37 secure API use cases (15 in basic and 22 in advanced), which a tool should not raise any alerts on. GTP stands for ground truth positive, which is the number of insecure API use cases in the benchmark. Findings of the table are reported in Section 3.4.3.

No.	Misuse Categories	GTP	SpotBugs		CryptoGuard		CrySL		Tool A	
			TP	**FP**	**TP**	**FP**	**TP**	**FP**	**TP**	**FP**
1	Cryptographic Key	7	0	3	5	1	0	8	5	1
2	Password in PBE	8	2	0	7	1	0	10	7	1
3	Password in KeyStore	7	1	1	7	1	0	10	5	1
4	Credentials in String	7	–	–	–	–	0	8	–	–
5	Hostname Verifier	1	–	–	1	0	–	–	1	0
6	Certificate Validation	3	3	0	3	0	–	–	3	0
7	SSL Socket	1	–	–	1	0	–	–	1	0
8	HTTP Protocol	6	–	–	6	1	–	–	–	–
9	PRNG	1	1	0	1	0	–	–	–	–
10	Seed in PRNG	14	–	–	11	2	0	15	1	2
11	Salt in PBE	7	–	–	6	1	6	1	–	–
12	Mode of Operation	6	1	3	6	1	5	1	1	1
13	Initialization Vector	8	3	6	7	1	7	1	–	–
14	Iteration Count in PBE	7	–	–	5	1	5	3	–	–
15	Symmetric Cipher	30	5	11	30	5	25	5	4	4
16	Asymmetric Ciphers	5	–	–	4	1	5	1	–	–
17	Cryptographic Hash	24	4	8	24	4	20	4	4	4
18	MAC Algorithm	2	–	–	–	–	2	0	–	–
Total		**144**	**20**	**32**	**124**	**20**	**75**	**67**	**32**	**14**

- In total, the benchmark contains 144 vulnerable test cases and among these true positive cases, SpotBugs, CryptoGuard, CrySL, and Tool A detect 20, 124, 75, and 32 cases, respectively.

- In addition, SpotBugs, CryptoGuard, CrySL, and Tool A also generate 32, 20, 67, and 14 false alarms, respectively that are included as false positive cases.

Analysis on Basic Benchmark

Table 3.6 shows the performance analysis result of four detection tools on six common cryptographic misuse categories based on the basic benchmark. We capture the following findings:

- SpotBugs shows 3 false positive errors. It detects all cases except one. SpotBugs is not

Table 3.6: CryptoAPI-Bench comparison of SpotBugs, CryptoGuard, CrySL, and Tool A on six common misuse categories with CryptoAPI-Bench's common 21 basic cases. TP, FP, and FN stand for true positive, false positive, and false negative, respectively. Findings of the table are reported in Section 3.4.3.

Basic Test Cases	**TP Count**	**TN Count**	**SpotBugs**			**CryptoGuard**			**CrySL**			**Tool A**		
			TP	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
IntraProcedural	14	6	13	3	1	14	0	0	10	7	4	13	0	1
Result	**Recall (%)**		**92.86**			**100.00**			**71.43**			**92.86**		
	Precision (%)		**81.25**			**100.00**			**58.82**			**100.00**		

designed to capture threats in the basic case of vulnerable cryptographic key misuse.

- CrySL produces 7 false positive errors due to maintaining strict rules in Crypto APIs of the cryptographic key, password in PBE, and password in KeyStore.

- Tool A does not generate any false positive errors. It can successfully detect every vulnerability except one. Tool A is not designed to capture IDEA as a vulnerable cryptographic algorithm.

- For insecure uses of pseudo-random number generators, SpotBugs and CryptoGuard flag all uses of java.util.Random. However, CrySL flags the insecure random variable when used in crypto contexts.

In summary, for all basic cases, CryptoGuard and Tool A generate a precision of 100%. SpotBugs and CrySL produce some false positives and hence generate a precision of 81.25%, and 58.82% respectively.

Analysis on Advanced Benchmark

Table 3.7 shows the performance analysis result of four detection tools on six common cryptographic misuse categories based on the advanced benchmark. We capture the following findings:

- In the prospect of path sensitivity, it is obvious that none of the cryptographic vulnerability detection tools is path-sensitive in their static analysis. The tools generate 10, 13, 13, and 12 false positive alerts for path-sensitive cases, respectively. The possible reason for the false positive alert is that for the concerned variable, a container is defined to store all values of the concerned variable. There is no ordered list that shows the latest assignment. Therefore, alerts will be raised if the container contains any vulnerable value that is intended to be used in the Crypto API. A significant reason for having a high false positive rate is because of the tools being path insensitive.

- SpotBugs is not designed to capture vulnerability threats in advanced cases. Therefore, it shows 0% precision and recall.

- SpotBugs produces 12 false positives for combined cases. In combined cases, SpotBugs failed to detect the source of vulnerability using both interprocedural and field-sensitive analysis. For example, in Symmetric Cipher cases, instead of showing the correct "CIPHER_INTEGRITY" alert, it produces an incorrect "HARD_CODE_PASSWORD" alert.

- CryptoGuard performs better than other tools in terms of both precision and recall. The reasons behind this include 1) Cryptoguard performs dataflow analysis based on forward slicing and backward slicing that efficiently handles the advanced cases, 2) CryptoGuard follows several refinement insights that systematically remove irrelevant constants, hence reducing false positives. However, as being a static analysis tool, CryptoGuard cannot handle path-sensitive cases. In addition, CryptoGuard missed 3 vulnerabilities due to clipping orthogonal method invocation (i.e., limiting the depth to visit the callee method).

- CrySL produces incorrect "RequiredPredicateError" alerts for the cryptographic key,

Table 3.7: CryptoAPI-Bench comparison of SpotBugs, CryptoGuard, CrySL, and Tool A on six common misuse categories with CryptoAPI-Bench's common 84 advanced cases. TP, FP, and FN stand for true positive, false positive, and false negative, respectively. Findings of the table are reported in Section 3.4.3.

Advanced Test Cases	**TP Count**	**TN Count**	**SpotBugs**			**CryptoGuard**			**CrySL**			**Tool A**		
			TP	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
Two-Interprocedural	13	0	0	0	13	12	0	1	10	3	3	3	0	10
Three-Interprocedural	13	0	0	0	13	12	0	1	10	3	3	3	0	10
Field Sensitive	13	0	0	0	13	13	0	0	10	2	3	1	0	12
Combined Case	13	0	0	12	13	12	0	1	0	2	13	3	0	10
Path Sensitive	0	13	0	10	0	0	13	0	0	13	0	0	12	0
Miscellaneous Cases	3	2	0	0	3	3	0	0	0	5	3	0	0	3
Multiple Class methods	13	0	0	0	13	13	0	0	10	3	3	3	0	10
Results	**Recall (%)**		**0.00**			**95.59**			**58.82**			**19.12**		
	Precision (%)		**0.00**			**83.33**			**56.34**			**52.00**		

password in PBE, and password in KeyStore misuse test cases that contribute to generating a high false positive rate. The reason is that the cryptographic APIs used in these cases follow strict rules in CrySL. Therefore, even if we use a secure unpredictable byte array as an argument for crypto APIs, it still generates incorrect alerts.

- Tool A is not designed to detect vulnerable ciphers and cryptographic hash functions in advanced cases. That is the reason for having high false negative values and generating high FNR in Tool A. Tool A is a close-sourced detection tool. Therefore, we are unable to confirm the reason for the incorrect detection cases.

In summary, for all of the advanced cases, SpotBugs is not designed to identify the advanced vulnerability threats correctly. Therefore, the precision rate is 0%. CryptoGuard detects fairly well (missed only 3 cases) among all detection tools with a precision of 83.33%. CrySL produces a precision of 56.34%. Tool A generates a precision of 52.00%.

3.4.4 Evaluation on ApacheCryptoAPI-Bench

Table 3.8 presents the number of true positive and false positive vulnerability threats detected by the tools. CrySL fails to analyze spark and artemis-commons projects. Tool A fails to analyze artemis-commons project. SpotBugs and CryptoGuard successfully analyze all 10 projects. Overall, we capture the following findings.

- Tool A has a low false positive value. However, SpotBugs and CryptoGuard have high false positive values of 26, and 29, respectively. The main reason is that CryptoGuard and SpotBugs consider all usages of Java.util.Random as vulnerable whereas the majority of the random is used in a non-security context. We have discussed the reason for generating high false positives for CrySL in Section 6.3.2.

- SpotBugs, CryptoGuard, CrySL, and Tool A can accurately detect 35, 37, 40, and 21 alerts respectively from 64 alerts. The main reason for missed alarms is that no tool can detect all 18 types of vulnerabilities as shown in Table 3.4. For example, SpotBugs and CryptoGuard cannot capture vulnerable crypto algorithm usage in SecretKeySpec API. Among the successfully compiled programs (i.e., from 8 Apache projects), CrySL captures 40 out of 45.

- After analyzing ten Apache projects, we find that there are 82 basic cases, whereas, the number of advanced cases is only 39. Therefore, in real-world codes, the number of basic cases is much higher than advanced cases. Vulnerability detection tools should consider expanding their coverage to detect more categories of vulnerabilities.

- From Table 3.8, we observe that CrySL fails to analyze two Apache projects: spark and artemis-commons. CrySL throws StackOverFlowError (i.e., memory error) during analyzing objects for spark. The probable reason is the larger number of files and lines of code Spark contains for analysis. For artemis-commons, CrySL throws NullPointerErrorException during analysis due to the reference variable not pointing to any object. Tool A fails to analyze only the artemis-commons project. Tool A is a closed-source tool, therefore, we are unable to confirm the reason for this failure.

Table 3.8: ApacheCryptoAPI-Bench comparison of SpotBugs, CryptoGuard, CrySL, and Tool A on 10 Apache projects. GTP stands for ground truth positive, which is the number of insecure API use cases in the Apache codes.

		SpotBugs			CryptoGuard			CrySL			Tool A		
Apache Project	**GTP**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
deltaspike	2	2	0	0	2	0	0	2	3	0	2	0	0
directory-server	19	11	0	8	5	0	14	18	6	1	5	0	14
incubator-traverna-workbench	8	2	0	6	4	0	4	7	0	1	3	0	5
manifoldcf	3	0	3	3	0	3	3	3	2	0	2	1	1
meecrowave	3	3	0	0	2	0	1	2	0	1	2	0	1
spark	12	9	12	3	12	14	0	–	–	–	4	0	8
tika	0	0	0	0	0	0	0	0	2	0	0	0	0
tomee	7	3	1	4	4	2	3	6	0	1	3	1	4
wicket	3	0	2	3	3	2	0	2	2	1	0	0	3
artemis-commons	7	5	8	2	5	8	2	–	–	–	–	–	–
Total	**64**	**35**	**26**	**29**	**37**	**29**	**27**	**40**	**15**	**5**	**21**	**2**	**36**

Table 3.9 shows the runtime on Apache projects for only CryptoGuard and CrySL. For Tool A and SpotBugs, we use the web version that takes all scan requests for users and reports results after complete scanning. Therefore, we cannot calculate their original runtime for comparison. Among the 8 successfully analyzed projects, we observe average runtime for CrySL is 14.64 seconds and CryptoGuard is 11.46 seconds. For the largest Apache project Spark (LoC: 311,856), CryptoGuard successfully analyzes in 88.68 seconds and CrySL produces an incomplete analysis report after running for 46.84 seconds. Overall, SpotBugs and CryptoGuard successfully analyze all 10 Apache projects. Therefore, SpotBugs and CryptoGuard are scalable for large projects.

Table 3.9: Runtime for analyzing Apache projects. Star (*) symbol indicates that the analysis was unsuccessful.

		Runtime (sec)	
Apache Projects	**LoC**	**CryptoGuard**	**CrySL**
deltaspike	5.1K	4.31	6.95
directory-server	20.8K	8.96	23.03
incubator-taverna-workbench	9.9K	12.69	7.94
manifoldcf	17K	7.07	8.20
meecrowave	5.6K	4.67	7.24
spark	311.9K	88.68	46.84*
tika	16.6K	7.46	8.15
tomee	118.7K	40.52	34.81
wicket	13.4K	5.99	20.83
artemis-commons	8.9K	5.63	19.82*

3.4.5 Verifiability

Our benchmarks are open-sourced and are available on GitHub [37, 38]. It contains the Java cryptographic API test cases. Detailed documentation and explanation are provided there.

3.5 Discussion

In this section, we discuss the insights of the tools, case studies, and limitations of our developed benchmarks.

3.5.1 Tool insights

No tool can cover all categories of vulnerabilities (Table 3.5). However, their methodologies can be extended to cover most of these vulnerabilities. For example, the technique that Tool A uses to detect constant cryptographic keys can be transferred to detect static IVs or fewer iteration counts.

The main differences among different tools are within their approach to trade-offs between false positives and false negatives. Our experimental evaluation reveals that all of these tools produce a number of false positives and false negatives. CryptoGuard performs on-demand inter-procedural dataflow analysis. Its backward data flow analysis starts from the slicing criteria and explores upward ($\uparrow$) and orthogonally ($\rightarrow$) on-demand. Orthogonal method invocation chains always return to the call sites. By leveraging this insight, CryptoGuard offers a performance vs scalability tradeoff by limiting the depth of the orthogonal invocations (which is "clipping of orthogonal method invocations"). In the current implementation, the depth is set to 1. That means CryptoGuard will skip deeper orthogonal callee methods, which may result in false negatives. However, the advantage of the orthogonal method invocation technique is that it helps to improve precision.

The main focus of CrySL is to provide a language to specify a class of cryptographic misuse vulnerabilities that can be detected using a generic detection engine. For the version that we tested, CrySL would raise an alert if a cryptographic key is not generated using a key generator. However, one can legitimately reuse a previously generated key, which CrySL would mistakenly detect as a vulnerability. An impressive aspect of CrySL is that it is constantly being maintained and updated to improve its accuracy. The methodology of SpotBugs is inherently limited to detecting advanced cases as they use patterns to detect most of the vulnerabilities.

None of these tools are path-sensitive, i.e., all raise false alerts in path-sensitive cases. A possible reason for this failure is that the existing path-sensitive analysis techniques are usually costly, i.e., high runtime complexity.

CryptoAPI-Bench cannot be used to evaluate scalability property. All of our test cases are lightweight by design. The primary focus is to produce easily readable test cases that demand minimal code to express complex program properties. On the other hand, all of the projects in are complex programs including a lot of files and lines of code. The primary focus is to test the vulnerability detection tool's scalability property and extrapolation to applications on real-world code.

3.5.2 Case Studies

Table 3.5 shows that many misuse cases are still uncovered by tools (e.g., CryptoGuard, SpotBugs, Tool A cannot handle MAC misuses) that should be addressed to expand coverage. Among the covered rules, there are also some deficiencies. For example, CryptoGuard and SpotBugs can capture RC4 as a vulnerable cipher but not ARCFOUR cipher algorithm as the static code does not specify ARCFOUR as a vulnerable cipher. Static or predictable initialization vector defined in another method, class, file, or field variable (i.e., advanced cases) cannot be captured using SpotBugs and Tool A. In another advanced case, a java file in manifoldcf contains `SecretKeySpec key = new SecretKeySpec(secretKey.getEncoded(), "AES")`. This secretKey parameter is initialized as a field variable with a static string value of "NowIsTheTime" and passed through three procedures. This complex case cannot be captured by SpotBugs, CryptoGuard, or Tool A.

3.5.3 Limitation of Benchmarks

Currently, our benchmark does not contain cryptographic cases, e.g., digital signature, CBC-MAC misuses in MAC, or other modes of operations (e.g., CTR). We plan to include test cases based on these cryptographic vulnerabilities in our CryptoAPI-Bench benchmark. Furthermore, our benchmark does not have any cases that involve Java reflection APIs. The primary reason is that the use of Java reflection during cryptographic coding is highly unlikely. Consequently, none of the existing open-sourced tools is designed to detect such cases. However, we plan to include new cases that leverage Java reflection APIs to induce cryptographic misuse vulnerabilities.

3.6 Summary

We believe that for scientific, in-depth, and reproducible comparisons benchmarking is an important component. In this chapter, we present CryptoAPI-Bench and ApacheCryptoAPI-Bench to evaluate the detection accuracy, scalability, and security guarantees of various cryptographic misuse detection tools. Our benchmarks are open-sourced and are available on

GitHub. We evaluated four static analysis tools that are capable of detecting cryptographic misuses. Our evaluation revealed some interesting insights, i.e., *i)* tools that are specialized to detect cryptographic misuses (e.g., CryptoGuard, CrySL) cover more rules and higher recall than general-purpose tools (e.g., SpotBugs, Tool A), *ii)* none of the existing tools is path-sensitive.

Chapter 4

Prediction Bias Correction for Underrepresented Patients

In this chapter, we present our developed bias correction technique for the underrepresented population subgroups and the advantage of our developed sampling technique over other existing sampling techniques.

4.1 Introduction

Researchers have trained machine learning models to predict many diseases and conditions, including Alzheimer's disease [116], heart disease [102], risk of developing diabetic retinopathy [61], cancer risk [135] and survivability [84], genetic testing for diseases [134], hypertrophic cardiomyopathy diagnosis [50], psychosis [123], PTSD [77], and COVID–19 [120]. Neural network-powered automatic image analysis has also been shown useful for fast disease detection, e.g., breast cancer [56] and lung cancer [110]. A study showed that deep learning algorithms diagnose breast cancer more accurately (AUC=0.994) than 11 pathologists [56]. Hospitals (e.g., Cleveland Clinic's partnership with Microsoft [18], John Hopkins hospital partnership with GE) [17] are reported to use predictive analytics for monitoring patients' health status and preventing emergencies [15, 51, 81, 88].

However, clinical datasets are intrinsically imbalanced due to the naturally occurring frequencies of data [89]. The data is not evenly distributed across prediction classes (e.g., disease class vs. healthy class), race, age, or other subgroups. Data imbalance is a major cause of biased prediction results [89]. Biased prediction results may have serious consequences for some patients. For example, a recent study showed that automatic enrollment of high–risk patients into the health program favors white patients, although black patients had 26.3% more chronic health conditions than equally ranked white patients [114]. Similarly, algorithmic osteoarthritis pain prediction shows 43% racial disparities [119]. The design of widely used case-control studies are shown to have temporal bias reducing predictive accuracy [142]. For non–medical applications, researchers also identified serious biases in high–profile machine learning applications, e.g., a widely deployed recidivism prediction tool [9, 22, 69], online advertisement system [133], Amazon's recruiting engine [24], and face recognition system [63]. The lack of external validation and overclaiming causal effect in machine learning also raise concerns [140].

We present two categories of contributions to machine learning prognosis for underrepresented patients. One contribution is empirical evidence showing severe racial and age prediction disparities and the deceptive nature of common metrics. Another contribution is in evaluating the bias-correction ability of sampling methods, including a new double prioritized (DP) bias correction technique.

