8-2017

Dynamic Adversarial Mining - Effectively applying machine learning in adversarial non-stationary environments

Tegjyot Singh Sethi

DYNAMIC ADVERSARIAL MINING - EFFECTIVELY APPLYING MACHINE LEARNING IN ADVERSARIAL NON-STATIONARY ENVIRONMENTS

By

Tegjyot Singh Sethi
M.S., University of Louisville, USA, 2013
B.Tech., GITAM University, India, 2012

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in Computer Science and Engineering

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

August 2017

DYNAMIC ADVERSARIAL MINING - EFFECTIVELY APPLYING MACHINE LEARNING IN ADVERSARIAL NON-STATIONARY ENVIRONMENTS

By

Tegjyot Singh Sethi
M.S., University of Louisville, USA, 2013
B.Tech., GITAM University, India, 2012

A Dissertation Approved On

June 19, 2017

by the following Dissertation Committee:

Mehmed Kantardzic, Ph.D., Dissertation Director

Adel Elmaghraby, Ph.D.

Roman Yampolskiy, Ph.D.

Adrian Lauf, Ph.D.

Jeff Hieb, Ph.D.

DEDICATION

To Daduji and Dadiji, for always believing in me, and to my parents, for their unconditional love and sacrifices.

ACKNOWLEDGEMENTS

The completion of this dissertation has been a humbling and enriching experience. It was made possible by the support and encouragement of several people.

I would like to thank my mentor Dr. Mehmed Kantardzic, who has always motivated and encouraged me to complete my dissertation. He has provided me with valuable guidance and support, and has always made sure that I have a well rounded graduate school experience. He encouraged me to be an active member of the larger scientific community, and pushed me to explore new skills, which I never knew I could possess. Without his encouragement, I wouldn't have come this far in my research endeavors.

I would like to thank my committee members- Dr. Adel Elmaghraby, Dr. Roman Yampolskiy, Dr. Adrian Lauf and Dr. Jeff Hieb, for their guidance and feedback, in writing this dissertation. I am also thankful to Dr. Mahendra Sunkara, for introducing me to the department and for motivating me to pursue my doctoral education.

The love and support of my family and friends, has been an immense factor in the completion of this dissertation. My parents, Manmohan Singh and Gurinder Sethi, and my sister Ankita, have always been a bastion of support. Rachita Singh and Devinder Singh, my uncle and aunt, Jia and Angad, my cousins, have kept me cheerful throughout the process and inspired me to achieve this goal.

I am thankful to the University of Louisville and the Department of Computer Engineering and Computer Science, for their continued support and exposure to opportunities. I would also like to thank the members of the Data Mining Lab, for all the brainstorming sessions. Lastly, I would like to thank Kaldi for discovering Coffee, without which this dissertation would not have been possible.

ABSTRACT

DYNAMIC ADVERSARIAL MINING - EFFECTIVELY APPLYING MACHINE LEARNING IN ADVERSARIAL NON-STATIONARY ENVIRONMENTS

Tegjyot Singh Sethi

June 19, 2017

While understanding of machine learning and data mining is still in its budding stages, the engineering applications of the same has found immense acceptance and success. Cybersecurity applications such as intrusion detection systems, spam filtering, and CAPTCHA authentication, have all begun adopting machine learning as a viable technique to deal with large scale adversarial activity. However, the naive usage of machine learning in an adversarial setting is prone to reverse engineering and evasion attacks, as most of these techniques were designed primarily for a static setting. The security domain is a dynamic landscape, with an ongoing never ending arms race between the system designer and the attackers. Any solution designed for such a domain needs to take into account an active adversary and needs to evolve over time, in the face of emerging threats. We term this as the *Dynamic Adversarial Mining* problem, and the presented work provides the foundation for this new interdisciplinary area of research, at the crossroads of Machine Learning, Cybersecurity, and Streaming Data Mining.

We start with a white hat analysis of the vulnerabilities of classification systems to exploratory attack. The proposed *Seed-Explore-Exploit* framework provides characterization and modeling of attacks, ranging from simple random evasion attacks to sophisticated reverse engineering. It is observed that, even systems having prediction accuracy close to

100%, can be easily evaded with more than 90% precision. This evasion can be performed without any information about the underlying classifier, training dataset, or the domain of application.

Attacks on machine learning systems cause the data to exhibit non stationarity (i.e., the training and the testing data have different distributions). It is necessary to detect these changes in distribution, called concept drift, as they could cause the prediction performance of the model to degrade over time. However, the detection cannot overly rely on labeled data to compute performance explicitly and monitor a drop, as labeling is expensive and time consuming, and at times may not be a possibility altogether. As such, we propose the *Margin Density Drift Detection (MD3)* algorithm, which can reliably detect concept drift from unlabeled data only. MD3 provides high detection accuracy with a low false alarm rate, making it suitable for cybersecurity applications; where excessive false alarms are expensive and can lead to loss of trust in the warning system. Additionally, MD3 is designed as a classifier independent and streaming algorithm for usage in a variety of continuous never-ending learning systems.

We then propose a *Dynamic Adversarial Mining* based learning framework, for learning in non stationary and adversarial environments, which provides 'security by design'. The proposed *Predict-Detect* classifier framework, aims to provide: robustness against attacks, ease of attack detection using unlabeled data, and swift recovery from attacks. Ideas of feature hiding and obfuscation of feature importance are proposed as strategies to enhance the learning framework's security. Metrics for evaluating the dynamic security of a system and recover-ability after an attack are introduced to provide a practical way of measuring efficacy of dynamic security strategies. The framework is developed as a streaming data methodology, capable of continually functioning with limited supervision and effectively responding to adversarial dynamics.

The developed ideas, methodology, algorithms, and experimental analysis, aim to provide a foundation for future work in the area of *Dynamic Adversarial Mining*, wherein a holistic approach to machine learning based security is motivated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ALGORITHM Page

CHAPTER 1

INTRODUCTION

Living in a 'always-on' world, where more than 40% of the population is reachable at the click of a button[1], has provided us with immense economic and social opportunities. Whether it be for accessing the wealth of information available on the web using Google[2], for building an encyclopedia of the human knowledge base using Wikipedia[3], for funding your next business/art project via Kickstarter[4], or for making your voice heard to over half a billion people on Facebook[5]/Twitter[6], the internet has changed the scale at which we humans think and operate. The growing scale and reachability of modern day web applications, has also made it vulnerable to cyberattacks, which threaten to affect the entire globe. While application developers intend to touch the lives of billions by making their services available everywhere, it leaves the arena open to adversarial activity, where the attacks can originate from anywhere, anytime and can take any shape, unseen before. Conventional methods of security mechanisms, such as passwords, firewalls, authentication tools and signature based black and white lists, cannot effectively thwart evolving attacks at this scale [12]. Signature based and rule based systems, summarize existing malicious activities and trigger an alarm when a similar event is repeated [13, 14]. However, their inability to extrapolate from previously seen data about attacks, make them ineffective in case of zero-day exploits and anomalies, which have no precedent analogues. On the other hand, when these traditional security tools do detect a potential breach, they are one of the many alarms raised by such systems. The severe shortage of top tier security professionals,

[1] `http://www.internetlivestats.com/`

[2] `www.google.com`

[3] `www.wikipedia.org`

[4] `www.kickstarter.com`

[5] `www.facebook.com`

[6] `www.twitter.com`

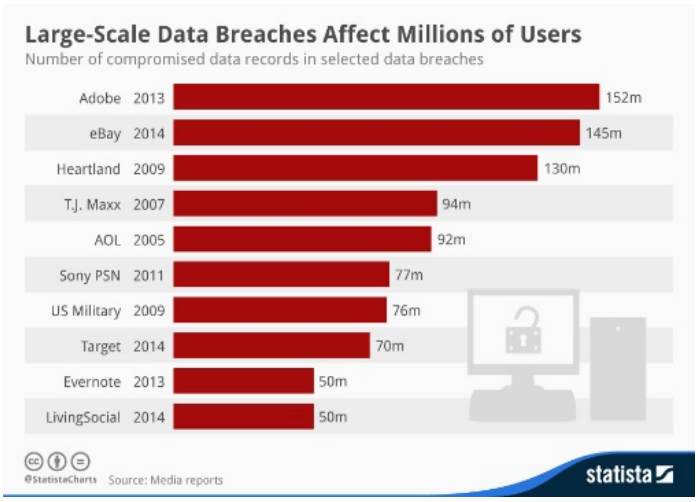


Figure 1.1: Recent large scale data breaches[2].

makes the diligent analysis of each of these alarms an intractable task, often causing attacks to go unnoticed. Current reported statistics show that attackers go unnoticed for more than 200 days and cost businesses over $300B, every single year[1].

The immediate need to learn from observed data has been met by the development of various machine learning techniques [15, 16], which are capable of learning from the known data and generalizing to the unknown realm. The *Big Data* revolution has fueled the development of scalable and practical machine learning system, which has led to its wide spread adaptation and popularity. The domain of cybersecurity has also recognized the need for a data driven solution [17–20], owing to the increased scale and sophistication of attacks in recent times (Figure 1.1[2]). It is impossible to rely solely on human expertise to curate these attacks, as the process is laborious, expensive and is made impractical in current times, owing to the shortage in skilled security analysts[3]. Machine learning systems provide a semi-automated approach, by integrating knowledge learned from previously seen data with budgeted human expertise, to provide a scalable solution with human in the loop.

Use of machine learning has found success in several security applications [17–20]. Anomaly and outlier detection techniques are being used for network intrusion detec-

[1] http://www.cyberark.com/blog/fast-facts-noteworthy-cyber-security-statistics/

[2] http://www.slideshare.net/LucasRivera4/ucsp615narratedpowepointtrendsincybersecurity

[3] http://data-informed.com/cyber-security-skill-shortage-case-machine-learning/

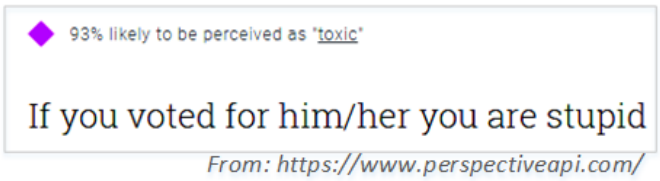


Figure 1.2: Illustration of the Perspective API[1], which leverages machine learning to assess toxicity of online comments, to prevent harassment of users.

tion [21–26]. Supervised techniques of classification are used for Spam filtering [27–30]. Biometric authentication using mouse movement patterns and keystroke dynamics, are also using supervised learning techniques [31]. N-gram analysis of executables is being used for distinguishing malware from regular programs [32], and also for identifying infected pdf documents [33]. Unsupervised clustering of clicks is being used for identifying new click frauds [34]. Search poisoning [35], phishing detection [36], malicious ad campaigns [3], hate speech detection[1](Figure 1.2) and crowdturfing [37], are some of the other security applications which are also immensely benefiting from the use of data driven methodologies. Machine learning allows scalable and swift learning from large amount of streaming environment data, and also provides generalization to improve longevity of the security mechanisms. With a barrage of algorithms and benchmark datasets developed in literature [25, 30, 31], the use of machine learning has shown promise in its applicability to this field. Table 1.1, shows results of some popular techniques on standard benchmark datasets, with a high classification prediction performance seen as a general trend across different applications.