In our first contribution, we use two large medical datasets (MIMIC III and SEER) to show multiple types of prediction disparities, including the metric disparity. Poor prediction performance in minority samples is not reflected in widely used metrics. For imbalanced datasets, conventional metrics such as overall accuracy and AUC–ROC are largely influenced by the performance of the majority of samples, which machine learning models aim to fit. Unfortunately, this serious deficiency is not well discussed or reported by medical literature. For example, a study showed 66.7% of the 33 medical-related machine learning papers used AUC–ROC to evaluate models trained on imbalanced datasets [127]. We report racial, age, and metric disparities in machine learning models trained on clinical prediction benchmark [81] on MIMIC III and cancer survival prediction [84] on SEER cancer dataset. Both training datasets are imbalanced, in terms of gender, race, or age distribution. For example, for the in-hospital mortality (IHM) prediction with MIMIC III, 70.6% of data represents White patients, whereas only 9.6% represents Black patients. MIMIC III and SEER also have data imbalance problems among the two class labels (e.g., death vs. survival). For the IHM prediction, only 13.5% of the data belongs to the patient who died in the hospital. These data imbalances result in serious prediction biases. A typical neural network-based machine learning model14 that we tested correctly predicts 87.6% of non-death cases, but only 60.9% of death cases. Meanwhile, overall accuracy (computed over all patients) is relatively high (0.85), and AUC–ROC is 0.86, as a result of the good performance in the majority class. These high scores are misleading. Our study also reveals that accuracy disparity among age or race subgroups can be severe. For example, the mortality prediction precision (i.e., the fraction of actual deaths among predicted deaths) of young patients under 30 is 0.09, substantially lower than the whole population (0.40). Recognizing these accuracy disparities will help advance AI-based technologies to better serve underrepresented patients.

In our second contribution, we present a new technique, double prioritized (DP) bias correction, that aims to improve the prediction accuracy of specific demographic groups through sample enrichment. DP trains customized prediction models for specific subpopulations, a departure from the existing one-model-predicts-all-demographics paradigm. DP prioritizes specific underrepresented groups, as opposed to sampling across the entire patient population. Our results show that DP is effective in reducing disparity among age and race groups. For the in-hospital mortality (IHM) and 5-year breast cancer survivability (BCS) predictions, DP shows 8.6% to 23.8% improvement over the original model and 5.6% to 86.8%

improvement over eight existing sampling techniques, in terms of minority class recall. Our cross-race and cross-age-group results also suggest the need for training specialized machine learning models for different demographic subgroups. All sampling techniques (including DP) are not designed to address biases caused by under diagnosis, measurement, or any other sources of disparity besides data representation. In what follows, DP assumes that the noise is the same across all demographic subgroups and that the only source of bias that it aims to correct is representational.

4.2 Background

In this section, we explain several existing data imbalance correction methods, clinical datasets, and prediction tasks we studied.

4.2.1 Sampling Techniques

A widely used bias-correction approach to the data imbalance problem is sampling. We briefly describe several existing undersampling and oversampling techniques.

Undersampling

Random Undersampling (RUS) balances a dataset by randomly selecting samples of the majority class [137]. Several K–nearest neighbor (K–NN) classifier–based undersampling techniques [103] (e.g., NearMiss1, NearMiss3, Distant) exist. Nearmiss1 balances a dataset by selecting the majority class samples whose average distance to the three closest minority class samples are smallest. NearMiss3 balances a dataset by selecting the majority class samples whose distance is closest to each minority class sample. The Distant method balances a dataset by selecting the majority class samples whose average distance to the three closest minority class samples is the farthest.

Oversampling

Replicated oversampling balances a dataset by replicating samples of the minority class. State-of-the-art solutions are all oversampling methods, including Synthetic Minority Oversampling Technique (SMOTE) [65], Adaptive Synthetic Sampling (ADASYN) [83], and Gamma [91]. SMOTE generates new minority points between existing neighboring minority samples by linear interpolation. ADASYN generates new minority points between existing neighboring minority samples with more emphasis on the class border. The Gamma technique generates new minority points between existing neighboring minority samples using Gamma distribution. Another approach is augmenting synthetic minority class data using generative adversarial network (GAN) [80].

Stratified Sampling

Stratified random sampling [117] involves dividing the population (e.g., race, age, survived patient) into groups called strata. Random samples (i.e., patients) are then selected from each stratum so that selected samples are balanced or maintain the demographic ratio. The difference between simple random sampling and stratified sampling is that simple random sampling treats all members to have an equal likelihood of being sampled whereas stratified sampling samples are selected among groups or strata rather than the whole population.

4.2.2 Model Reweighting

Reweighting is an alternative bias correction approach to sampling [45, 106]. The reweighting approach assigns different importance to samples in the training data, in order for some minority class samples to impact more on training outcomes. Standard reweighting aims to make the weights of the two prediction classes balanced. In the standard reweighting approach, new weights are applied to the entire class population as follows. Reweight all samples so that each majority sample weights less than 1 and each minority sample weights more than 1, while satisfying the constraint that the total weight of each prediction class is equal.

4.2.3 Constraint in Objective Function

The seldonian algorithm [136] is designed to prevent undesirable behavior of machine learning models. It adds constraints to the objective function so that the prediction error can be bouned within a certain threshold among subgroups (for example, male and female). Also, a safety test is included in the seldonian algorithm to check whether the model is confident that for the applied constraints, it returns a solution.

4.3 Methodology

In this section, we present the double prioritized (DP) bias correction methodology and several other evaluation methodologies.

4.3.1 Double Prioritized (DP) Bias Correction Method

DP prioritizes a specific demographic subgroup (e.g., Black patients) that suffers from data imbalance by replicating minority prediction class (C1) cases from this group (e.g., Black in-hospital deaths). DP incrementally increases the number of duplicated units and chooses the optimal unit number based on the resulting models' performance. Figure 4.1 shows the machine learning workflow with DP bias correction. The main steps are described next. Sample enrichment replicates minority class C1 samples in the training dataset for a target demographic group g up to n times. Each time, duplicated samples are merged with the original training dataset, which forms a new training dataset. Thus, we obtain n+1 sets of training datasets, including the original one. Our experiment sets n to 19. The value n can be empirically determined based on prediction performance.

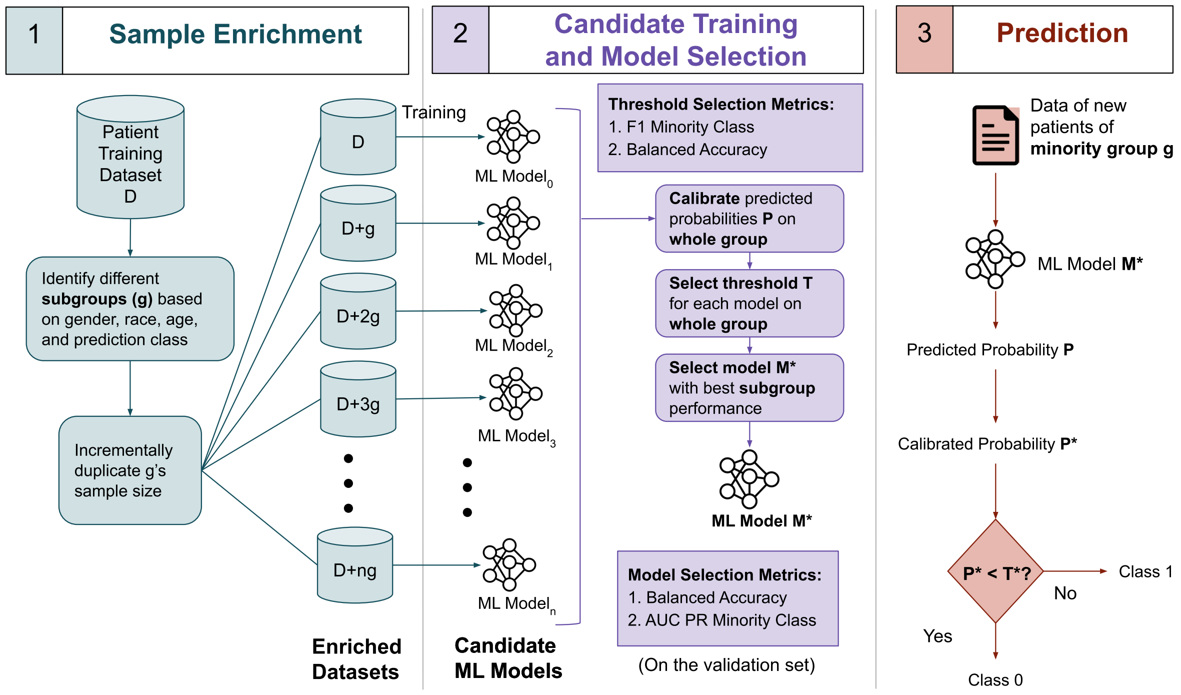

1 Sample Enrichment
2 Candidate Training and Model Selection
3 Prediction
Patient Training Dataset D
Identify different subgroups (g) based on gender, race, age, and prediction class
Incrementally duplicate g's sample size
D
D+g
D+2g
D+3g
D+ng
Training
ML Model_0
ML Model_1
ML Model_2
ML Model_3
ML Model_n
Enriched Datasets
Candidate ML Models
Threshold Selection Metrics:
1. F1 Minority Class
2. Balanced Accuracy
Calibrate predicted probabilities P on whole group
Select threshold T for each model on whole group
Select model M* with best subgroup performance
ML Model M*
Model Selection Metrics:
1. Balanced Accuracy
2. AUC PR Minority Class
(On the validation set)
Data of new patients of minority group g
ML Model M*
Predicted Probability P
Calibrated Probability P*
P* < T*?
No
Class 1
Yes
Class 0

Figure 4.1: Workflow for improving data balance in machine learning prognosis prediction using double prioritized (DP) bias correction.

Candidate training is to generate a set of candidate machine learning models. Each of the $n+1$ datasets is used to train and generate a candidate machine learning model.

Model selection is to identify the optimal machine learning model among the $n+1$ candidate models. We choose a final machine learning model M^* after evaluating all candidate models' performance as follows. For each model, we first calibrate the predicted probabilities on the validation set. Calibration is to adjust the distribution of probabilities before mapping probabilities into labels. We calibrate the output probabilities using the Isotonic Regression technique. We then perform threshold tuning to find the optimal threshold based on balanced accuracy and the F1_C1 score. Specifically, we first identify the top three thresholds that give the highest F1_C1 scores and then further select the optimal threshold that gives the highest balanced accuracy for the entire samples. For some subgroups, there are only a couple of hundreds of samples in the validation set. Selecting the threshold based on subgroup data may cause overfitting to the validation set. Therefore, we choose thresholds based on the whole group's performances. Given a threshold, we then identify the top three machine learning models with the highest balanced accuracy (i.e., average recall of both C0 and C1 classes, Equation A.6) values and select the model that gives the highest PR_C1 (the area under the curve (AUC) of minority class C1's precision-recall curve, denoted by AUC-PR_C1 or PR_C1) for demographic group g. In this step, no enrichment is applied to the validation dataset. When deciding thresholds, AUC-PR cannot be used, as it is a threshold-free metric. Thus, we use balanced accuracy and F1_C1.

Prediction applies model M^* to new patients' records of minority group g' and obtains a binary class label. At deployment, the demographic group g of duplicated samples during Sample enrichment and test group g' should be the same, e.g., the DP model trained with duplicated Black samples is used to predict new Black patients. Evaluation metrics include accuracy, balanced accuracy, AUC–ROC score, precision, recall, AUC–PR, and F1 score of minority and majority prediction classes, the whole population, and various demographic subgroups, including gender (male, female), race (White, Black, Hispanic, Asian), and 8 age groups. Minority class C1 precision and C1 recall are the two most used metrics in our work. C1 precision calculates the fraction of actual minority C1 class cases among predicted ones. C1 recall calculates the fraction of C1 cases that are predicted by a machine learning model. We use the relative disparity metric to capture the disparity among race groups or age groups. Equation 4.1 shows the equation for the relative disparity. All other metrics are defined in Appendix A.1.

$$RelativeDisparity = \frac{R_1}{R_2} \tag{4.1}$$

where R_1 is the highest and R_2 is the lowest evaluation metrics value within groups. Similar to other studies [72, 91], our workflow does not sample the test dataset, because the ground truth (i.e., new patient's disease or health label) is unknown in the real world.

Model specialization needs to rely on the whole group samples. Training a model solely based on particular subgroup samples (e.g., Black patients) gives poor results, worse than the original model on almost all metrics, due to small sample sizes.

4.3.2 Comparison with Other Bias Correction Techniques

The existing sampling approaches being compared include four undersampling techniques (namely, random undersampling, NearMiss1, NearMiss3, Distant method, stratified random undersampling), and four oversampling techniques (namely, replicated oversampling, SMOTE, ADASYN, Gamma, stratified random oversampling). Undersampling balances the distribution of the two prediction classes by selecting only a subset of the majority class cases.

Oversampling balances the dataset by populating the minority class. For decompensation prediction, we apply the two most commonly used sampling techniques, random undersampling (RUS) and replicated oversampling (ROS). For balancing the dataset using GAN, we use tabular generative adversarial network [48] to create synthetic minority samples that follow the existing minority dataset distribution. We have to exclude other sampling techniques as their pairwise quadratic distance computation is expensive for 2,377,768 patients' time series training dataset.

We also consider enriching the minority class (class 1) by creating synthetic datasets using generative models to balance the imbalanced healthcare dataset. For in-hospital mortality prediction tasks, the training dataset contains survival cases (Class 0) and mortality cases (class 1). As we know, there are only 13.5% of the data belong to class 1 (i.e., minority class). Therefore, we generate additional 9,935 class 1 synthetic data and add it with the existing 1,987 class 1 data so that both classes can be balanced. For generating the synthetic dataset, we use the TabGan [32] library which is used for generating tabular data with similar distributions to the existing mortality data.

4.3.3 Comparison with Reweighting

Following our DP design, we also invent a new prioritized reweighting approach. Prioritized reweighting selectively reweights specific subgroup minority samples, as opposed to reweighting all minority class C1 samples as in the standard reweighting. In the new prioritized reweighting method, we dynamically reweight minority class samples of selected demographic subgroups and choose the optimal machine learning model using the same metrics and procedure as in DP. Specifically, in each round of prioritized reweighting experiments, we multiply the selected samples' default weight by a unit number n, where n

ranges from 1 to 20. The weights of samples in other subgroups and majority class samples in the selected subgroup remain the default value, i.e., 1. These weights are used to train a machine learning model. Once the n machine learning models are trained, we follow DP's model selection operation for calibration and threshold selection.

4.3.4 Comparison with Seldonian

For the Seldonian algorithm, the demographic group information is needed to apply the constraint. Therefore, we train two seldonian models by adding age and race information respectively. We set the constraint for the seldonian algorithm so that the expected prediction error among the groups will be within ϵ=0.005 based on the Recall metric. The seldonian models passed the safety test which indicates that the model has sufficient confidence that it returns a solution given the constraints.

4.3.5 Cross-racial-group and Cross-age-group Experiments

We also perform a series of cross-group experiments, where enriched samples and test samples are from different demographic groups, i.e., group g used for Sample enrichment and test group g' are different. The purpose is to assess the impact of different machine learning models on prediction outcomes.

4.3.6 Whole-group vs. Subgroup-based Threshold Tuning

When analyzing the performance of the original model without bias correction, we evaluate two different settings. The first setting is to select an optimal threshold based on all samples in the validation set. We refer to the selected threshold as the whole group threshold. The second setting is to select an optimal threshold for each demographic subgroup based on that specific subgroup's performance in the validation set. We refer to the selected thresholds as the subgroup thresholds. In both settings, we calibrate the prediction on all samples (i.e., whole group) and select the thresholds with the top 3 highest F1 C1 scores and choose the one with the best-balanced accuracy.

4.4 Evaluation and Findings

In this section, we present an empirical study of disparity in biased prediction and the impact of DP over other sampling techniques.

4.4.1 Experimental Setup

Prediction and data analysis code are in Python programming language. The hospital record prediction tasks were executed on a virtual machine with Ubuntu 18.04 operating system, x86-64 architecture, 8 cores, 40 GB RAM, and 1 GPU. Cancer survivability prediction tasks were performed using a Ubuntu 21.04 operating system, x86-64 architecture, 16 cores, 40 GB RAM, and 1 GPU.

Dataset

We use MIMIC III (Medical Information Mart for Intensive Care) [81, 87] and SEER (Surveillance, Epidemiology, and End Results) cancer datasets [6], both collected in the US.

Table 4.1: Learning parameters for four prediction models

Learning Parameter	**BCS Prediction**	**IHM Prediction**	**LCS Prediction**	**Decomp Prediction**
Hidden layers	(20, 20)	(16, 16)	(20, 20)	(128)
ANN	MLP	LSTM	MLP	LSTM
Learning Rate	0.001	0.001	0.001	0.001
Optimizer	adam	adam	adam	adam
Dropout	0.1	0.3	0.1	0

Machine Learning Models

We test existing machine learning models in a clinical prediction benchmark [81] for MIMIC III and reproducible cancer survival prediction [84] for SEER. We study a total of four binary classification tasks, in–hospital mortality (IHM) prediction and decompensation prediction from the clinical prediction benchmark, 5-year breast cancer survivability (BCS) prediction, and 5-year lung cancer survivability (LCS) prediction. In what follows, we denote the minority prediction class as Class 1 (or C1) and the majority class as Class 0 (or C0). For decompensation prediction on the MIMIC III dataset, the minority class C1 represents patients whose health condition.

Two types of neural networks are used, the long short-term memory (LSTM) model and the multilayer perceptron (MLP) model. Following Harutyunyan et al [81], for the hospital record prediction tasks, patients' data is preprocessed into time-series records and fed into an LSTM model. Cancer survivability prediction utilizes an MLP model, following Hegselmann et al. [84]. Model parameters remain constant in different bias correction techniques (Table 4.1).

While comparing with the seldonian algorithm, we use logistic regression classifiers with L2 regularization with regularization strength 1000 (i.e., C = 0.001).