Although the usage of machine learning in cybersecurity has found early success, its own vulnerabilities have mostly been overlooked. Machine learning was not designed with security in mind, and traditionally its goal has been to provide good generalization from the data it is presented with [39]. Operating in a nonstationary and adversarial environment, as is the dynamic world of cybersecurity, requires a new paradigm in the application of machine learning principles. Our understanding needs to extend beyond its naive usage

[1] `https://www.perspectiveapi.com/`

TABLE 1.1

Machine learning systems used for cybersecurity, with reported performance.

Application domain	**Reported performance**
Network intrusion detection [21]	True Positive: 95%
Malicious executable detection [32]	Area Under ROC: 0.996
Spam email detection [29]	Accuracy: 97.6%
Mouse based behavioral biometrics for CAPTCHA [31]	Accuracy: 99%
Phishing websites detection (Google) [3]	False positive rate: 0.1%
Malicious Pdf file classification [33]	False positive rate: 0.2%
Android malicious apps detection [38]	False positive rate: 5.7%

borrowed from other domains, where it has shown early success. Machine learning is a tool available to both the system attacker and the defender. While, it has been used for defending systems, its usage by an adversary to reverse engineer and evade the system, has only recently started to be considered [40]. This dissertation aims to further bridge the gap between machine learning and cybersecurity, by integrating the actual needs of security applications (i.e., to provide security), in the design of the data driven models.

1.1 Learning in adversarial environments

With the introduction of machine learning at the core of cybersecurity systems, a new gamut of vulnerabilities have been introduced, which need further analysis and understanding [1, 5, 41, 42]. These attacks are themselves machine learning driven, as they rely on probing the system in an attempt to reverse engineer it, and as a result pose new threats to the system [43]. We particularly consider the task of binary classification, where the defender tries to separate instance of the *Legitimate* and the *Malicious* type. This is a common setting in many cybersecurity applications, which are interested in keeping all malicious activity out, while allowing *Legitimate* benign traffic to enter the system. In such settings, the defender tries to generalize from the provided data $D = \{(x, y)|x \in X, y \in Y\}$, where x is a d-dimensional vector and $y \in \{Legitimate, Malicious\}$, is the class label of the sample. The goal of the defender is to find a classifier hypothesis f, based on the following optimization function [41]:

$$f_{optimal} \quad = \quad argmin_{f \in F} \sum_{(x,y) \in D} l(y, f(x)) \quad + \quad \lambda \quad O(f) \tag{1.1}$$

where, $f_{optimal}$ is the learned model, $l(.)$ is the loss function which penalizes incorrectly classified samples, and $O(.)$ captures hypothesis complexity, which is controlled by the regularization parameter λ, to prevent overfitting. In a non-adversarial setting, the optimization function of Equation 1.1 should result in good prediction performance over the data, under the assumption of Stationarity, which states that the training and the test data are drawn from the same distribution (i.e., are Independent and Identically Distributed (IID)) [41]. However, in an adversarial setting (particularly), this assumption is violated. An attacker aiming to evade the system, will generate data different from what is being blocked by the defender's classifier. The adversary's goal is to increase the false negative rate of the classifiers (i.e., increase samples which are *Malicious* but classified as *Legitimate*). In [44] it was shown that once an attack has been detected and blacklisted, its usage itself drops, making static counter measures less useful for the long run. This observation makes the naive application of machine learning to cybersecurity, a futile effort.

Traditional metrics of model performance, such as: accuracy, precision, recall and f-score [45]; have little meaning, if the trained model can be easily evaded at test time. An adversary, also equipped with machine learning, can query the current deployed model to gain information, in an attempt to reverse engineer it, and then generate malicious samples which will result in a high evasion rate [5]. Table 1.2 shows our initial adversary experimentation on three popular cybersecurity datasets: CAPTCHA [31] - Dataset containing mouse movement data to distinguish malicious bots from humans, KDD99 [24]- Intrusion detection dataset containing information about malicious and benign network connections, and SPAMBASE [30]- Email spam dataset. These datasets have shown to benefit from the naive application of classification models, as shown by the initial perceived accuracy. All three datasets show a good accuracy (>80%), when considered as a static machine learning model fitting task. However, it takes less than 30 random probes to evade these models and obtain an evasion accuracy of over 75%. This evasion attack was done by probing

TABLE 1.2

Initial experiments showing vulnerability of machine learning systems to evasion attacks.

Analysis / Dataset	**CAPTCHA**	**KDD99**	**SPAMBASE**
Initial perceived accuracy (10 fold - linear SVM)	100%	99.03%	80.8%
Probes needed to get a legitimate attack (Avg over 30 runs)	3.6	8	29.4
% of attack samples misclassified by the initial SVM	92.9%	87.9%	77.1%

the classifier (by randomly generated data points, upto 100 points), to get initial samples classified as type- *Legitimate*; and then generating 100 attack points, by mutating (adding small numerical perturbations) these initial legitimate samples, to exploit the new found vulnerability. An evasion rate of over 75% on these 3 diverse and high dimensional datasets, while using a simple random probing based approach, indicates the innate vulnerabilities of machine learning. The trained classifier model is like a Swiss cheese barrier to the attackers, who can easily percolate the system, over time. The accuracy of machine learning provides a false sense of security, as seen in Table 1.2 for CAPTCHA, where a 100% accurate model was easily evaded by $\sim$4 random probes, and a misclassification rate of 92.9% was obtained with just 100 probes, in total. This experiment was conducted to highlight the need for a different set of metrics, modeling and paradigm, when applying machine learning in the domain of cybersecurity.

The issue of adversarial manipulation of learning models has gained attention in recent times, with researchers trying to characterize the attacks and defenses on popular machine learning models [1, 40, 41, 46, 47]. This has led to the development of the new field of 'Adversarial Machine Learning', at the intersection of machine learning and cybersecurity [48]. This field aims to study and enable the safe application of machine learning, when adversarial activity is expected, and to develop proactive robust machine learning models which are resistant to attacks. Google's framework for adversarial advertisement detection [3], Facebook's phishing protection system [49], Sandia National Lab's report on their Counter-adversarial data analytics project [50], and work done by several

TABLE 1.3

Categorization of attacks against machine learning systems.

Influence	*Causative* - Influences training data, to mislead learning
	Exploratory - Affects data at test time, to evade detection
Specificity	*Targeted* - Attack is on a particular instance
	Indiscriminate - Attack irrespective of any particular instance
Security violation	*Integrity* - Results in increased false negatives for the defender
	Availability - Causes increased errors of prediction (DOS attack)

researchers [37,41,51–53], all indicate a pressing need for security analysis and robust design of learning systems.

Categorization of attacks against machine learning systems

One of the earliest works on machine learning security was presented in [1], where a taxonomy of attacks based on the principles of information security, was defined. This taxonomy categorized attacks along the three axis of: Specificity, Influence and the Security violation, as shown in Table 1.3. Causative and Exploratory attacks are the two broad categorizations of attacks on machine learning, based on their influence on the data, and are widely studied in literature [52]. Causative attacks affect the training data, while Exploratory attacks affect the test data, on which the learning model operates [52]. Based on the Specificity of attack, an adversary could either affect a particular targeted subset of samples or could generate attacks indiscriminately, without starting with a preset instance. Based on the security violation, attacks can be used to violate integrity, by gaining unsanctioned access to the system, or can be used to launch a denial of service availability attack on the machine learning based system.

Causative attacks aim to mislead training, by poisoning the training set, so that future attacks are caused easily [54, 55]. An example of causative attacks in the real world is the incident with Microsoft's AI driven chat bot, called Tay[1], which got trained to become racist within 24 hours of being deployed into the real world. Causative attacks are severe

[1] `http://fortune.com/2016/03/24/chat-bot-racism/`

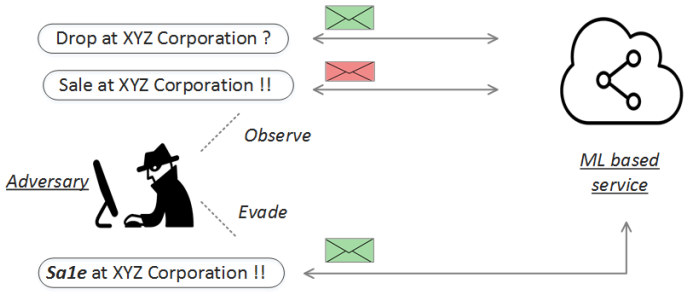


Figure 1.3: Illustration of exploratory attacks on a machine learning based spam filtering system.

in effect, but can be prevented by careful curation of the training data and by keeping the training data secure- using database security measures, authentication and encryption. Exploratory attacks are more commonplace, less moderated and can be launched remotely without raising suspicion. These attacks affect the testing time data and are aimed at reducing the classifier's performance [52]. This is done by means of crafting the attack samples, so as to evade detection by the defender's model [5, 51]. Exploratory attacks start with some level of reverse engineering, where the attacker performs a reconnaissance of the system to understand its behavior, and then leveraging of the learned information to modify the attack payload and escape detection. As an example, consider the good words and bad words attacks on spam detection systems [56], as illustrated in Figure 1.3. By launching two spam emails, each differing in only one word 'Sale', it can be ascertained that this word is important to the classification, if the email containing that word is flagged as spam. Knowing this information, the adversary modifies the word to be 'Sa1e', which looks visually similar but avoids detection. Similarly, to evade a behavioral keystroke biometric systems [31], an attacker can make bots having different keypress speeds across the spectrum, and then choose the bot parameters for which the biometric classifier does not flag suspicious behavior.

Exploratory attacks are unrestricted and commonplace, for any deployed web based system. These attacks are non-intrusive in nature and originate from the client side, as data which is blended with the other system inputs [37]. As such, they are harder to detect and prevent. In web based application relying on ad revenue, the goal is to make the service freely

available to a large number of users across the world, to increase value. In order to do so, the applications have to keep the system open to new users joining and using the system, in large volumes. Having a system out in the open makes it a target for attacks by adversaries, who can pose as users, gain information about the classifier providing security, and then launch a distributed attack which goes unnoticed for a long time. Stringent verification techniques, like two step verification, are not widely used for these applications, as they can cause the user experience to degrade and consequently lead to loss in revenue. Consider for example, a click ads system wanting to detect click fraud [57]. Users hate advertisements, and if they are forced to authenticate every time they need to see an advertisement, it would lead to attrition. Most decision on malicious intent is made on latent signals, such as- click time, text content, propagation rate across users, and mouse movement [27, 31, 57]. All of which can be reverse engineered by carefully crafted attack samples. Machine learning models naively used in these environments do not consider this vulnerability innate to them. This defeats the goals of security, which is the primary purpose of the cybersecurity systems in which these models are employed. In this dissertation, exploratory integrity attacks are studied, and strategies to secure against such attacks are proposed, as these attacks occur frequently at test time and can affect any machine learning based system deployed in the wild.