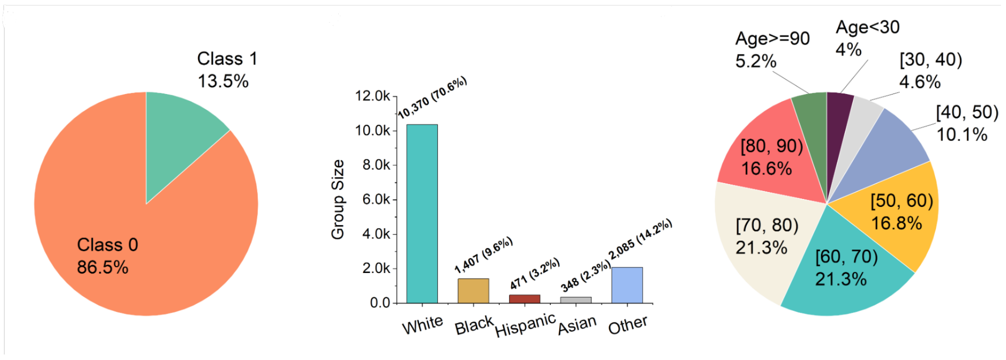


Figure 4.2: Statistics of dataset for IHM prediction

4.4.2 Analysis of Imbalanced Clinical Datasets

Figure 4.2 shows the composition of IHM training data, which contains 14,681 time-series samples from MIMIC III. The majority of the records (86.5%) belong to Class 0 (i.e., patients who do not die in the hospital). The rest (13.5%) belong to Class 1 (i.e., the patients who die in the hospital). The percentage of Class 1 samples within each subgroup slightly varies but is consistently low. 70.6% of the patients are White and 76% belong to the age range [50, 90). 45.1% of the patients are females and 54.9% are males. The training set contains insufficient data for the young adult population. Distributions of the decompensation training dataset (of size 2,377,768) are similar (Figure 4.3).

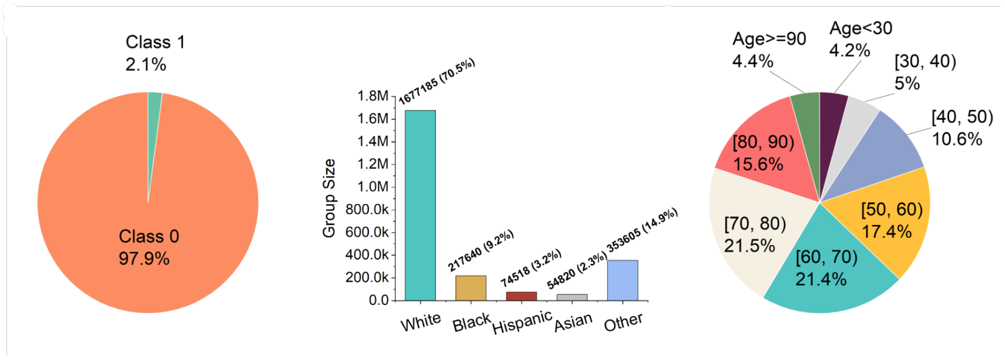


Figure 4.3: Statistics of dataset for Decomp prediction

Frequency distributions of features for MIMIC training data used for IHM prediction are

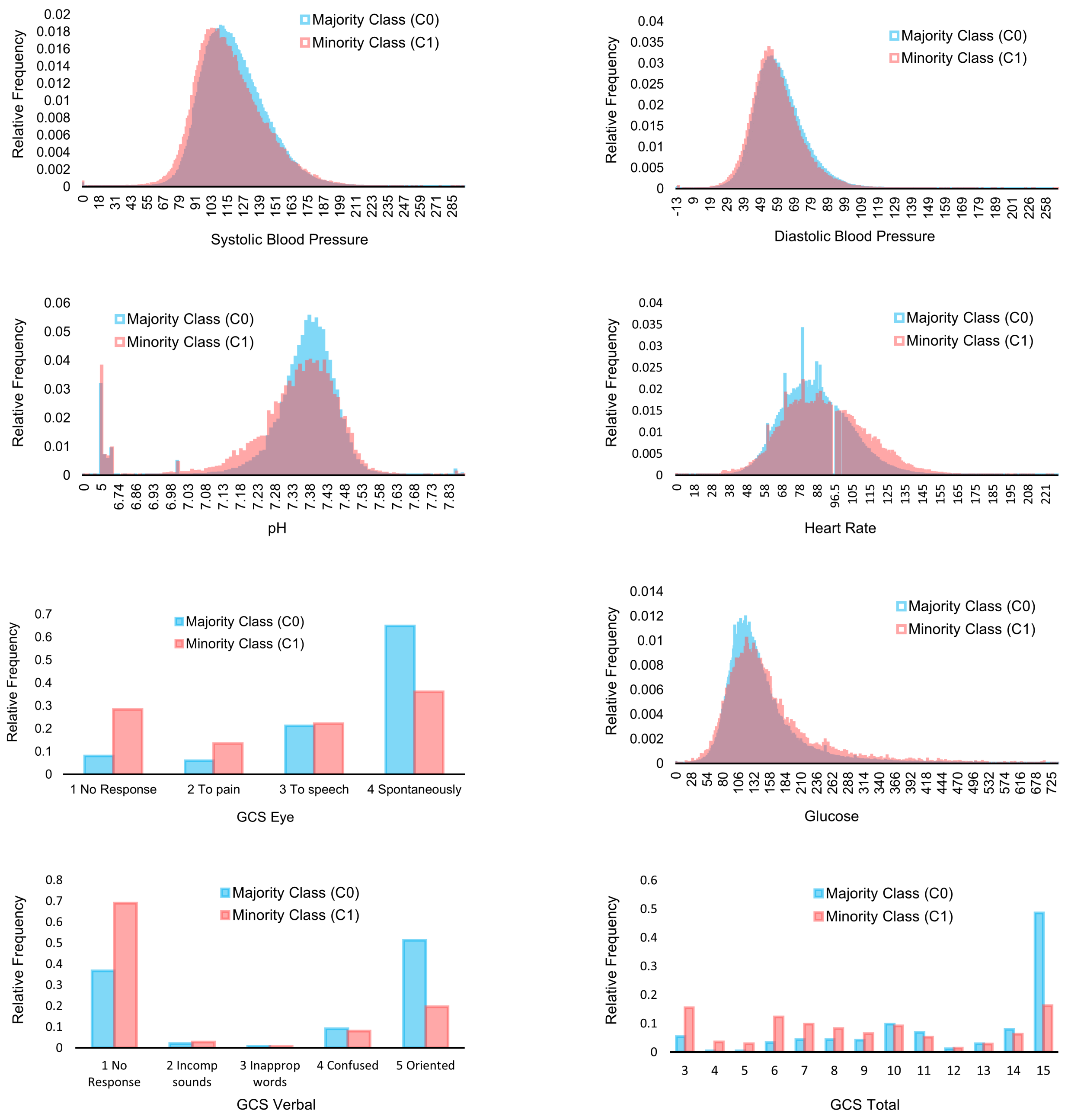
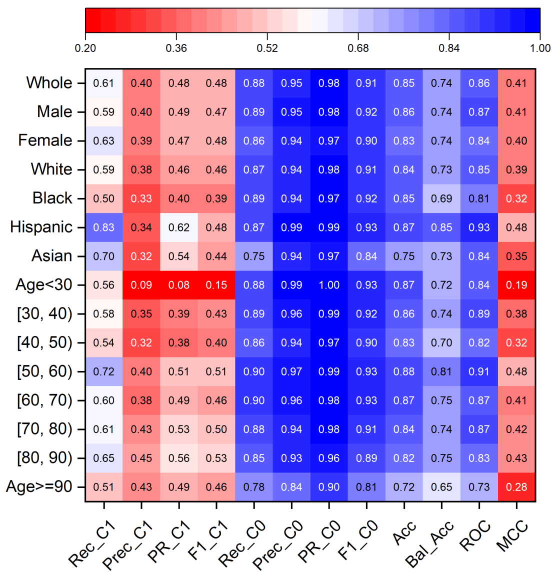


Figure 4.4: Relative frequency of several features in the MIMIC dataset for in-hospital mortality tasks

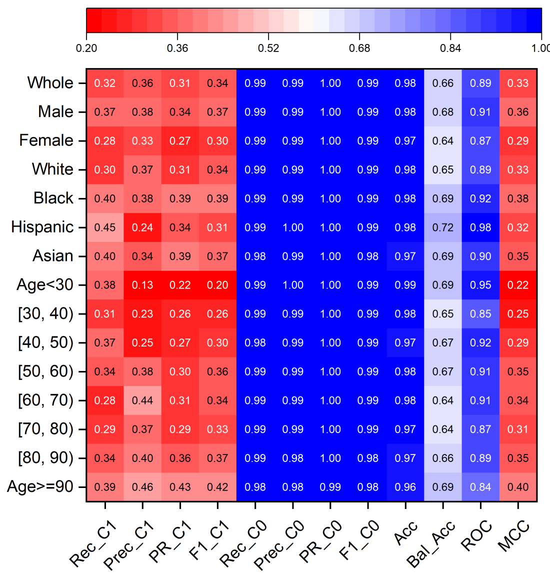
shown in Figure 4.4. We show the distribution in terms of relative frequency in two classes (C1 and C0). The remaining feature's relative frequency is also shown in Appendix (Figure A.1). There are in total 17 clinical features of patients used in IHM and decomp prediction tasks. These are systolic blood pressure, diastolic blood pressure, pH, heart rate, GCS eye scale, GCS motor scale, GCS verbal scale, GCS total, glucose, capillary refill rate, fraction inspired oxygen, height, weight, oxygen saturation, mean blood pressure, temperature and respiratory rate.

4.4.3 Disparity Among Prediction Classes

Without any bias correction, the original machine learning model demonstrates a substantial accuracy disparity between the majority prediction class C0 and the minority prediction class C1. For IHM, the Recall value (0.61) for the minority class C1 is 31% lower than the Recall of the majority class (0.88). For Decomp, the Recall_C0 (0.99) is three times higher than the Recall_C1 (0.32). This disparity is consistently observed for various demographic groups, with a few exceptions of senior patients for BCS prediction. We further show detailed IHM predictions with the MIMIC III dataset for various subpopulations under 12 metrics in a heatmap in Figure 4.5a. 12.4% of non-death cases (class C0) in IHM prediction are wrong, whereas the missed mortality prediction (class C1) rate is much higher at 39%. For Black patients, while recall, precision, F1, and AUC-PR are all above or equal to 0.89 for class C0, the recall of class C1 is only 0.50, i.e., for every 100 Black patients who die in hospital, the model would mispredict 50 of them. A similar trend is observed for the Decomp prediction results (Figure 4.5b).



(a) IHM prediction results



(b) Decomp prediction results

Figure 4.5: Prediction results under the original machine learning models (no bias correction) using one optimized threshold for all demographic groups.

4.4.4 Disparity Across Demographic Subgroups

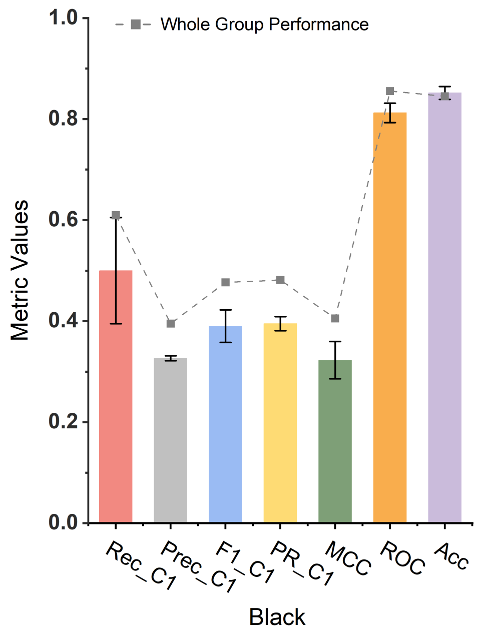
Besides disparity between prediction classes, the original model also shows disparity across demographic subgroups. For the IHM prediction (Figure 4.5a), Black patients have the lowest minority class C1 recall (0.50), lower than the whole group (0.61) and Hispanic patients (0.83). The disparity among C1 recalls of various age subgroups is lower, all in the range of [0.51, 0.72]. Most subgroups have somewhat similar C1 precision values, except the age <30 group. Young patients under 30 have a low C1 precision of 0.09, substantially lower than the whole population (0.40). Young patients under 30 accounts for only around 4% MIMIC III datasets (Figure 4.2), respectively. Their predictions are consistently poor. Despite the large disparity in minority class C1 performance, majority class C0 precisions and recalls are consistently high for all subgroups, with most values above 0.85. Despite small sample sizes, some demographic groups (e.g., Hispanic groups in IHM prediction) have high prediction accuracies even without sampling. For decomp prediction (Figure 4.5b), prediction accuracy also differs across demographic subgroups, e.g., C1 precision is 0.46 for age 90+ patients and 0.13 for age <30 patients.

4.4.5 Disparity Among Performance Metrics

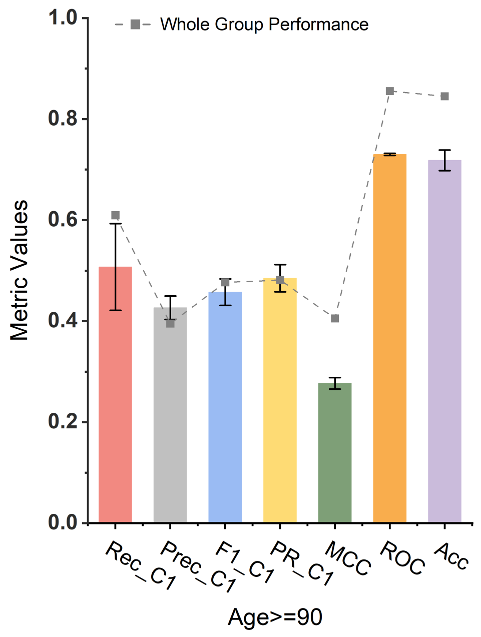
For imbalanced datasets, commonly used metrics such as AUC-ROC and accuracy are deceptive and do not reflect minority class performance. These metrics may show misleadingly higher values, even when the performance of the minority class is poor. Figure 4.6 shows that the overall accuracy and AUC-ROC values are consistently high (> 0.80 in most cases) across different subgroups, even when minority class C1's performance is dismal, e.g., the F1-score is only 0.39 for Black patients in IHM prediction. Accuracy and AUC-ROC values are dominated by the overwhelmingly high precision and recall (> 0.85 in most cases) of the majority prediction class C0. Thus, these commonly used metrics in prediction do not reflect the minority class performance under data imbalance. In biased datasets, AUC-ROC is no longer sufficient, as it covers both classes with one dominating class. This deficiency is well established in the machine learning literature [67, 70, 71], where multiple previous studies pointed out that AUC-ROC gives an overly optimistic view of imbalanced classification. Our work points out the severity of the metrics issue in digital health applications.

4.4.6 DP Method Reduces Disparity

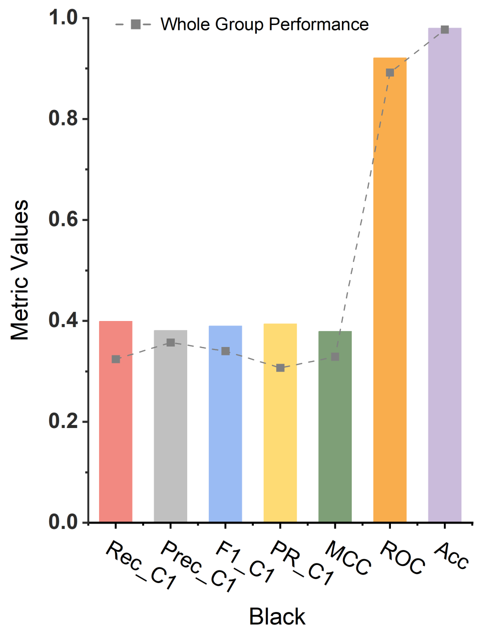
We use relative disparity (defined in Equation 4.1) as a metric to quantify accuracy gaps across demographic subgroups under various machine learning conditions, including the original model (without any bias correction), DP bias correction, and existing sampling methods. Relative disparity measurement below 1.25 is considered fair, following the 80% rule for as-



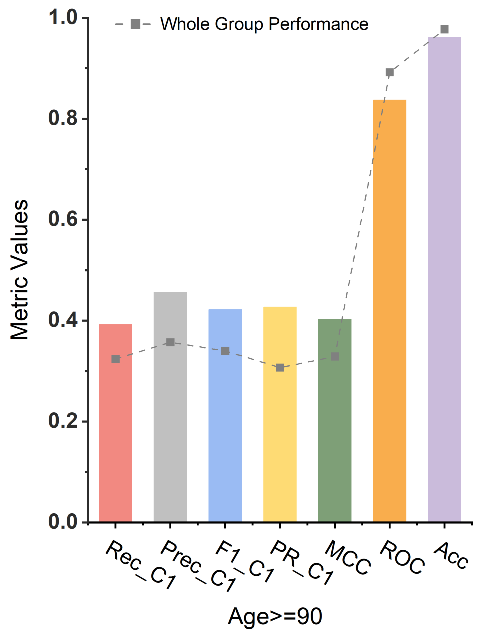
(a) Black subgroup performance for IHM task



(b) Age 90+ subgroup performance for IHM task



(c) Black subgroup performance for Decomp task



(d) Age 90+ subgroup performance for Decomp task

Figure 4.6: Performance metrics disparity

sessing disparate impact [14]. Our results show that machine learning models trained with our DP bias correction method exhibit the smallest racial and age disparities (Figure 4.7). For balanced accuracy, C1 recall and MCC of both IHM and Decomp task, most of DP's relative disparity values are in the fair range (1.25 and lower), substantially reducing the disparity in the original model. Specifically, DP has a 14.8% to 23.9% improvement over the original model in terms of C1 recall disparity. We observe a similar reduction in balanced accuracy disparity and MCC disparity.