1.2 Learning in non-stationary environments

As most real world data tends to be dynamic and ever changing, the training data, on which the machine learning models were built, is different from the test data, seen by the system once it is deployed. This change in the underlying data distribution is called *Concept Drift* [58, 59]. Concept drift can cause the model's generalization performance to drop, and as such warrants an adaptive learning scheme capable of learning from new data, as it becomes available [60]. Concept drift is a result of the dynamics of the world, which causes static models - trained on earlier obtained training data, to grow old and stale. For example, consider a model trained to predict the demand for battery powered cars. Before

the release of Tesla[1], this demand was from a select group of early adopters only. But new technologies and changing markets, led to a previously unforeseen demand, and the need for a new model to predict it. The old model of demand was no longer usable, as the new data arrived. This phenomenon is exacerbated in cybersecurity domains, as changes are not only plausible but infact expected, due to adversarial activity. There is a need to detect data shifts and adapt to them, in real world machine learning systems, to ensure continued efficacy of the system.

Detecting concept drift is a challenging task, as it requires human expertise in the form of labeled data. In a streaming environment, where data is constantly flowing in, labeling all samples is not practical. Labeling requires human intervention, and is therefore expensive and time consuming [61]. Consider for example, a large scale spam filter, using classifier at its core and facing the worldwide email traffic (estimated at 205.6B [62]). If even 0.01% of the emails are asked to be labeled, by use of crowdsourcing websites like Mechanical Turk[2], this would imply an expenditure of $200K (each worker paid 1 cent), and an available workforce of 10K workers (assuming each worker reads 100 emails/hour and work 20 hours/day), every single day. This makes the over-dependence on labeled samples a practical and economic limitation. Streaming data applications need to be able to operate and detect drifts from unlabeled data, or atmost sparsely labeled data, to be of any real use.

Drift can be formalized as a change in the joint probability distribution of samples X and their labels Y, over time , as shown in Equation 1.2 and 1.3.

$$P(X,Y) \quad = \quad P(X|Y).P(Y) \quad = \quad P(Y|X).P(X) \tag{1.2}$$

$$P_{t1}(X,Y) \neq P_{t2}(X,Y), \Longrightarrow Concept \quad Drift \tag{1.3}$$

where, change in the joint probability distribution $P(X,Y)$ form t1 to t2 ($t1 < t2$), is termed as concept drift. Here, drift can be a result of the change in the class

[1] www.teslamotors.com

[2] www.mturk.com

distribution $P(Y)$, or a change in the classifier model boundary given by $P(Y|X)$, a change in the class conditional probability $P(X|Y)$, or a change in the data distribution $P(X)$ [58]. While any of these changes leads to a concept drift, due to changed data distribution, the change in $P(Y|X)$ is considered most critical and as such has been widely studied [63] . This change is called the real concept drift, as it most commonly causes the classification performance to drop. Detecting these changes usually requires labeled samples, which are scarce in streaming data. To reduces dependence on labeling, unlabeled change detection schemes have been proposed, which track change to $P(X)$, as a surrogate for real drift [64]. However, these methods suffer from excessive paranoia, as they raise false alarms, being overly sensitive to change. When coupled with human intervention, these can cause the humans to loose trust in the alarm system, as it leads to wasted effort in verifying changes which are flagged frequently and do not result in any actual change in $P(Y|X)$. What is needed is a drift detection system capable of detecting real drifts (i.e., drifts that leads to degradation in the model performance), and one that does so without using excessive labeled data.

Exploratory attacks can be seen as a special type of concept drift, where the drift is driven by an adversary aimed at evading the system. As mentioned in [51], adversary activity is different from traditional concept drift, in the sense that the drift is a function of the learned model itself. Since, the goal of an adversary is to evade the classification, the learned classifier guides the attacks. This phenomenon was observed in the real world, in the study of spam evolution over the period 1998-2010 [44]. It was shown that popularity of the spam construction techniques drives the spam arms race. As soon as the spam campaigns start getting detected and identified, adversaries stop using the same techniques. From a classification point of view, an attack can be seen as a change in the adversary's data- $P(X|Y = Malicious)$ in Equation 1.4.

$$\begin{aligned} P(X,Y) = P(X|Y = Legitimate).P(Y = Legitimate) \\ +P(X|Y = Malicious).P(Y = Malicious) \end{aligned} \tag{1.4}$$

To study adversarial change, the legitimate data is considered to be stationary (no natural concept drift), while an adversary is considered intelligent enough to cloak changes in *P(Y=Malicious)*, to avoid any sudden suspicious spikes in attack vectors. A targeted adversary gains information about the defender's model C; by means of carefully crafted queries, to obtain membership information [6, 51]. It then uses this information to create the attack campaign $P(X|Y = Malicious)$, so as to increase the false negative rate of the learned classifier C. In doing so, only efforts of serious attackers, which launch campaign to significantly subvert the learned classifier and not just to get one single sample past security, are considered. These attackers use machine learning, reverse engineering and diligent reconnaissance missions, to launch large scale attacks. These attacks will force the defender to expend significant time and resources to redesign the entire machine learning model, starting from new data acquisition, retraining, testing and deployment. Study of adversarial learning as a nonstationary phenomenon and as a special case of the general concept drift phenomenon [65], is at the core of our work. We refer to such drifts in the data distribution as the *Adversarial Drift*, to separate it from the broader Concept Drift term.

1.3 Dynamic Adversarial Mining

Existing works on adversarial mining, approach the problem from two perspectives : i) As a streaming data problem, where any change is possible and needs to be detected and fixed [59], or ii) As a security problem, where proactive decisions are made in the training of resilient models, to delay the onset of attacks, but nothing is done once the system is deployed [41]. There is a knowledge gap between these two approaches, as they fail to undertake a holistic view on the security of a system. A holistic view of the security would understand that it is a cyclic process, where delaying attacks, detecting them and then recovering, are all equally important to meet the system's security goals. A dynamic security paradigm would take proactive measures to delay attacks, and to make their subsequent detection and fixing easier, as well. As such metrics like Attack delay, Difficulty to reverse

engineer, Detect-ability of vulnerability of classifiers, and Recover-ability after attacks; are more useful to dynamic security, than the traditional metrics of accuracy, precision and recall. Also, dynamic security needs to operate as a *never-ending* learning scheme, with efficient involvement of human in the loop, to provide expertise and retraining ability, from time to time. The interdisciplinary field of *Dynamic Adversarial Mining* (Figure 1.4), needs to incorporate lessons learned from: Machine Learning, Cybersecurity and Stream Data Mining; to rethink use of machine learning in security applications and develop systems which are *'secure by design'* [66]. Some core ideas under the dynamic adversarial mining scheme are:

- Ability to leave feature space honeypots in the learned classifier at training time, to efficiently detect attacks using unlabeled data at test time.
- Semi automated self aware algorithms, which can detect abnormal data distribution shifts, at test time.
- Maintaining multiple backup models, which can provide prediction when the main model gets attacked.
- Ability to use a stream labeling budget effectively and to distribute the budget appropriately, to detect and fix attacks. Thereby, managing human involvement in the process.
- Understanding attack vulnerabilities on classifiers, from a purely data driven perspective, to effectively test the security measures employed.

This work takes steps towards an integrated approach of machine learning security, under the canopy of *Dynamic Adversarial Mining*. Our work aims to encourage further awareness and interest in the need for a dynamic security paradigm, by bringing together ideas from the domains of Streaming data and Cybersecurity.

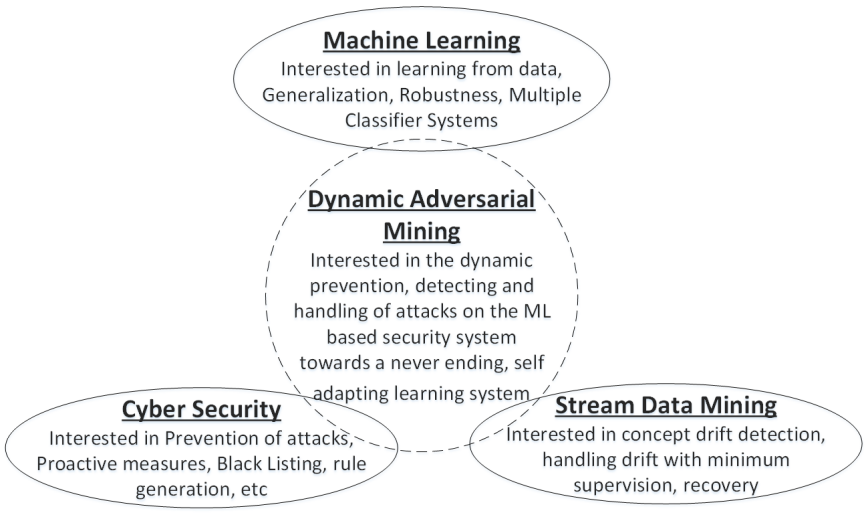


Figure 1.4: The field of *Dynamic Adversarial Mining*, derives from the concepts of Machine learning, Stream data mining and Cybersecurity.

1.4 Dissertation overview and major contributions

In this dissertation, we present approaches for dealing with adversarial drift in machine learning systems, when used in vulnerable application domains. We start by demonstrating shortcomings in current usage of machine learning based classification techniques, by developing an exploratory attack simulation framework. We then propose an unsupervised drift detection methodology, capable of detecting data distribution changes, with a low false alarm rate. The specific nature of adversarial environment and the impact of the Defender's decision, on the space of possible attacks, is demonstrated next. Based on lessons learned from these works, we develop a dynamic security system which combines proactive and reactive security paradigms, for providing long term security in *Dynamic-Adversarial* environments. Detailed layout of the thesis is presented next.

Chapter 2 presents the background related work on the security of machine learning. Two schools of thought are identified in literature: The *Proactive* security community, which wants to delay attacks, and the *Reactive* security community, which wants to recover from attacks swiftly. Survey of existing approaches in both these areas is presented and the need for an integrated approach is highlighted. Also, work on multiple classifier systems and their advantages, to both proactive and reactive security, is presented.