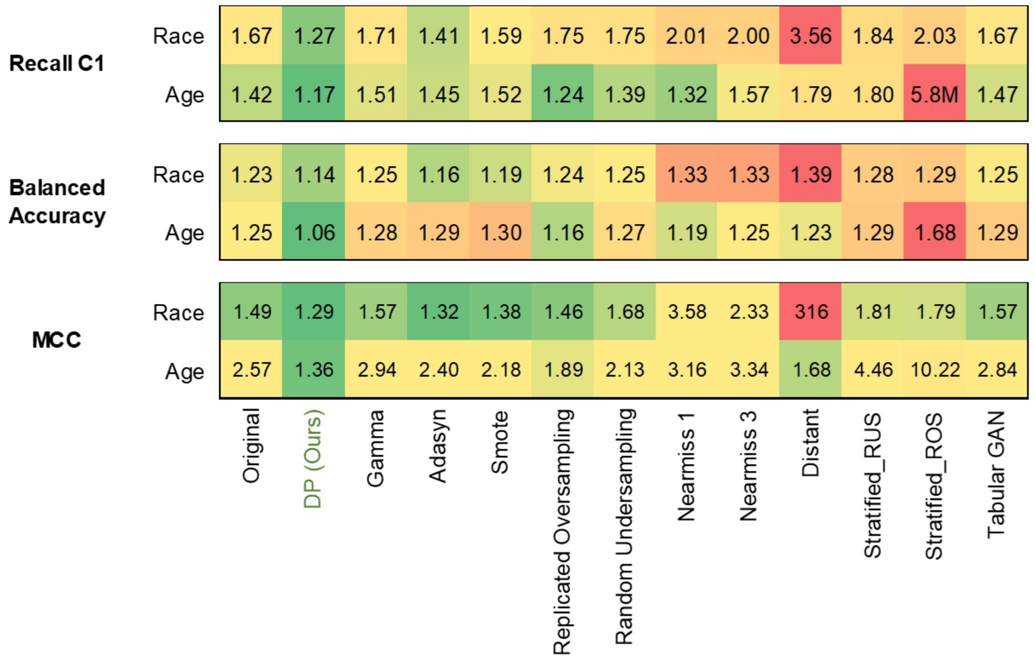


Figure 4.7: Relative disparity among racial and age groups under various sampling conditions for IHM prediction

In contrast, all three state-of-the-art sampling methods (namely, Gamma, Adasyn, and SMOTE) fail to reduce the racial and age disparities in the IHM task, with some models (e.g., Gamma) slightly exacerbating disparity. Undersampling methods (especially Distant) perform even worse than these oversampling methods. When compared to the eleven existing methods, DP reduces racial disparity by 10.2% (ADASYN) to 64.3% (Distant). The

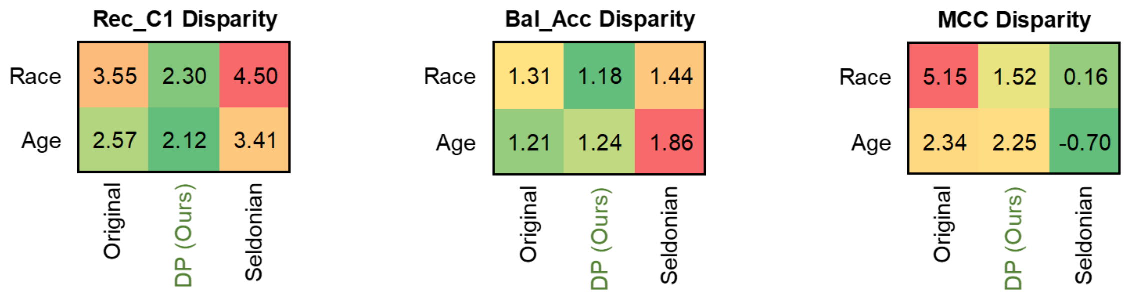


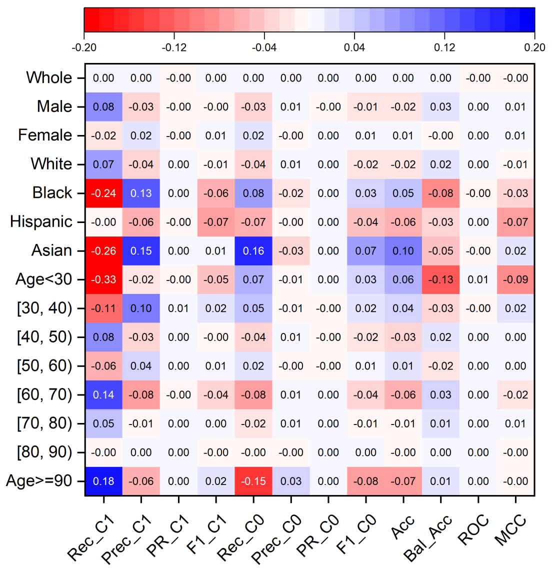
Figure 4.8: Relative disparity among racial and age groups for In-hospital mortality prediction using logistic regression models

age disparity is higher than 5.6%, in terms of the minority C1 recall for IHM prediction. (Figure 4.7). Balanced accuracy and MCC results follow a similar trend.

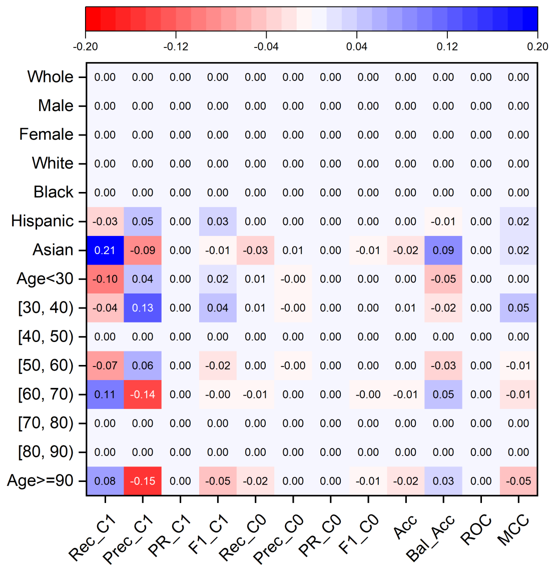
Regarding fairness for the decompensation task, the relative disparity of DP is lower than or comparable to other sampling approaches for most cases (Figure A.4) which is consistent with the trend observed in Figure 4.5a. We examine an exceptional case for race groups in terms of recall, where the high Hispanic group performance (0.76) increases the disparity value.

we also compute in-hospital mortality prediction tasks using ML regression models. We compare our proposed DP technique with the seldonian algorithm that constrains behavior to maintain fairness.

Figure 4.8 shows the relative disparity values in terms of Recall C1, Balanced Accuracy, and MCC metric. From Recall C1 disparity, we find that the disparity significantly decreases using our proposed DP technique. However, the seldonian models increase disparity than the original models. A similar trend is observed for balanced accuracy. However, in terms of the MCC metric, the seldonian shows significantly lower disparity values. We find that the seldonian algorithm significantly decreases subgroup performance in terms of precision and MCC.



(a) Difference in IHM prediction results

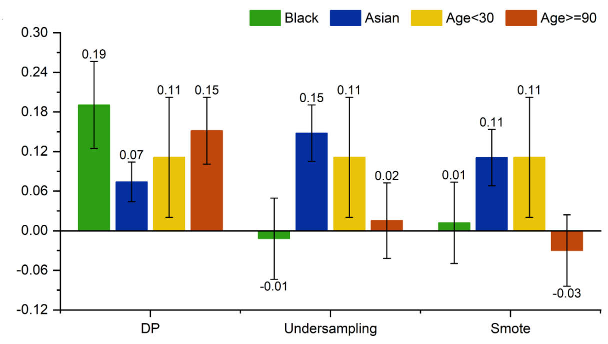


(b) Difference in Decomp prediction results

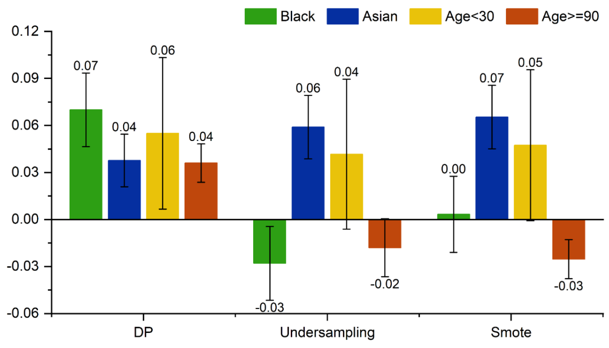
Figure 4.9: Difference in performance of the original machine learning models (no bias correction) using subgroup thresholds (i.e., different optimized thresholds for different demographic groups) and whole group threshold.

4.4.7 Mitigation Solely Based on Adjusting Thresholds

We also test whether or not threshold tuning alone can boost the performance of demographic subgroups and reduce disparity. Specifically, we compare the prediction performance under the whole group threshold and subgroup thresholds, which are described in the Methods section. Prediction results under the original machine learning models (no bias correction) using different optimized thresholds for different demographic groups are shown in Figure 4.9. For the IHM task, the performance differences between using the whole-group threshold and subgroup threshold are small (< 0.1), in terms of C1 precision and recall, for subgroups with relatively large sizes (e.g. middle-aged patients). However, for other smaller subgroups (e.g. young patients with age<30), the performance decreases. A likely reason is overfitting, i.e., the threshold selected based on a small sample size in the validation set is not optimal on the test set, due to the small sample sizes. Decomp results follow similar patterns.



(a) Difference in Recall C1



(b) Difference in Balanced Accuracy

Figure 4.10: DP and two representative sampling techniques performance comparison over the original model for IHM Prediction

Thus, threshold adjustment alone is clearly insufficient for the data imbalance and disparity problems.

4.4.8 Subpopulation-based vs. Whole-population-based Sampling

Existing sampling solutions do not differentiate subpopulations. We found such whole-population-based sampling methods decrease the performance of some underrepresented groups. In Figure 6, we compare DP with two common sampling techniques (i.e., random undersampling and SMOTE) with four demographic groups (namely, Black, Asian, age < 30, 90+ for the IHM task, and Hispanic, Asian, age <30, 90+ for the BCS task). These groups are chosen because of their low performances under the original machine learning model. Figure A.7 shows that DP consistently boosts the performance of most underrepresented demographic groups. In contrast, this consistent improvement is not observed in the other two methods. For example, for the IHM task, although the undersampling technique boosts the balanced accuracy for Asian patients, the performance of Black and age 90+ subgroups slightly decreases (Figure 4.10b). The complete comparison results with the 11 existing sampling methods are shown in Figure 4.11 for the black subgroup and Figure 4.12

for the age 90+ subgroup.

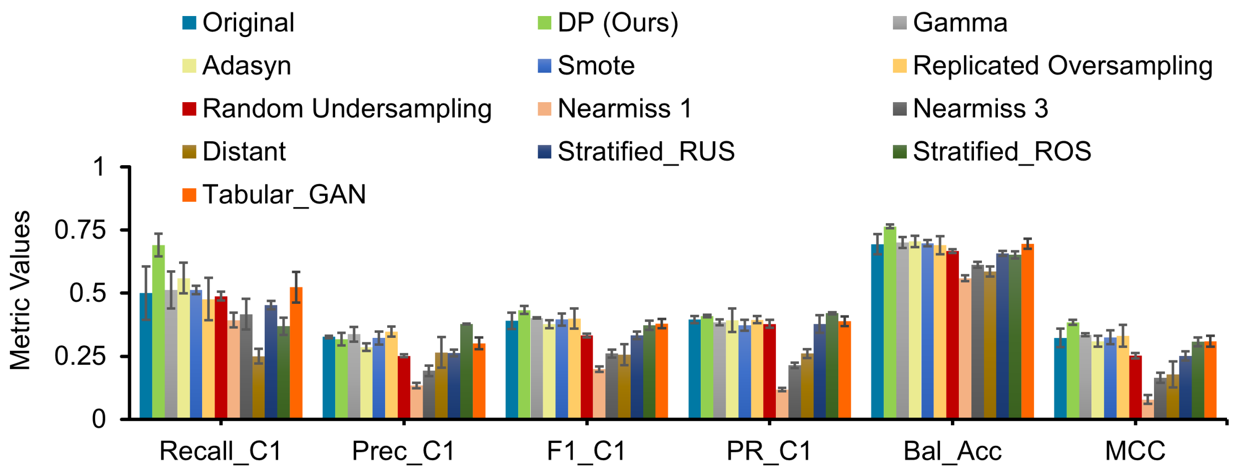


Figure 4.11: In-hospital mortality (IHM) prediction under various sampling conditions for black subgroup

We perform the Kruskal Wallis Test [5] on the results of sampling techniques. Kruskal Wallis test is a non-parametric method that we use to compare two different sampling techniques with DP shown in Figure 4.11. We find that the p-value in terms of minority class Recall and balanced accuracy is between 0.046 and 0.0495 which is less than 0.05. Therefore, we reject the null hypothesis, meaning the sampling techniques do not perform equally as DP. DP achieves better performance in terms of minority class recall and balanced accuracy. In terms of the minority class F1 score, we see the p-value higher than 0.05 for the original model, Smote, replicated oversampling technique. Therefore, we can not reject the null hypothesis for these cases. Hence, we do not have sufficient proof to claim that there are statistically significant differences among these models. For the other nine sampling technique, the p-value is less than 0.05. Therefore, DP shows significant performance improvement over than nine sampling techniques in terms of minority class F1 value. In terms of minority class PR curve value, we only see the p-value higher than 0.05 for the original model, Adasyn, Smote, Replicated oversampling, stratified undersampling, and Tabular GAN-based sampling For the other 6 sampling techniques, the p-value is lower than 0.05. The detailed p-values are shown in Table A.1.

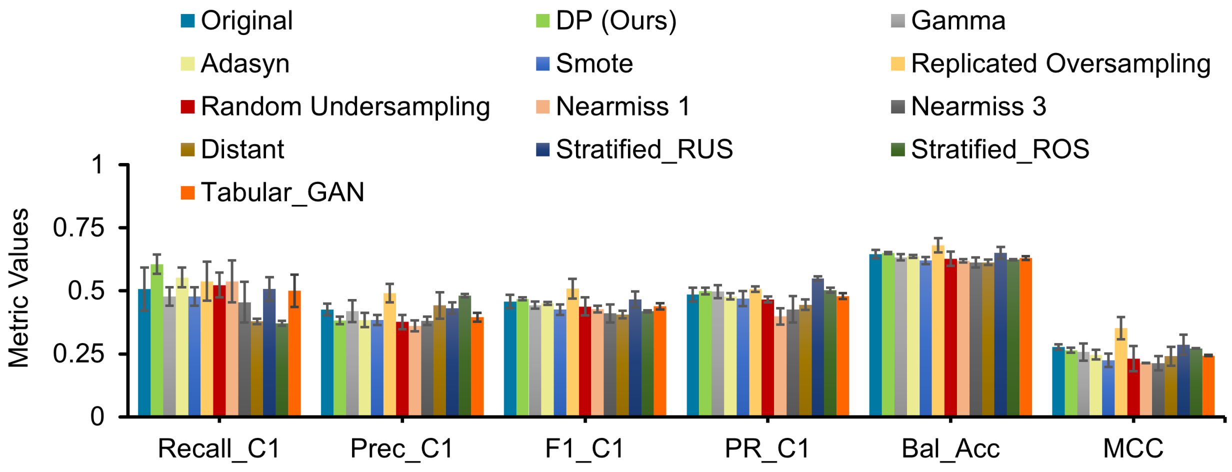


Figure 4.12: In-hospital mortality (IHM) prediction under various sampling conditions for age 90+ subgroup

From the original regression model, we observe that the Hispanic and Age 90+ population shows the worst performance among the racial and age population group respectively. We show these population group performances using DP and the seldonian algorithm in Figure 4.13. DP shows better performance than the original and the seldonian algorithm in terms of Recall_C1, F1_C1, Balanced Accuracy, and MCC.

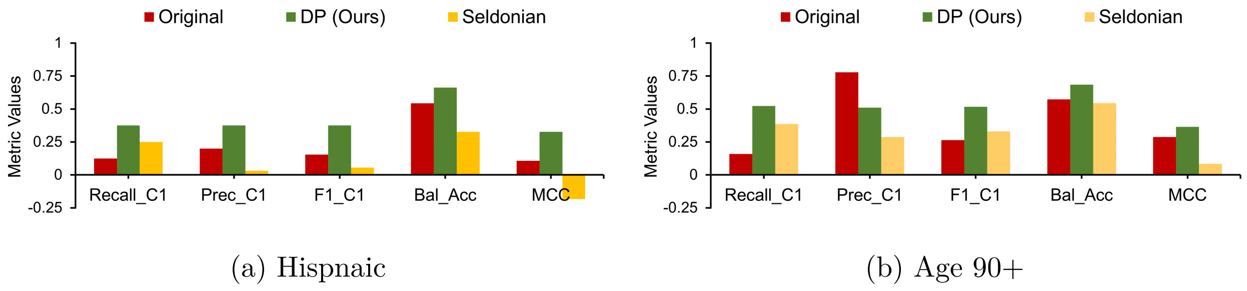
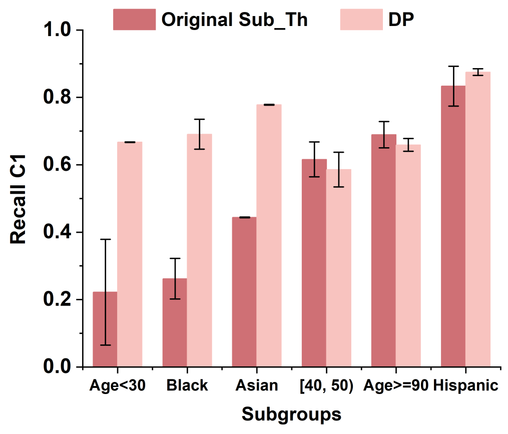
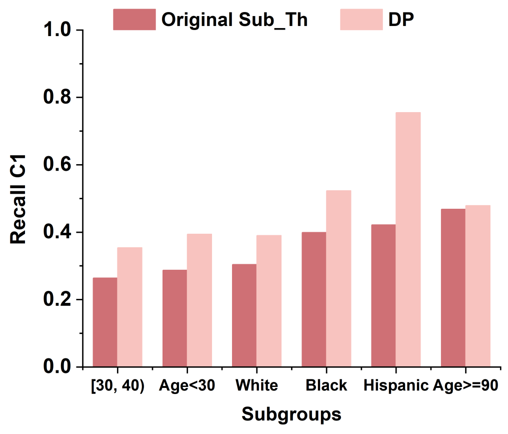


Figure 4.13: In-hospital mortality prediction using logistic regression models

For subgroups with lower original performance, DP brings stronger C1 recall improvements. We show this trend in Figure 4.14, where we compare the minority class recall between the original model with the subgroup threshold and the DP model trained for each subgroup. For the IHM task, DP improves the C1 recall by 200.4%, 163.4%, and 75.2%, respectively, for the age <30, Black, and Asian patients (Figure 4.14a). For the decomp task, DP shows similar performance in Figure 4.14b. However, we see some exceptions (e.g., Hispanic subgroup).