The vulnerabilities of machine learning to exploratory attacks, at test time, is empirically evaluated and studied in *Chapter 3*. This chapter presents the adversary's view of machine learning based defense, and presents data driven techniques, which an adversary could use to circumvent these defenses. The Seed-Explore-Exploit (SEE) framework is proposed, as a way to simulate attacks on black box classifier models. Two attack techniques are presented under the SEE framework: The Anchor Points (AP) attack and the Reverse Engineering (RE) attacks. These attacks represent different attacker goals, resources and effects; and are evaluated on 10 real world datasets. Additionally, metrics to evaluate quality of attacks, based on attack accuracy and diversity, are proposed and evaluated, to better understand adversary goals, capabilities and intentions.

As explained in Section 1.2, attacks on machine learning systems are a special case of concept drift. Detecting these concept drifts is a challenging task, as labeling is scarce and expensive, in a streaming cybersecurity application. *Chapter 4* presents an unlabeled drift detection methodology, which reduces the dependence on labeled samples. The proposed Margin Density Drift Detection (MD3) algorithm is a distribution independent, classifier independent, robust and streaming unlabeled drift detection technique, which is shown to have a high reliable drift detection rate. The MD3 algorithm performs as well as the fully labeled drift detectors, in terms of stream prediction performance. Also, the MD3 algorithm leads to lower false alarm rate, when compared to other unlabeled drift detection methods. This reduction in false alarm is especially useful for cybersecurity domains, where expert intervention is limited by time and resources. Lower false alarms also leads to increased trust in the system. The presentation of the MD3 algorithm is done from a domain agnostic perspective, making it a suitable technique for dealing with dynamics in the prediction problem, over time.

In *Chapter 5*, the impact of the Defender's classifier design, on the adversarial capabilities at test time, is analyzed. In particular, existing design strategies of- Restrictive one class classifiers, Robust majority voted classifiers, and Randomization based classifier; are shown to exhibit shortcomings in *Dynamic-Adversarial* environments. The idea of attack

Detect-ability and Recover-ability, are highlighted as essential requirements for dynamic environments. Based on this analysis, a novel feature importance hiding design is proposed for better long term reactive security.

Based on insights we obtained about the nature of data driven adversarial activities (Chapter 3, Chapter 5), and the ability to detect unlabeled changes efficiently (Chapter 4), we propose the development of a dynamic security framework, in *Chapter 6*. The proposed *Predict-Detect* streaming classifier framework, relies on the motivation that adversarial concept drift is dependent on the learned model. As such, the framework aims to mislead attackers, by goading them to affect feature honeypots left in the trained classifier. Thus allowing them to be detected easily, using unlabeled samples. The framework also provides benefits for active learning, for better retraining after attacks. It provides for a novel adversarial aware drift handling system, which is designed to be extensible and customize-able, to suit different design needs. Concluding remarks and potential future work, is presented in *Chapter 7*.

The major contributions of this dissertation are:

- Demonstrating vulnerability of black box classifier systems, to adversarial test time evasion samples. A data driven framework is presented, for white hat analysis and systematic testing of adversarial capabilities and impact.

- A reliable unsupervised drift detection framework, for effective signaling of significant performance affecting concept drifts from unlabeled streaming data. The framework is developed to be domain agnostic and classifier type independent, for generalized usage in dynamic environments.

- Detailed evaluation of the impact of classifier design, on the adversarial capabilities at test time. The idea of feature importance hiding is demonstrated as a veritable solution to ensure continued usage in adversarial drifting environments.

- An adversarial aware drift detection framework, which can provide unsupervised indication of adversarial activity for deployed classifier models. The first of it's kind anal-

ysis of simulated adversarial drift generation and handling, in a continuous streaming environment, is presented.

- Foundations and extensible experimental frameworks, for work in the new interdisciplinary area of *Dynamic Adversarial Mining*, where a holistic approach to security is demonstrated to be effective for long term security.

CHAPTER 2

REVIEW OF WORK ON THE SECURITY OF MACHINE LEARNING

Cybersecurity systems, by their very nature, are in a never ending arms race between the attacker and the system defender (Figure 2.1). In a cyclic process, the defender tries to make attack resistant systems, while the attacker continuously tries to find and exploit design flaws [49]. This warrants a dynamic view of the security problem (i.e., as an online system capable of learning and adapting to changes over time), instead of trying to find a one shot solution. The recent popularity of machine learning in cybersecurity domains (e.g., spam classification, CAPTCHA systems, Intrusion detection), has garnered interest in its security properties, from various viewpoints [52, 67–69]. The current research in the domain of machine learning security can be broadly divided into two school of thoughts: a) *Proactive security* and b) *Reactive security* [66, 68].

Proactive security is aimed at anticipating adversary strategies and making systems resistant to possible attacks [70–76]. These methods focus on the *Prevention* component of the attack-defense cycle (Figure 2.1), with the aim of delaying an attack till the maximum possible extent. As shown in Figure 2.2) a), these defensive measures aim at increasing the attackers effort δ_P, which is needed to degrade the target system's performance (evade the system). Model performance is given by the correct classification capability of the defender's model, and is typically measured using metrics such as: Accuracy, F-score, Precision and Recall. Work on proactive security views the attack-defense problem as a static one, with the intention of bolstering defenses before deploying the system. The goal of these approaches is to make the system stay active for a long period of time, with less frequent human supervision and reduced susceptibility to most attacks. However, effects of system attacks and fixing the system following an attack, are not discussed or analyzed by works