(a) IHM prediction



(b) Decomp prediction

Figure 4.14: Performance of DP and subgroup-threshold-based original model in terms of minority class recall

4.4.9 Cross-group: Prediction Outcome with Specialized ML

In our cross-group experiments, we use the DP model trained for demographic group A (e.g., Black) to predict group B (e.g., Hispanic). The aim is to evaluate the impact of different machine learning models on prediction outcomes. We perform both cross-race and cross-age-group experiments for IHM prediction (Figure 4.15), which involve three underrepresented races and three underrepresented age groups For IHM prediction, DP models' advantage is observed in three out of the six groups (for Black, <30, and 90+ groups), which is less pronounced than BCS prediction. In the cross-age-group experiment, both DP <30 and 90+ models demonstrate advantages. For Hispanic and Asian patients, the DP Black model gives the best recall C1, higher than DP Hispanic and DP Asian models. Figure A.5 shows

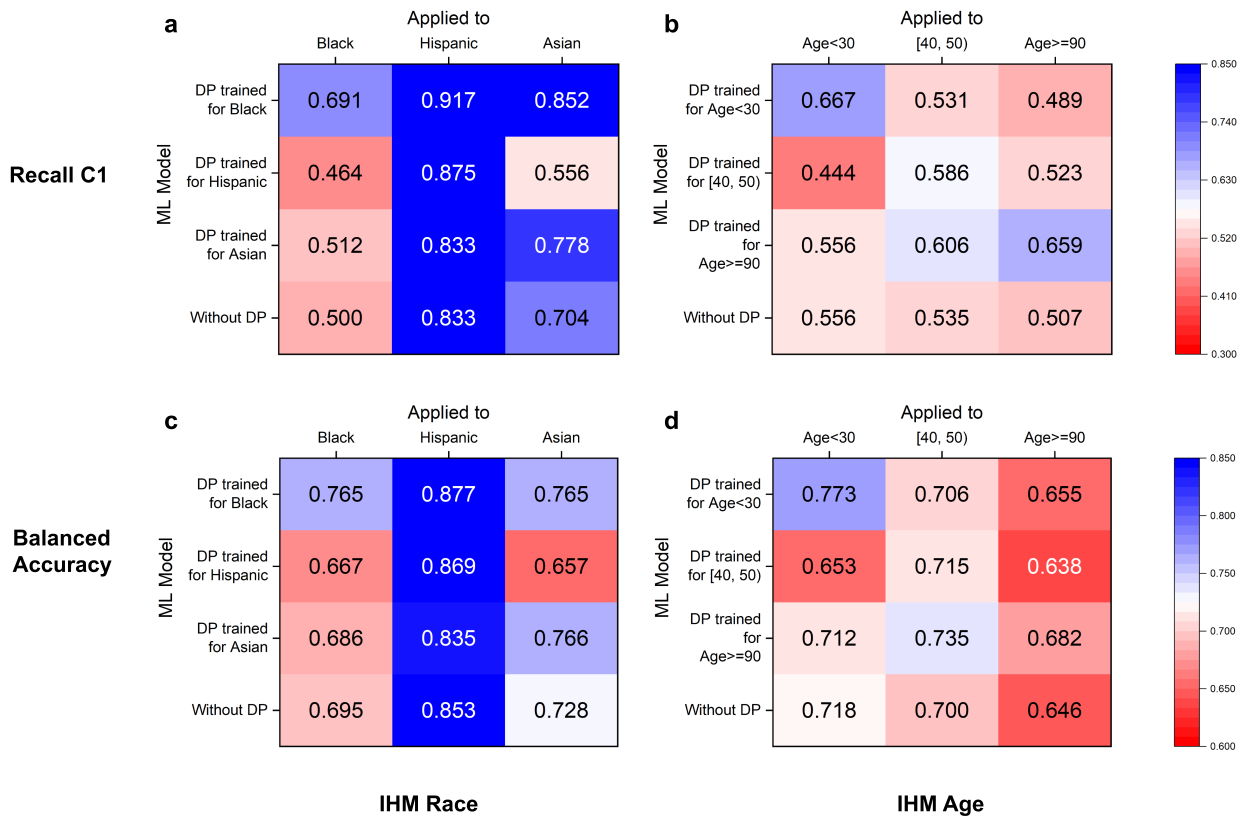
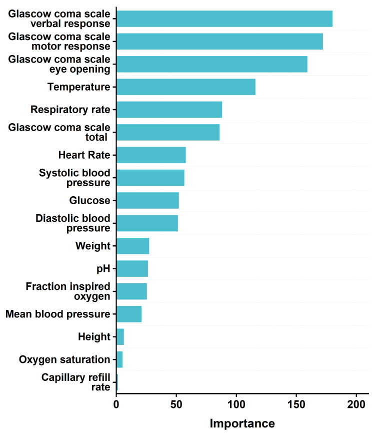


Figure 4.15: DP's cross-group performance under various race and age settings for recall C1 and balanced accuracy for the IHM prediction

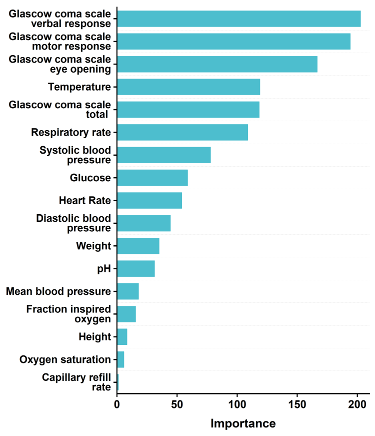
that matching DP models show some degree of advantage in four out of six settings for the decompensation task.

4.4.10 Feature Importance

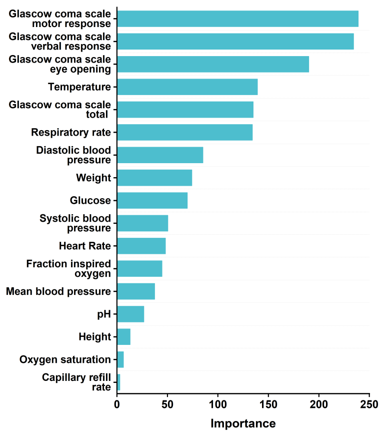
We compute feature importance using SHAP-sum where the importance of columns representing the same variable is summed up. For the IHM prediction task under SHAP-sum, the top features of the DP models and the original models are similar, slightly differing in their feature ordering (Figure 4.16). For example, for IHM prediction DP age 90+ model ranks weight at the fourth position, slightly higher than its ranking in the DP Black and the original models (both at the seventh position). This observation may suggest that being overweight in older patients is more likely to cause serious consequences. Following the existing benchmark17, our IHM and decompensation predictions only use 17 clinical features and exclude race and age information in MIMIC III. We found that SHAP-sum identifies very different top features from SHAP-avg, highlighting categorical features due to their multiple one-hot encoding representations for machine learning. We also show the SHAP-avg feature ranking of IHM prediction in Figure A.8.



(a) Original Model



(b) DP model for Black



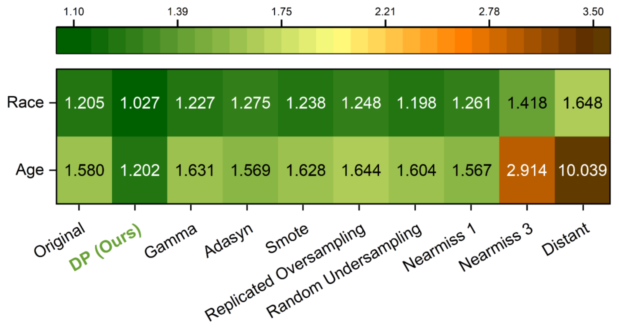
(c) DP model for Age 90+

Figure 4.16: SHAP-sum feature importance of different IHM experiments.

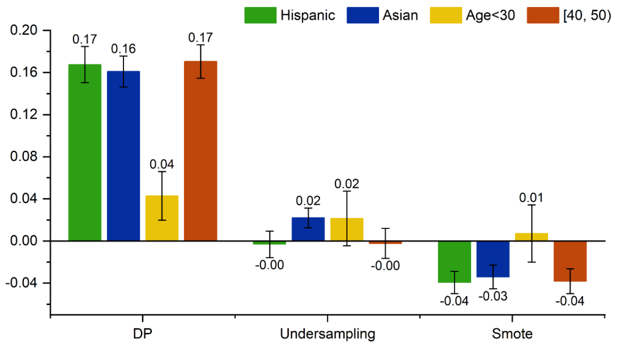
4.4.11 Cancer Survivability Prediction Tasks

We repeat the experiments for the other two tasks, 5-year breast cancer survivability (BCS), and lung cancer survivability (LCS) prediction on the SEER dataset, and observe similar patterns.

The advantage of DP is still consistently observed for BCS prediction. Overall, DP shows 14.3% (Random Undersampling) to 37.7% (Distant) improvement among racial groups and



(a) Relative Disparity

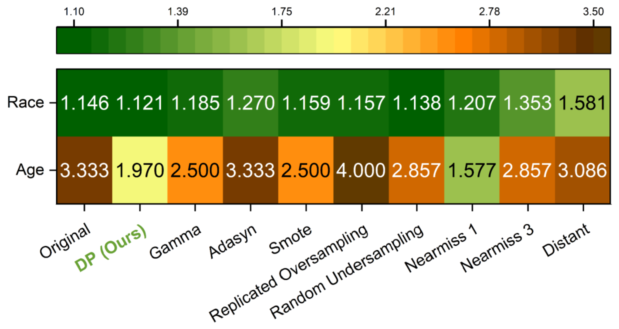


(b) Performance comparison

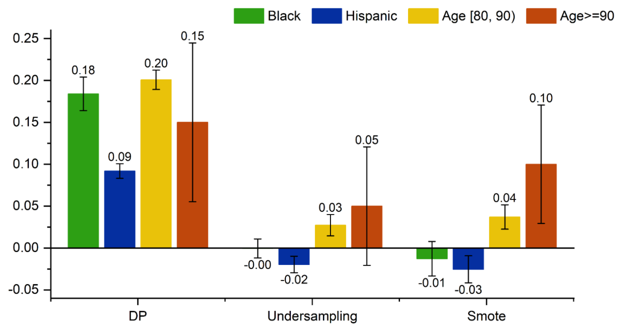
Figure 4.17: Relative disparity and performance comparison over the original model for BCS prediction task in terms of Recall C1

23.3% (NearMiss1) to 88.0% (Distant) improvement for age groups in terms of C1 recall (Figure 4.17a) compared to existing sampling methods. SMOTE slightly decreases the C1 recall for the Hispanic, Asian, and age [40,50) groups (Figure 4.17b)

Similar advantages for DP are also present in LCS prediction (Figure 4.18). One exception is that NearMiss1 undersampling shows the lowest relative disparity for age groups in terms of C1 recall (Supplementary Figures 4.18a). While NearMiss1 brings C1 Recall of all age groups to a relatively good range of [0.63, 1.00], its C1 precision ([0.03, 0.54]) and C1 AUC-PR ([0.02, 0.86]) are poor, resulting in high disparity.



(a) Relative Disparity



(b) Performance comparison

Figure 4.18: Relative disparity and performance comparison over the original model for LCS prediction task in terms of Recall C1

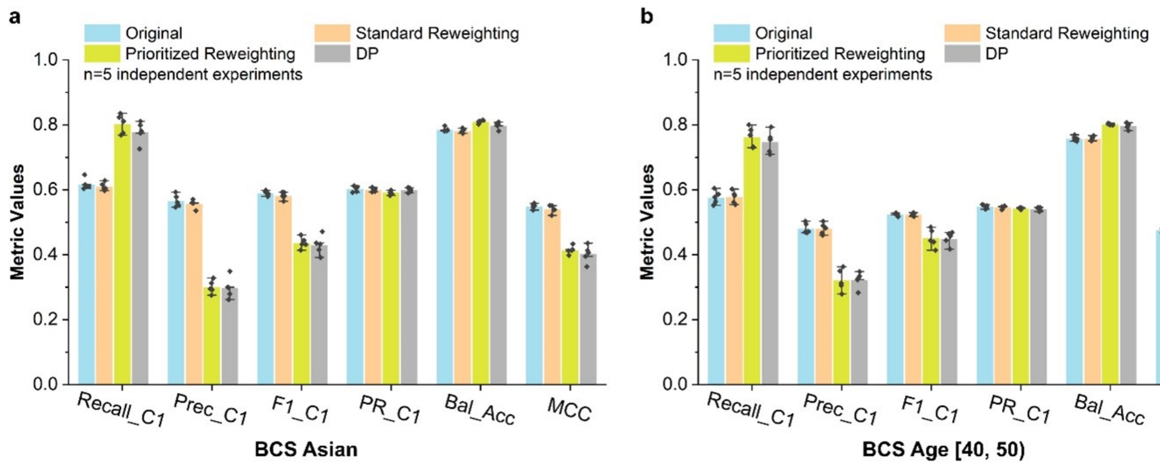


Figure 4.19: Performance comparison of the original model (without bias correction), standard reweighting, prioritized reweighting, and DP for Asian patients and [40, 50) patients for BCS prediction task

We compare DP with two methods, the standard reweighting method and a new prioritized reweighting method. The standard reweighting models, where reweighting does not differentiate subpopulations, perform almost identically to the original model when applied to Asian and age [40, 50) patient groups (Figure 4.19). In contrast, prioritized reweighting, where new weights are optimally placed on a specific group of patients, boosts C1 recall in BCS prediction for Asian patients from 0.617 to 0.802 and from 0.577 to 0.763 for age [40, 50) patients. This boost is comparable to DP's performance. DP and prioritized reweighting also exhibit comparable performances under other metrics. In our standard reweighting experiment, the minority class has a weight of 3.94 and the majority class has a weight of 0.57 for BCS prediction.

4.5 Discussion

Our findings empirically demonstrate multiple deficiencies of typical machine learning prognosis procedures when they are applied to imbalanced medical datasets. One deficiency is that the weak performance of underrepresented patients may be eclipsed by the whole population's performance and not accurately reported. Underrepresentation is twofold: (i) demographic subgroups and (ii) the minority prediction class. The low accuracy problem is particularly severe when a patient belongs to both categories. For example, for the IHM prediction, Black patients' C1 recall (0.50) is 18% lower than the whole group (0.61) (Figure 4.5). Low recalls in the disease group can lead to underestimation of risks, missed treatment opportunities, or potentially life-threatening wrong prognoses. In addition, racial and age disparities in machine learning-based prognoses are also observed.

A key contribution of our work is to systematically compare the conventional one-model-fits-all approach with a new double prioritized (DP) bias correction approach, where specialized prognosis models are trained for minority prediction class patients of a certain race or age. Conceivably, it is challenging to train a single machine learning model that optimizes for all demographic groups. In contrast, the DP bias correction technique allows one to train models for specific demographic groups, not having to use the same model for the entire patient population. The key enabler of DP is demographic-specific sampling, i.e., selectively enriching the number of samples in the minority prediction class (C1). Training a specific machine learning model for some patient groups is necessary. For example, the oldest-old age group (typically defined as 85+)[98] is a growing population in the US [20]. However, our study shows that 90+ patients' recall C1 value (0.51) in the mortality prediction is 16% lower than the whole group (0.61) in the original model. Prioritized bias correction is highly effective for improving C1 recalls of demographic subgroups who are underrepresented in the training data, e.g., DP's recall C1 is 0.66 (29.4% improvement) for 90+ patients in mortality prediction.

Our results show that DP can mitigate racial and age disparities introduced by data underrepresentation in training machine learning models, better than the existing 11 sampling methods being compared. However, data imbalance is only one source of disparity. For example, the diagnosis and treatment conditions may vary across different demographic subgroups and affect data quality. These variations may also contribute to the disparity observed across groups. Eliminating such more fundamental and systemic medical biases is beyond the scope of technical solutions.

As underrepresentation is prevalent in clinical medicine, our findings likely have broad implications beyond the specific datasets and demographic groups studied. Fully recognizing accuracy disparities associated with imbalanced data will help reduce life-threatening prediction mistakes. Future directions of this work can be developing more demographic-specific sample enrichment techniques, as well as exploring how data underrepresentation impacts the quality of medical image analysis and mutation-based evolutionary computation.

4.5.1 Limitations of Double Prioritized Sampling

For double prioritized sampling, we create separate models (e.g., dp black model, dp age<30 model) for minority demographic groups that take more time to train and more space to store these models. Moreover, during the training phase of each demographic model, we train $(n + 1)$ models before choosing a final optimal candidate model that also takes more time and space. However, these training processes can be done in parallel if enough computing resources are available. Parallel training can be reduced the total training time to a single model training time.

Approaches toward separate models can be observed for precision medicine [94] that maximizes the quality of health care by individualizing the healthcare procedure to the uniquely evolving health status of each patient. Therefore, the precision medicine concept also supports the multi-model setup.

Our current workflow (Figure 4.1) improves overall performance, especially minority class recall performance. However, if a prediction application requires improvement in terms of other metrics (e.g., precision), other performance metrics can be considered as model selection metrics and threshold selection metrics.

4.6 Summary

Many clinical datasets are intrinsically imbalanced and dominated by overwhelming majority groups. Off-the-shelf machine learning models that optimize the prognosis of majority patient types (e.g., healthy class) may cause substantial errors in the minority prediction class (e.g., disease class) and demographic subgroups (e.g., Black or young patients). In the typical one-machine-learning-model-fits-all paradigm, racial and age disparities are likely to exist but unreported. In addition, some widely used whole-population metrics give misleading results. We design a double prioritized (DP) bias correction technique to mitigate representational biases in machine learning-based prognosis. Our method trains customized machine learning models for specific ethnicity or age groups, a substantial departure from the one-model-predicts-all convention. We compare with other sampling and reweighting techniques in mortality and cancer survivability prediction tasks. We first provide empirical evidence showing various prediction deficiencies in a typical machine learning setting without bias correction. For example, missed death cases are 3.14 times higher than missed survival cases for mortality prediction. Then, we show DP consistently boosts the minority class recall for underrepresented groups, by up to 38.0%. DP also reduces relative disparities across race and age groups, e.g., up to 88.0% better than the 8 existing sampling solutions in terms of the relative disparity of minority class recall. The cross-race and cross-age-group evaluation also suggests the need for subpopulation-specific machine learning models. Biases exist in the widely accepted one-machine-learning-model-fits-all-population approach. We invent a bias correction method that produces specialized machine learning prognostication models for underrepresented racial and age groups. This technique may reduce potentially life-threatening prediction mistakes for minority populations.

Chapter 5

Conclusions and Future Work

In this chapter, we will conclude and provide future directions for researchers.

5.1 Conclusion

This dissertation describes several methodologies for improving the performance of classification tasks in critical applications. We consider two critical application domains. One is software security and another is healthcare. In software security, we described our effort of developing a benchmark named CryptoAPI-Bench [39, 122] and ApacheCryptoAPI-Bench [41]. These benchmarks include 18 common crypto misuse categories that we find from NIST documents, previous research papers, and different blogs. The objectives of these benchmarks are to identify the limitations of Java cryptographic vulnerability detection tools and improve their tools. These benchmarks can also educate novice developers as the benchmark contains both cryptographically secure and insecure cryptographic API usage. We also evaluated and compared four state-of-the-art cryptographic vulnerability detection tools (CryptoGuard, CrySL, SpotBugs, and Anonymous tool) and showed their limitation and coverage. Many tools including CryptoGuard, CrySL, and Parfait (crypto vulnerability detection tool by Oracle) used our benchmark to improve the performance of their tools.

In healthcare prognosis, as underrepresentation is prevalent in clinical medicine, our findings likely have broad implications beyond the specific datasets and demographic groups studied.

Fully recognizing accuracy disparities associated with imbalanced data will help reduce potentially life-threatening prediction mistakes. Vast accuracy gaps exist between minority C1 and majority C0 classes and across some demographic subgroups. When training and testing machine learning models, using multiple metrics is crucial, including balanced accuracy and separate metrics for the two prediction classes. Commonly used metrics, namely AUC-ROC and accuracy, are heavily influenced by the majority class and may fail to reflect the minority class performance when the dataset is imbalanced. Existing sampling techniques (e.g., SMOTE, ADASYN, Gamma, Random Oversampling, Random Undersampling, NearMiss1, NearMiss3, Distant, stratified sampling, GAN-based sampling) and other techniques (model reweighting and loss function constraint) are not well equipped to reduce bias among demographic subpopulations. So, we proposed a double prioritized (DP) [40] sampling technique that incrementally increases the minority subgroups in the minority class so which helps to reduce bias. We compare our proposed DP technique with sampling techniques and show that the DP technique indeed reduces disparity than other sampling methods. DP bias correction is applicable to medical datasets, where data imbalance may be a source of accuracy disparity. The method is not designed to address non-representational disparities, e.g., underdiagnosis and measurement bias.

5.2 Future Work

In software security, an interesting future research direction can be motivating the research of cryptographic misuse detection tools for other platforms, we plan to extend CryptoAPI-Bench to cover other popular languages, e.g., Python. Also, the addition of other interesting cases can extend the benchmark, for example, other non-cryptographic API misuses (e.g., Android APIs to access sensitive information (location, IMEI, passwords, etc.) [62, 112], fingerprint protection [58], cloud service APIs for information storage [143]) are also proven to cause catastrophic security consequences.

In healthcare prognosis using machine learning models, some future directions include further enhancing the interpretability of machine learning prognosis models, as well as exploring how data underrepresentation impacts the quality of medical image analysis and mutation-based evolutionary computation [105]. Also, testing or monitoring machine learning models to test their performance on the corner cases using domain knowledge and visualizing the failure cases is another interesting research direction. Another research direction can be obtaining more informed prediction analysis with multimodal models with different types of inputs. For example, more informative multimodal model disparity analysis can be done using chest X-rays and time-series ICU data of patients using MIMIC-CXR [43].

Bibliography

[1] AES Encryption. https://aesencryption.net/. Online; Last accessed: Dec 3, 2020.

[2] Top 10 Apache Projects in 2021, from Superset, to Nuttx and Pulsar. https://thestack.technology/top-apache-projects-in-2021-from-superset-to-nuttx/. Online; Last accessed: April 4, 2020.

[3] Find Security Bugs. https://find-sec-bugs.github.io/. Online; Last accessed: Dec 3, 2020.

[4] Hostname Verification to SSL Socket. https://www.the-codingforums.com/threads/adding-hostname-verification-to-sslsocket.958287/. Online; Last accessed: Dec 3, 2020.

[5] Kruskal–Wallis One-way Analysis of Variance. https://en.wikipedia.org/wiki/Kruskal%E2%80%93Wallis_one-way_analysis_of_variance. Online; Last accessed: Dec 8, 2022.

[6] Seer Incidence Data. https://seer.cancer.gov/data/. Online; Last accessed: March 7, 2022.

[7] URL Spoofing. http://www.securitysupervisor.com/security-q-a/network-security/262-what-is-url-spoofing.html. Online; Last accessed: Dec 3, 2020.

[8] AndroZoo. https://androzoo.uni.lu/. Online; Last accessed: Dec 3, 2020.

[9] A Popular Algorithm Is No Better at Predicting Crimes Than Random People. https://www.theatlantic.com/technology/archive/2018/01/equivant-compas-algorithm/550646/. Online; Last accessed: March 7, 2022.

[10] Critical Application Definition. https://www.lawinsider.com/dictionary/critical-application. Online; Last accessed: November 27, 2022.

[11] CryptoGuard. https://github.com/CryptoGuardOSS/cryptoguard, . Online; Last accessed: Dec 3, 2020.

[12] Cognicrypt_SAST: CrySLtoStatic Analysis Compiler. https://github.com/CROSSINGTUD/CryptoAnalysis, . Online; Last accessed: Dec 3, 2020.

[13] The Third-Leading Cause of Death in US Most Doctors Don't Want You to Know About. https://www.cnbc.com/2018/02/22/medical-errors-third-leading-cause-of-death-in-america.html. Online; Last accessed: November 27, 2022.

[14] Disparate Impact. https://en.wikipedia.org/wiki/Disparate_impact. Online; Last accessed: April 7, 2022.

[15] How America's 5 Top Hospitals are Using Machine Learning Today. https://emerj.com/ai-sector-overviews/top-5-hospitals-using-machine-learning/. Online; Last accessed: March 7, 2022.

[16] GRAMMATECH. https://www.grammatech.com/. Online; Last accessed: Dec 3, 2020.

[17] Command Center to Improve Patient Flow. https://www.hopkinsmedicine.org/news/articles/command-center-to-improve-patient-flow. Online; Last accessed: March 7, 2022.

[18] Cleveland Clinic to Identify At-Risk Patients in ICU using Cortana Intelligence. https://docs.microsoft.com/en-us/archive/blogs/machinelearning/cleveland-clinic-to-identify-at-risk-patients-in-icu-using-cortana-intelligence-suite. Online; Last accessed: March 7, 2022.

[19] NIST Computer Security Resource Center. https://csrc-.nist.gov/projects. Online; Last accessed: Sep 3, 2021.

[20] 2017 Profile of Older Americans. https://acl.gov/sites/default/files/Aging%20and%20Disability%20in%20America/2017OlderAmericansProfile.pdf. Online; Last accessed: November 11, 2022.

[21] Secure Coding Guidelines for Java SE. https://www.oracle.com/java/technologies/javase/seccodeguide.html. Online; Last accessed: Sep 3, 2021.

[22] Machine Bias: Theres Software Used Across the Country to Predict Future Criminals and Its Biased Against Blacks. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. Online; Last accessed: March 7, 2022.

[23] Quick Android Review Kit (QARK). https://github-.com/linkedin/qark. Online; Last accessed: Dec 3, 2020.

[24] Amazon Scraps Secret AI Recruiting Tool That Showed Bias Against Women. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G. Online; Last accessed: March 7, 2022.

[25] Secure Code Review: 8 Security Code Review Best Practices. https://snyk.io/blog/secure-code-review/, . Online; Last accessed: Sep 3, 2021.

[26] Colonial Pipeline Paid Roughly $5 Million in Ransom to Hackers. https://www.nytimes.com/2021/05/13/us/politics/biden-colonial-pipeline-ransomware.html, . Online; Last accessed: November 27, 2022.

[27] Log4j Flaw: Attackers Are Making Thousands of Attempts to Exploit this Severe Vulnerability. https://www.zdnet.com/article/log4j-flaw-attackers-are-making-thousands-of-attempts-to-exploit-this-severe-vulnerability/, . Online; Last accessed: November 27, 2022.

[28] All of JBS's U.S. Beef Plants Were Forced Shut by Cyberattack. https://www.bloomberg.com/news/articles/2021-05-31/meat-is-latest-cyber-victim-as-hackers-hit-top-supplier-jbs, . Online; Last accessed: November 27, 2022.

[29] SonarSource Static Code Analysis. https://rules.sonarsource.com/. Online; Last accessed: Dec 3, 2020.

[30] SpotBugs: Find Bugs in Java Programs. https://spotbugs-.github.io/. Online; Last accessed: Dec 3, 2020.

[31] Oracle. https://docs.oracle.com/javase/8/docs/api/java/secur-ity/Permission.html. Online; Last accessed: Sep 3, 2021.

[32] GANs for Tabular Data. https://pypi.org/project/tabgan/. Online; Last accessed: Oct 25, 2022.

[33] Transition of SWAMP software. https://continuousassurance-.org/blog/. Online; Last accessed: Dec 3, 2020.

[34] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. Peek-a-boo: I See Your Smart Home Activities, Even Encrypted! In *Proceedings of the 13th ACM*

Conference on Security and Privacy in Wireless and Mobile Networks, pages 207–218, 2020.

[35] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the Usability of Cryptographic APIs. In *IEEE Symposium on Security and Privacy, SP'17, San Jose, CA, USA, May 22-26*, pages 154–171, 2017.

[36] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *IEEE Symposium on Security and Privacy, SP'16, San Jose, CA, USA, May 23-25*, pages 289–305, 2016.

[37] S. Afrose. CryptoAPI-Bench. `https://github.com/CryptoAPI-Bench/CryptoAPI-Bench`, 2019.

[38] S. Afrose. ApacheCryptoAPI-Bench. `https://github.com/CryptoAPI-Bench/ApacheCryptoAPI-Bench`, 2020.

[39] Sharmin Afrose, Sazzadur Rahaman, and Danfeng Yao. CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 49–61. IEEE, 2019.

[40] Sharmin Afrose, Wenjia Song, Charles B Nemeroff, Chang Lu, and Danfeng Daphne Yao. Subpopulation-specific Machine Learning Prognosis for Underrepresented Patients with Double Prioritized Bias Correction. *Communications Medicine*, 2(1):1–14, 2022.

[41] Sharmin Afrose, Ya Xiao, Sazzadur Rahaman, Barton Miller, and Danfeng Daphne Yao. Evaluation of Static Vulnerability Detection Tools with Java Cryptographic API Benchmarks. *IEEE Transactions on Software Engineering*, 2022.

[42] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. Black Box Fairness Testing of Machine Learning Models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 625–635, 2019.

[43] Roger Mark Seth Berkowitz Alistair Johnson, Tom Pollard and Steven Horng. MIMIC-CXR database. In *PhysioNet*, 2019. doi: https://doi.org/10.13026/C2JT1Q.

[44] Amit Seal Ami, Nathan Cooper, Kaushal Kafle, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni. Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques. *arXiv preprint arXiv:2107.07065*, 2021.

[45] Jing An, Lexing Ying, and Yuhua Zhu. Why Resampling Outperforms Reweighting for Correcting Sampling Bias with Stochastic Gradients. *International Conference on Learning Representations*, 2021.

[46] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick D. McDaniel. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'14*, pages 259–269, 2014.

[47] Ken Ashcraft and Dawson R. Engler. Using Programmer-Written Compiler Extensions to Catch Security Holes. In *IEEE Symposium on Security and Privacy, (SP'02)*, pages 143–159, 2002.

[48] Insaf Ashrapov. Tabular GANs for Uneven Distribution. *arXiv preprint arXiv:2010.00638*, 2020.

[49] Hala Assal and Sonia Chiasson. Security in the Software Development Lifecycle. In

Fourteenth Symposium on Usable Privacy and Security, SOUPS'18, pages 281–296, 2018.

[50] João B Augusto, Rhodri H Davies, Anish N Bhuva, Kristopher D Knott, Andreas Seraphim, Mashael Alfarih, Clement Lau, Rebecca K Hughes, Luís R Lopes, Hunain Shiwani, et al. Diagnosis and Risk Stratification in Hypertrophic Cardiomyopathy using Machine Learning Wall Thickness Measurement: A Comparison with Human Test-retest Performance. *The Lancet Digital Health*, 3(1):e20–e28, 2021.

[51] Aya Awad, Mohamed Bader-El-Den, James McNicholas, and Jim Briggs. Early Hospital Mortality Prediction of Intensive Care Unit Patients Using an Ensemble Learning Approach. *International Journal of Medical Informatics*, 108:185–195, 2017.

[52] Leonardo Babun, Kyle Denney, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. A Survey on IoT Platforms: Communication, Security, and Privacy Perspectives. *Computer Networks*, 192:108040, 2021.

[53] Elaine Barker and Allen Roginsky. Transitioning the Use of Cryptographic Algorithms and Key Lengths. In *Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD*, 2019. doi: https://doi.org/10.6028/NIST.SP.800-131Ar2.

[54] Elaine Barker, Lidong Chen, Allen Roginsky, Richard Davis, and Scott Simon. Recommendation for Pair-wise Key Establishment Using Integer Factorization Cryptography. In *Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD*, 2019. doi: https://doi.org/10.6028/NIST.SP.800-56Br2.

[55] Christopher Andrew Barton, Graham Andrew Clarke, and Simon Crowe. Transferring Data via a Secure Network Connection, 2006. US Patent 7,093,121.

[56] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen AWM Van Der Laak, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, et al. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. *Jama*, 318(22):2199–2210, 2017.

[57] Mihir Bellare. New Proofs for NMAC and HMAC: Security Without Collision Resistance. *Journal of Cryptology*, 28(4):844–878, 2015.

[58] Antonio Bianchi, Yanick Fratantonio, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, Simon Pak Ho Chung, and Wenke Lee. Broken Fingers: On the Usage of the Fingerprint API in Android. In *25th Annual Network and Distributed System Security Symposium, NDSS'18*, 2018.

[59] Stephen M. Blackburn et al. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. In *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'06, Portland, Oregon, USA*, pages 169–190, 2006.

[60] Miranda Bogen and Aaron Rieke. Help Wanted: An Examination of Hiring Algorithms, Equity, and Bias. 2018.

[61] Ashish Bora, Siva Balasubramanian, Boris Babenko, Sunny Virmani, Subhashini Venugopalan, Akinori Mitani, Guilherme de Oliveira Marinho, Jorge Cuadros, Paisan Ruamviboonsuk, Greg S Corrado, et al. Predicting the Risk of Developing Diabetic Retinopathy using Deep Learning. *The Lancet Digital Health*, 3(1):e10–e19, 2021.

[62] Amiangshu Bosu, Fang Liu, Danfeng(Daphne) Yao, and Gang Wang. Collusive Data Leak and More: Large-Scale Threat Analysis of Inter-app Communications. In *ACM*

ASIA Conference on Computer and Communications Security, AsiaCCS'17, pages 71–85, 2017.

[63] Joy Buolamwini and Timnit Gebru. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91. PMLR, 2018.

[64] Elisa Burato, Pietro Ferrara, and Fausto Spoto. Security Analysis of the OWASP Benchmark with Julia. *The Italian Conference on CyberSecurity (ITASEC)*, 17, 2017.

[65] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[66] Cancan Chen, Shan Zheng, Lei Guo, Xuebing Yang, Yan Song, Zhuo Li, Yanwu Zhu, Xiaoqi Liu, Qingzhuang Li, Huijuan Zhang, et al. Identification of Misdiagnosis by Deep Neural Networks on a Histopathologic Review of Breast Cancer Lymph Node Metastases. *Scientific reports*, 12(1):1–10, 2022.

[67] Jesse Davis and Mark Goadrich. The Relationship Between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006.

[68] DHS SWAMP. Welcome to the SWAMP. https://continuousassurance.org, 2018.

[69] Julia Dressel and Hany Farid. The Accuracy, Fairness, and Limits of Predicting Recidivism. *Science advances*, 4(1):eaao5580, 2018.

[70] Chris Drummond and Robert C Holte. Explicitly Representing Expected Cost: An Alternative to ROC Representation. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 198–207, 2000.

[71] Chris Drummond and Robert C Holte. What ROC Curves Can't Do (and Cost Curves Can). In *ROCAI*, pages 19–26. Citeseer, 2004.

[72] Rashmi Dubey, Jiayu Zhou, Yalin Wang, Paul M Thompson, Jieping Ye, Alzheimer's Disease Neuroimaging Initiative, et al. Analysis of Sampling Techniques for Imbalanced Data: An n= 648 ADNI Study. *NeuroImage*, 87:220–241, 2014.

[73] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An Empirical Study of Cryptographic Misuse in Android Applications. In *ACM Conference on Computer and Communications Security, CCS'13*, pages 73–84, 2013.

[74] Yaniv Eytani, Klaus Havelund, Scott D Stoller, and Shmuel Ur. Towards a Framework and a Benchmark for Testing Tools for Multi-Threaded Programs. *Concurrency and Computation: Practice and Experience*, 19(3):267–279, 2007.

[75] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, Lars Baumgärtner, and Bernd Freisleben. Why Eve and Mallory Love Android: An Analysis of Android SSL (in) Security. In *the ACM Conference on Computer and Communications Security, CCS'12*, pages 50–61, 2012.

[76] François Gagnon, Marc-Antoine Ferland, Marc-Antoine Fortier, Simon Desloges, Jonathan Ouellet, and Catherine Boileau. AndroSSL: A Platform to Test Android Applications Connection Security. In *International Symposium on Foundations and Practice of Security, FPS'15*, pages 294–302, 2015.

[77] Isaac R Galatzer-Levy, Karen-Inge Karstoft, Alexander Statnikov, and Arieh Y Shalev. Quantitative Forecasting of PTSD from Early Trauma Responses: A Machine Learning Application. *Journal of psychiatric research*, 59:68–76, 2014.

[78] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness Testing: Testing Software for Discrimination. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, pages 498–510, 2017.

[79] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software. In *the ACM Conference on Computer and Communications Security, CCS'12*, pages 38–49, 2012.

[80] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, 2020.

[81] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multitask Learning and Benchmarking with Clinical Time Series Data. *Scientific data*, 6(1):1–18, 2019.

[82] Klaus Havelund, Scott D Stoller, and Shmuel Ur. Benchmark and Framework for Encouraging Research on Multi-Threaded Testing Tools. In *Proceedings International Parallel and Distributed Processing Symposium, IPDPS'03*, page 286. IEEE, 2003.

[83] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008.

[84] Stefan Hegselmann, Leonard Gruelich, Julian Varghese, and Martin Dugas. Reproducible Survival Prediction with SEER Cancer Data. In *Machine Learning for Healthcare Conference*, pages 49–66. PMLR, 2018.

[85] Catherine Hercus and Abdul-Rahman Hudaib. Delirium Misdiagnosis Risk in Psychiatry: A Machine Learning-Logistic Regression Predictive Algorithm. *BMC health services research*, 20(1):1–7, 2020.

[86] Mazharul Islam, Sazzadur Rahaman, Na Meng, Behnaz Hassanshahi, Padmanabhan Krishnan, and Danfeng Daphne Yao. Coding Practices and Recommendations of Spring Security for Enterprise Applications. In *2020 IEEE Secure Development (SecDev)*, pages 49–57. IEEE, 2020.

[87] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, A Freely Accessible Critical Care Database. *Scientific data*, 3(1): 1–9, 2016.

[88] Alistair EW Johnson, Tom J Pollard, and Roger G Mark. Reproducibility in Critical Care: A Mortality Prediction Case Study. In *Machine Learning for Healthcare Conference*, pages 361–376. PMLR, 2017.

[89] Justin M Johnson and Taghi M Khoshgoftaar. Survey on Deep Learning with Class Imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[90] Nidhi Kalra and Susan M Paddock. Driving to Safety: How Many Miles of Driving Would it Take to Demonstrate Autonomous Vehicle Reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[91] Firuz Kamalov and Dmitry Denisov. Gamma Distribution-based Sampling for Imbalanced Data. *Knowledge-Based Systems*, 207:106368, 2020.

[92] Martin Kellogg, Martin Schäf, Serdar Tasirans, and Michael D. Ernst. Continuous

Compliance. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 511–523, 2020.

[93] Donald E Knuth. *Art of Computer Programming, volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, 2014.

[94] Michael R Kosorok and Eric B Laber. Precision medicine. *Annual review of statistics and its application*, 6:263–286, 2019.

[95] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs. In *European Conference on Object-Oriented Programming, ECOOP'18*, pages 10:1–10:27, 2018.

[96] Stefan Krüger et al. CogniCrypt: Supporting Developers in using Cryptography. In *IEEE/ACM International Conference on Automated Software Engineering, ASE'17*, pages 931–936, 2017.

[97] Claire Le Goues, Neal Holtschulte, Edward K. Smith, Yuriy Brun, Premkumar T. Devanbu, Stephanie Forrest, and Westley Weimer. The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs. *IEEE Transactions on Software Engineering*, 41(12):1236–1256, 2015.

[98] Sang Bum Lee, Jae Hun Oh, Jeong Ho Park, Seung Pill Choi, and Jung Hee Wee. Differences in Youngest-old, Middle-old, and Oldest-old Patients who Visit the Emergency Department. *Clinical and Experimental Emergency Medicine*, 5(4):249, 2018.

[99] Li Li, Jun Gao, Médéric Hurier, Pingfan Kong, Tegawendé F. Bissyandé, Alexandre Bartel, Jacques Klein, and Yves Le Traon. AndroZoo++: Collecting Millions

of Android Apps and Their Metadata for the Research Community. *arXiv preprint arXiv:1709.05281*, 2017.

[100] Shan Lu, Zhenmin Li, Feng Qin, Lin Tan, Pin Zhou, and Yuanyuan Zhou. BugBench: Benchmarks for Evaluating Bug Detection Tools. In *Workshop on the Evaluation of Software Defect Detection Tools*, volume 5, 2005.

[101] Aravind Machiry, Chad Spensky, Jake Corina, Nick Stephens, Christopher Kruegel, and Giovanni Vigna. DR. CHECKER: A Soundy Analysis for Linux Kernel Drivers. In *26th USENIX Security Symposium, USENIX Security'17, Vancouver, BC, Canada*, pages 1007–1024, 2017.

[102] Amita Malav, Kalyani Kadam, and Pooja Kamat. Prediction of Heart Disease using K-means and Artificial Neural Network as Hybrid Approach to Improve Accuracy. *International Journal of Engineering and Technology*, 9(4):3081–3085, 2017.

[103] Inderjeet Mani and I Zhang. kNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. In *Proceedings of Workshop on Learning from Imbalanced Datasets*, volume 126, pages 1–7. ICML, 2003.

[104] Na Meng, Stefan Nagy, Daphne Yao, Wenjie Zhuang, and Gustavo Arango Argoty. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *International Conference on Software Engineering, ICSE'18*, May 2018.

[105] Risto Miikkulainen and Stephanie Forrest. A biological Perspective on Evolutionary Computation. *Nature Machine Intelligence*, 3(1):9–15, 2021.

[106] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model Cards

for Model Reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.

[107] Joydeep Mitra and Venkatesh-Prasad Ranganath. Ghera: A Repository of Android App Vulnerability Benchmarks. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE'17, Toronto, Canada*, pages 43–52, 2017.

[108] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS #5: Password-Based Cryptography Specification Version 2.1. 2017.

[109] Amitabha Mukerjee, Rita Biswas, Kalyanmoy Deb, and Amrit P Mathur. Multi-objective Evolutionary Algorithms for the Risk-Return Trade-off in Bank Loan Management. *International Transactions in operational research*, 9(5):583–597, 2002.

[110] Pritam Mukherjee, Mu Zhou, Edward Lee, Anne Schicht, Yoganand Balagurunathan, Sandy Napel, Robert Gillies, Simon Wong, Alexander Thieme, Ann Leung, et al. A Shallow Convolutional Neural Network Predicts Prognosis of Lung Cancer Patients in Multi-institutional Computed Tomography Image Datasets. *Nature machine intelligence*, 2(5):274–282, 2020.

[111] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs? In *International Conference on Software Engineering, ICSE'16*, pages 935–946, 2016.

[112] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. Finding Clues for Your Secrets: Semantics-Driven, Learning-Based Privacy Discovery in Mobile Apps. In *25th Annual Network and Distributed System Security Symposium, NDSS'18*, 2018.

[113] Duc Cuong Nguyen et al. A Stitch in Time: Supporting Android Developers in Writing Secure Code. In *ACM Conference on Computer and Communications Security, CCS'17*, pages 1065–1077, 2017.

[114] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting Racial Bias in an Algorithm Used to Manage the Health of Populations. *Science*, 366 (6464):447–453, 2019.

[115] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *IEEE Symposium on Security and Privacy, SP'18*, pages 634–647, 2018.

[116] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero, Ben Glocker, and Daniel Rueckert. Disease Prediction using Graph Convolutional Networks: Application to Autism Spectrum Disorder and Alzheimer's Disease. *Medical image analysis*, 48:117–130, 2018.

[117] Van L Parsons. Stratified Sampling. *Wiley StatsRef: Statistics Reference Online*, pages 1–11, 2014.

[118] Luca Piccolboni, Giuseppe Di Guglielmo, Luca P Carloni, and Simha Sethumadhavan. Crylogger: Detecting Crypto Misuses Dynamically. *arXiv preprint arXiv:2007.01061*, 2020.

[119] Emma Pierson, David M Cutler, Jure Leskovec, Sendhil Mullainathan, and Ziad Obermeyer. An Algorithmic Approach to Reducing Unexplained Pain Disparities in Underserved Populations. *Nature Medicine*, 27(1):136–140, 2021.

[120] Giulia Pullano, Eugenio Valdano, Nicola Scarpa, Stefania Rubrichi, and Vittoria Colizza. Evaluating the Effect of Demographic Factors, Socioeconomic Factors, and Risk Aversion on Mobility During the COVID-19 Epidemic in France Under Lockdown: a Population-based Study. *The Lancet Digital Health*, 2(12):e638–e649, 2020.

[121] Sazzadur Rahaman and Danfeng Yao. Program Analysis of Cryptographic Implementations for Security. In *IEEE Cybersecurity Development, SecDev'17, Cambridge, MA, USA*, pages 61–68, 2017. doi: 10.1109/SecDev.2017.23. URL https://doi.org/10.1109/SecDev.2017.23.

[122] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng(Daphne) Yao. CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects. In *ACM Conference on Computer and communications security, CCS'19*, pages 2455–2472, Nov. 2019.

[123] Lars Lau Raket, Jörn Jaskolowski, Bruce J Kinon, Jens Christian Brasen, Linus Jönsson, Allan Wehnert, and Paolo Fusar-Poli. Dynamic Electronic Health Record Detection (DETECT) of Individuals at Risk of a First Episode of Psychosis: a Case-Control Development and Validation Study. *The Lancet Digital Health*, 2(5):e229–e239, 2020.

[124] Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. Are Red Roses Red? Evaluating Consistency of Question-Answering Models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6174–6184, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1621. URL https://aclanthology.org/P19-1621.

[125] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Be-

yond Accuracy: Behavioral Testing of NLP Models with Checklist. *arXiv preprint arXiv:2005.04118*, 2020.

[126] David Rojas-Rueda, Mark J Nieuwenhuijsen, Haneen Khreis, and Howard Frumkin. Autonomous Vehicles and Public Health. *Annual review of public health*, 41(1):329–345, 2020.

[127] Takaya Saito and Marc Rehmsmeier. The Precision-Recall Plot Is More Informative Than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PloS One*, 10(3):e0118432, 2015.

[128] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Resampling or Reweighting: A Comparison of Boosting Implementations. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 445–451. IEEE, 2008.

[129] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. Aegis: A Context-aware Security Framework for Smart Home Systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 28–41, 2019.

[130] Daniel J Solove and Woodrow Hartzog. *Breached!: Why Data Security Law Fails and How to Improve it*. Oxford University Press, 2022.

[131] David Sounthiraraj, Justin Sahs, Garret Greenwood, Zhiqiang Lin, and Latifur Khan. SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *The Network and Distributed System Security Symposium, NDSS'14*, 2014.

[132] Johannes Späth, Karim Ali, and Eric Bodden. Context-, Flow-, and Field-sensitive Data-flow Analysis Using Synchronized Pushdown Systems. *Proc. ACM Program.*

Lang., 3(POPL), January 2019. doi: 10.1145/3290361. URL https://doi.org/10.1145/3290361.

[133] Latanya Sweeney. Discrimination in Online Ad Delivery. *Communications of the ACM*, 56(5):44–54, 2013.

[134] Stephany Tandy-Connor, Jenna Guiltinan, Kate Krempely, Holly LaDuca, Patrick Reineke, Stephanie Gutierrez, Phillip Gray, and Brigette Tippin Davis. False-positive Results Released by Direct-to-Consumer Genetic Tests Highlight the Importance of Clinical Confirmation Testing for Appropriate Patient Care. *Genetics in Medicine*, 20 (12):1515–1521, 2018.

[135] Kevin Ten Haaf, Jihyoun Jeon, Martin C Tammemägi, Summer S Han, Chung Yin Kong, Sylvia K Plevritis, Eric J Feuer, Harry J de Koning, Ewout W Steyerberg, and Rafael Meza. Risk Prediction Models for Selection of Lung Cancer Screening Candidates: A Retrospective Validation Study. *PLoS medicine*, 14(4):e1002277, 2017.

[136] Philip S Thomas, Bruno Castro da Silva, Andrew G Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing Undesirable Behavior of Intelligent Machines. *Science*, 366(6468):999–1004, 2019.

[137] Jason Van Hulse, Taghi M Khoshgoftaar, and Amri Napolitano. Experimental Perspectives on Learning from Imbalanced Data. In *Proceedings of the 24th international conference on Machine learning*, pages 935–942, 2007.

[138] Nicholas J White, James A Watson, Sophie Uyoga, Thomas N Williams, and Kathryn M Maitland. Substantial Misdiagnosis of Severe Malaria in African Children. *The Lancet*, 400(10355):807, 2022.

[139] Wikipedia contributors. Data-flow Analysis — Wikipedia, The Free Encyclopedia.

URL https://en.wikipedia.org/wiki/Data-flow_analysis. Last accessed: Sep 9, 2021.

[140] Jack Wilkinson, Kellyn F Arnold, Eleanor J Murray, Maarten van Smeden, Kareem Carr, Rachel Sippy, Marc de Kamps, Andrew Beam, Stefan Konigorski, Christoph Lippert, et al. Time to Reality Check the Promises of Machine Learning-powered Precision Medicine. *The Lancet Digital Health*, 2(12):e677–e680, 2020.

[141] Danfeng Daphne Yao, Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Miles Frantz, Ke Tian, Na Meng, Cristina Cifuentes, Yang Zhao, Nicholas Allen, et al. Being the Developers' Friend: Our Experience Developing a High-Precision Tool for Secure Coding. *IEEE Security & Privacy*, (01):2–11, 2022.

[142] William Yuan, Brett K Beaulieu-Jones, Kun-Hsing Yu, Scott L Lipnick, Nathan Palmer, Joseph Loscalzo, Tianxi Cai, and Isaac S Kohane. Temporal Bias in Case-control Design: Preventing Reliable Predictions of the Future. *Nature communications*, 12(1):1–10, 2021.

[143] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps. In *IEEE Symposium on Security and Privacy, (SP'19), London, UK*, 2019.

Appendices

Appendix A

MIMIC Appendix

A.1 Supplementary Equations

- BCS Class 1: Patient does not survive more than 5 years after breast cancer diagnosis
- IHM Class 1: Based on the first 48 hours of ICU information, the patient dies in ICU
- LCS Class 1: Patient survives more than 5 years after lung cancer diagnosis
- Decomp Class 1: Patient's health deteriorates after 24 hours

$$Recall\, C1\, or\, Sensitivity = \frac{\#Predicted\, True\, Class\, 1}{\#\, True\, Class\, 1} \tag{A.1}$$

$$Recall\, C0\, or\, Specificity = \frac{\#Predicted\, True\, Class\, 0}{\#\, True\, Class\, 0} \tag{A.2}$$

$$Precision\, C1\, or\, Positive\, Predictive\, Value = \frac{\#Predicted\, True\, Class\, 1}{\#\, Predicted\, Class\, 1} \tag{A.3}$$

$$Precision\, C0\, or\, Negative\, Predictive\, Value = \frac{\#Predicted\, True\, Class\, 0}{\#\, Predicted\, Class\, 0} \tag{A.4}$$

$$Accuracy = \frac{\#Predicted\,True\,Class\,1\ +\ \#Predicted\,True\,Class\,0}{\#\,True\,Class\,1\ +\ \#\,True\,Class\,0} \tag{A.5}$$

$$Balanced\,Accuracy = \frac{Recall\,C1\ +\ Recall\,C0}{2} \tag{A.6}$$

$$F1 - Score\,C1 = 2 \times \frac{Precision\,C1\ \times\ Recall\,C1}{Precision\,C1\ +\ Recall\,C1} \tag{A.7}$$

$$F1 - Score\,C0 = 2 \times \frac{Precision\,C0\ \times\ Recall\,C0}{Precision\,C0\ +\ Recall\,C0} \tag{A.8}$$

$$MCC = \frac{\#Predicted\,True\,Class\,1 \times \#Predicted\,True\,Class\,0 - \#Predicted\,False\,Class\,1 \times \#Predicted\,False\,Class\,0}{\sqrt{\#Predicted\,Class\,1 \times \#True\,Class\,1 \times \#Predicted\,Class\,0 \times \#True\,Class\,0}} \tag{A.9}$$

A.2 Relative Frequency Distribution of features

There are 17 clinical features used for in-hospital mortality prediction task. We analyse the relative frequency distribution of each features in two classes (i.e., majority class C0 and minority class C1). Figure 4.4 and Figure A.1 shows all 17 features distribution using relative frequency metric.

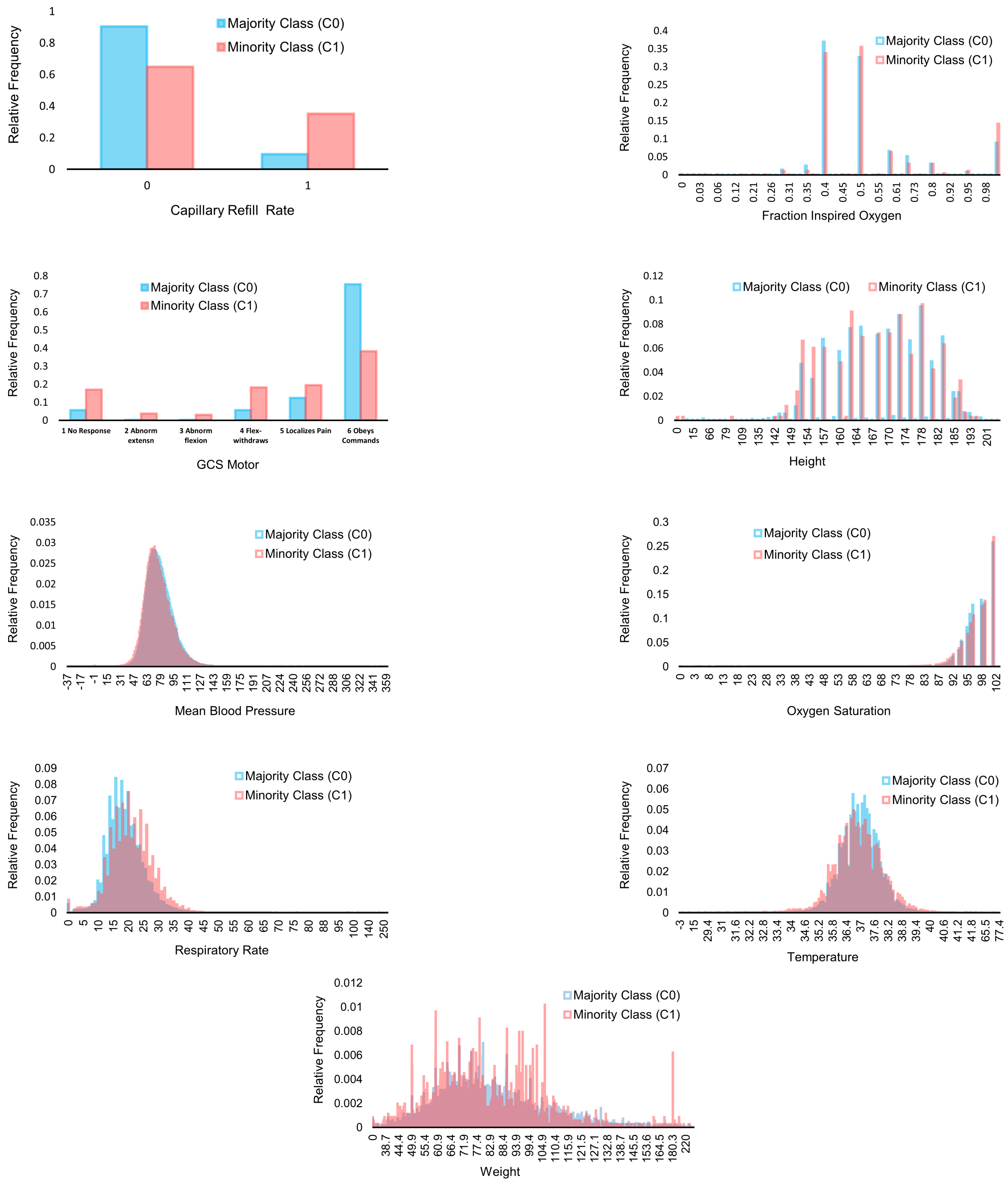


Figure A.1: Relative frequency of several features in MIMIC dataset for in-hospital mortality tasks

A.3 Kruskal Wallis Test

In Table A.1, We p-value where we compare DP with other models based on black subgroup performances on in-hospital mortality prediction task using the Kruskal Wallis test.

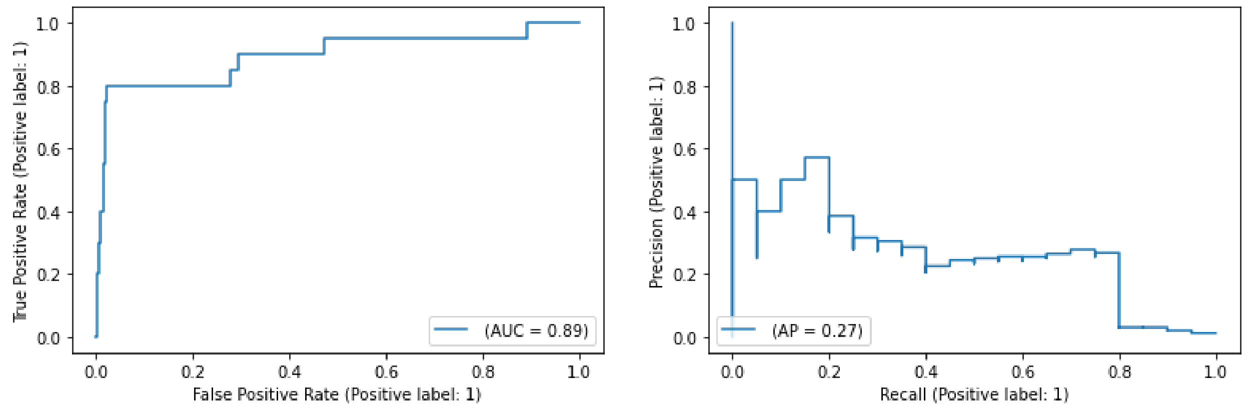
Table A.1: P-value of pairwise comparison with DP with other models using Kruskal Wallis Test in terms of minority class Recall, balanced accuracy, minority class F1 score, minority class PR curve value.

DP Comparison with Other Models	**Recall_C1**	**Bal_Acc**	**F1_C1**	**PR_C1**
Original	**0.049535**	**0.049534613**	0.12663	0.246315
Gamma	**0.049535**	**0.049535**	**0.046302**	**0.043114**
Adasyn	**0.049535**	**0.049535**	**0.049535**	0.824778
Smote	**0.046302**	**0.049535**	0.12663	0.121183
Replicated Oversampling	**0.046302**	**0.049535**	0.275234	0.121183
Random Oversampling	**0.046302**	**0.049535**	**0.049535**	**0.046302**
NearMiss1	**0.049535**	**0.049535**	**0.049535**	**0.046302**
NearMiss3	**0.049535**	**0.049535**	**0.049535**	**0.046302**
Distant	**0.049535**	**0.049535**	**0.046302**	**0.043114**
Stratified RUS	**0.046302**	**0.049535**	**0.049535**	0.121183
Stratified ROS	**0.046302**	**0.046302**	**0.046302**	**0.043114**
Tabular GAN	**0.049535**	**0.049535**	**0.049535**	0.121183

A.4 Deceptive Metrics of AUC for Imbalanced Data

The ROC curve access overall classification performance. The ROC curve plot the false positive rate vs. the true positive rate. If the dataset is skewed, $FPR = FP/(FP + TN)$ will be very small due to a higher TN value. Therefore, the effect of a false positive value will not impact much on the overall performance. In the precision-recall curve (PR) value, the precision metric is used instead of FPR. $Precision = TP/(TP+FP)$ directly influenced by class imblanced situation. Therefore, while evaluating a model built on the imbalanced situation, precision-recall Curve (PR) metrics are better than the ROC curve value.

A random classifier output with an imbalanced dataset (30 samples from Class 1 and 30,000 samples from Class 0) is shown in Figure A.2. The AUC value is 0.89 whereas the PR_C1 value is 0.27. Therefore, the PR_C1 metrics correctly indicate that the classifier model is not a good one.



(a) AUC (b) PRC

Figure A.2: Performance metrics performance under imbalanced situation

A.5 Variations of Loss functions

We applied different loss functions including binary cross-entropy loss, Poisson loss, log-cosh loss, KL divergence loss, and hinge loss. However, the hinge loss and KL divergence loss function can not classify class 1 and class 0 properly. More specifically, they fail to classify class 0 and predict every sample as class 1. The binary cross-entropy loss, Poisson loss, and log-cosh loss show comparable performance as shown in Figure A.3. The equations of binary cross-entropy loss, log-cosh loss, and Poisson loss are shown in Equation A.10, A.11 and A.12 respectively. We use binary cross-entropy loss for all other experiments.

$$Binary\ Cross\text{-}Entropy\ Loss = -\frac{1}{n}\sum_{i=1}^{n} y_i \ . \ log\ \hat{y}_i + (1-y_i)\ .\ log\ (1-\hat{y}_i) \tag{A.10}$$

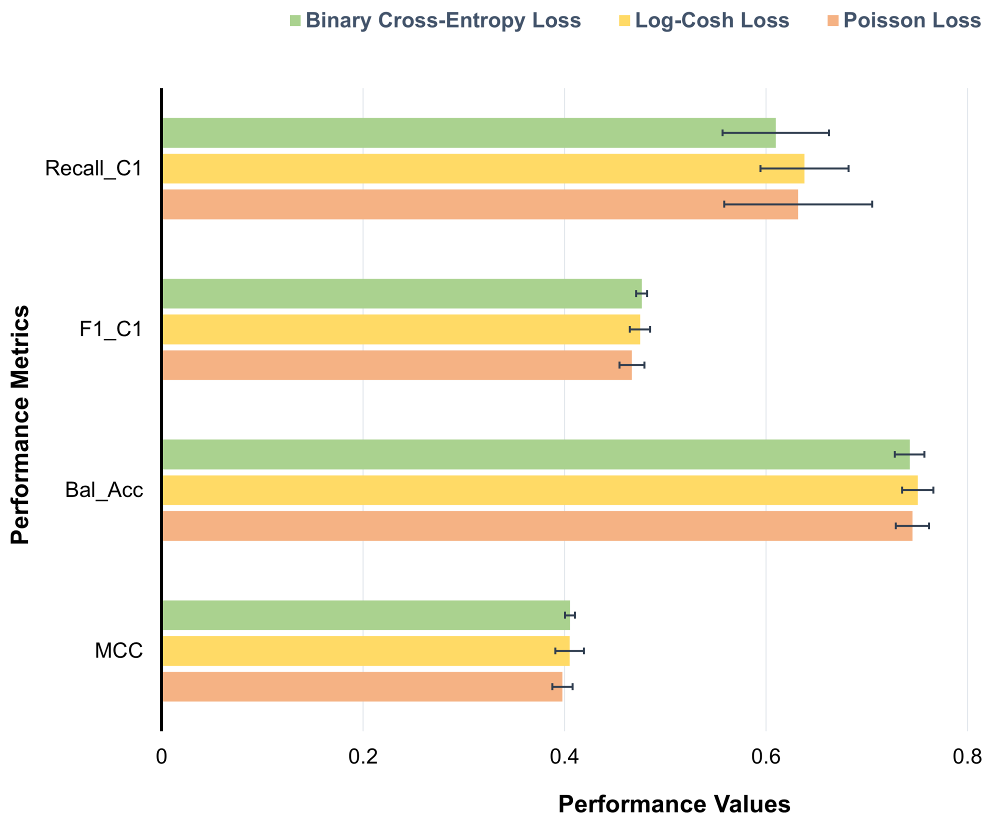
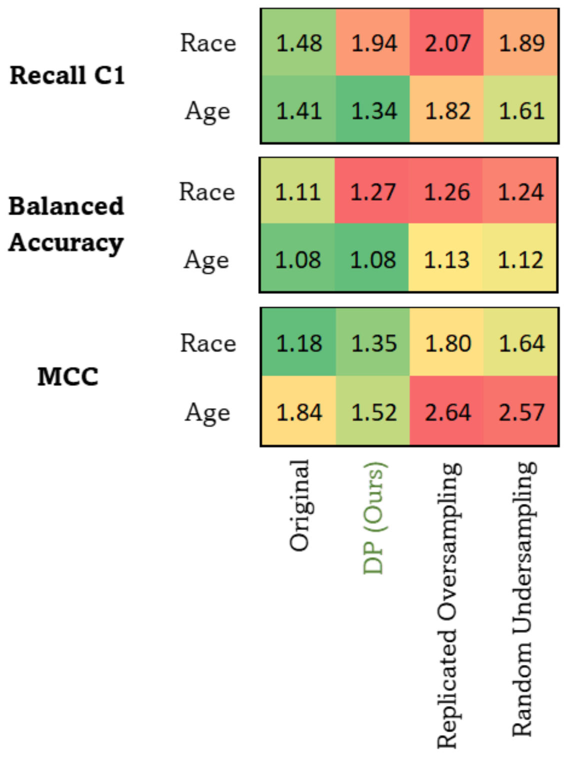


Figure A.3: Performance of different loss functions on the original model of in-hospital mortality tasks

$$Log\text{-}Cosh\ Loss = \sum_{i=1}^{n} log(cosh(\hat{y}_i - y_i)) \tag{A.11}$$

$$Poisson\ Loss = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i\ .\ log\ \hat{y}_i) \tag{A.12}$$



		Original	DP (Ours)	Replicated Oversampling	Random Undersampling
Recall C1	Race	1.48	1.94	2.07	1.89
	Age	1.41	1.34	1.82	1.61
Balanced Accuracy	Race	1.11	1.27	1.26	1.24
	Age	1.08	1.08	1.13	1.12
MCC	Race	1.18	1.35	1.80	1.64
	Age	1.84	1.52	2.64	2.57

Figure A.4: Relative disparity among racial and age groups under various sampling conditions for decomp prediction

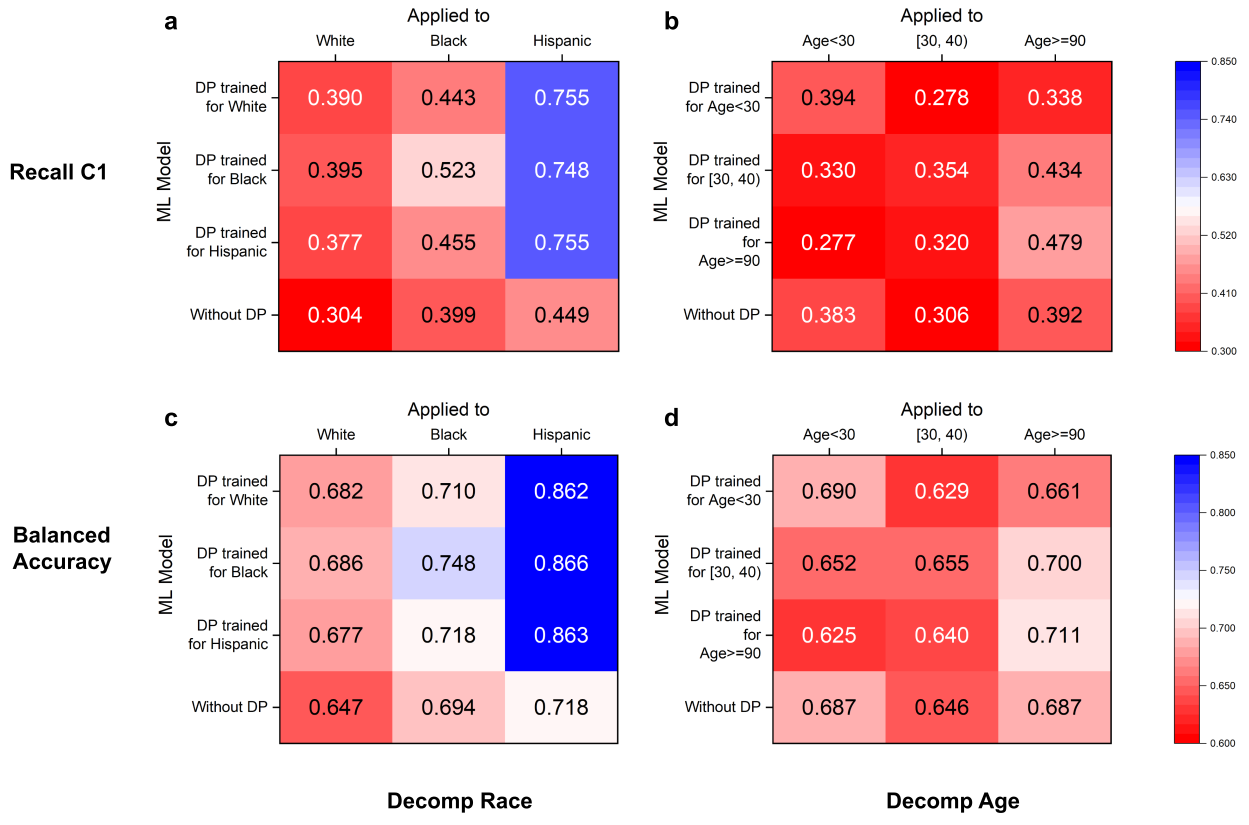


Figure A.5: DP's cross-group performance under various race and age settings for recall C1 and balanced accuracy for the decompensation prediction. In subfigures, each row corresponds to a DP model trained for a specific subgroup. Each column represents a subgroup that a model is evaluated on. The values on the diagonal are the performance of a matching DP model, i.e., a DP model applied to the subgroup that it is designed for. The last rows show the group's performance in the original model. To prevent overfitting, our method chooses optimal thresholds based on whole group performance, as opposed to the (small) minority groups in the validation sets. DP cross-group performance for (a) race subgroups and (b) age subgroups for the decompensation prediction in terms of recall C1. DP cross-group performance for (c) race subgroups and (d) age subgroups for the decompensation prediction in terms of balanced accuracy

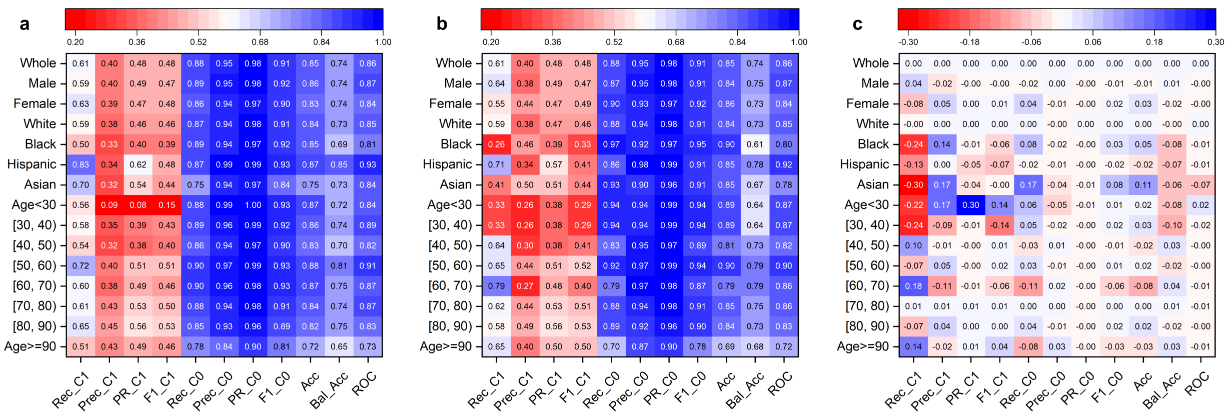
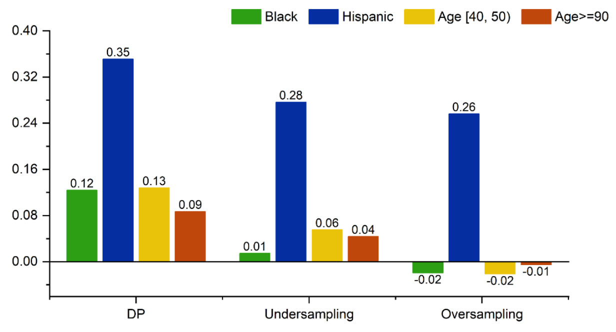
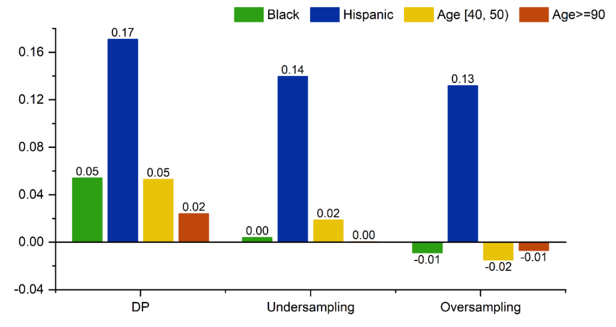


Figure A.6: In-hospital mortality prediction performance of the original model with (a) whole group calibration, (b) subgroup calibration, and (c) difference in the performance between the whole group and subgroup calibration. A positive value means subgroup calibration improves performance. Rec_C1, Prec_C1, PR_C1, F1_C1, Rec_C0, Prec_C0, PR_C0, F1_C0, Acc, Bal_Acc, ROC stand for Recall Class 1, Precision Class 1, Area Under the Precision-Recall Curve Class 1, F1 score Class 1, Recall Class 0, Precision Class 0, Area Under the Precision-Recall Curve Class 0, F1 score Class 0, Accuracy, Balanced Accuracy, Area under the ROC Curve, respectively.

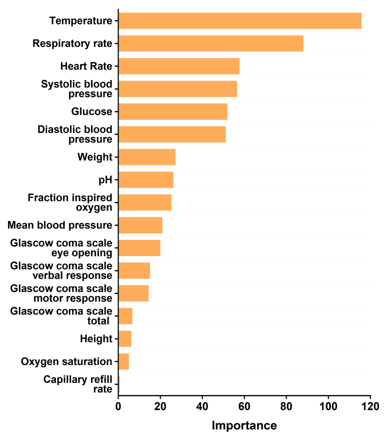


(a) Difference in Recall C1

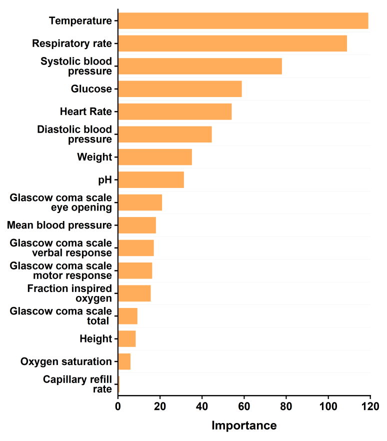


(b) Difference in Balanced Accuracy

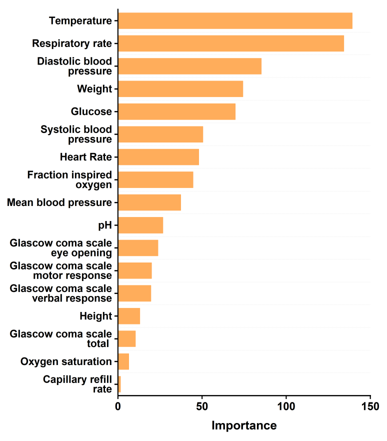
Figure A.7: DP and two representative sampling techniques performance comparison over the original model for decomp prediction



(a) Original Model



(b) DP model for Black



(c) DP model for Age 90+

Figure A.8: SHAP-avg feature importance of different IHM experiments. Original stands for the original machine learning model without any bias correction. DP stands for our Double Prioritized sampling method. In SHAP-avg, the importance of columns representing the same variable is averaged.