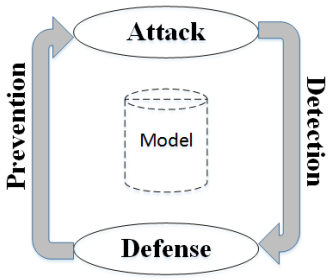
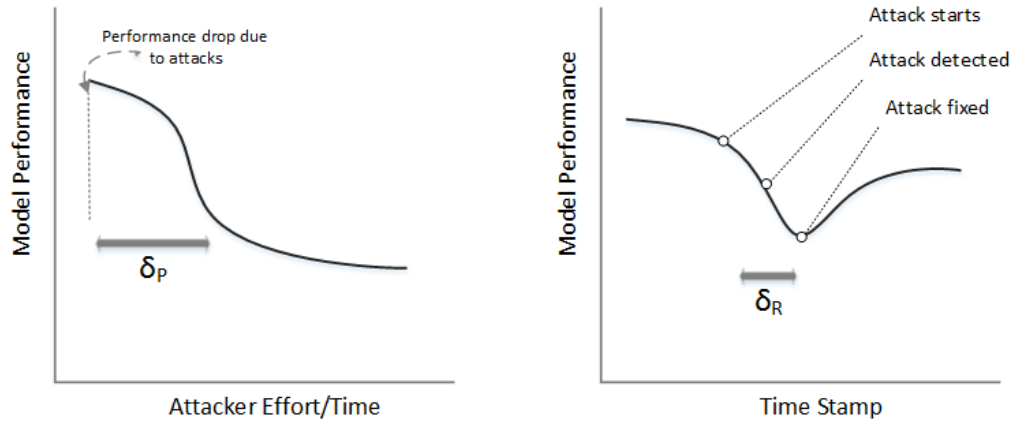


Figure 2.1: The Attack-Defense cycle between adversaries and system designers (i.e., defenders).

on proactive security.

While works on proactive security emphasize on the prevention problem, *Reactive security* focuses on the problem of fixing the system after it is attacked [68, 77]. Reactive security emphasizes on the *Detection* phase of the attack-defenses cycle (Figure 2.1). These approaches aim at being able to swiftly detect attacks and fix the system, with limited human supervision, to make the system available and effective again. As shown in Figure 2.2) b), these measures aim at reducing the delay δ_R, which is the time taken by the system to detect degradation and fix (retrain) the model. Works in the area of concept drift detection and adaptation are suitable for reactive security, as they do not make any explicit assumption about the type of change and aim at maintaining high model performance, over time [59, 78, 79].

Proactive security and *Reactive security*, are the different paradigms for securely using machine learning in an adversarial environment. An analogy to explain the two is: Consider the task of securing a precious artifact. *Proactive security* aims at securing the vault in which the artifact is kept- by making thicker walls, by setting up barricades, chains and locks, and by keeping track of suspects who have a history of theft. On the other hand, *Reactive security* would concentrate its efforts on setting up surveillance and alarm systems, so as to be able to apprehend the thief easily, if at all a larceny is attempted. While proactive security might seem to be an obvious choice, this is not always the case [68]. Proactive security relies on making explicit assumption about the adversary's behavior, its reward function, objectives and rationality. They are suited for prevention against catastrophic



(a) *Proactive security* aims at increase time and effort needed(δ_P) to degrade a model

(b) *Reactive security* aims at reduce the time taken to detect attacks and fix them(δ_R)

Figure 2.2: Goals of *Proactive* and *Reactive* Security.

events, which would lead to a sudden and total collapse of the system [68]. However, in most cases, Reactive security can perform equally well, without making limiting assumptions about the type of attacks, and by using fewer resources. In [68], it was suggested that security officers should concentrate effort on monitoring tools, to detect attacks, and on agile teams, to deals with an attack swiftly. While both schools of thought (*Proactive* and *Reactive*) are effective in dealing with a portion of the attack-defense cycle, a combined approach is needed. A practical cybersecurity system should take into account the dynamic and temporal nature of the problem, integrating proactive measures to delay attacks and reactive measures to swiftly detect vulnerabilities and recover from it. Effective involvement of human in the loop, to ensure detection and recovery, is also an essential consideration for such a system [61].

This chapter presents related work in the field of *Proactive* security and *Reactive* security, for dealing with exploratory attacks [41]. Section 2.1 presents works related to Proactive security. Works related to streaming data and reactive security approaches are presented in Section 2.2. Use of Multiple Classifier Systems(MCS) as a popular tool in cybersecurity streaming applications is discussed in Section 2.3. Chapter summary and

main takeaways is presented in Section 2.4.

2.1 Related work on proactive cybersecurity

Proactive security techniques have been developed for both: a) Causative attacks and b) Exploratory attacks. Causative attacks are caused by attackers affecting the training dataset [52, 54, 67], by techniques such as poisoning [80] and label flipping [55, 67], or by misleading the learning process, using red herring attacks [41]. These attacks poison the datasets, by adding samples which disproportionately move the classifier boundary [1]. An example of this is illustrated in Figure 2.3, where a one class outlier detector is gradually shifted, over several iterations, by introducing samples close to its boundaries. Defense against causative attacks are aimed at increasing the robustness of the learning model, so that it is not affected by small number of injected malicious samples and noise. The Reject On Negative Impact (RONI) is a popular defense strategy against causative attacks. RONI aims to remove samples with high influence on the classification boundary, before the training is performed [41]. This ensures that training is not driven by a handful of influential samples. Other techniques aimed at dealing with causative attacks use game theory models to ascertain the optimal strategy of the defender, knowing the adversary's optimal strategy [70, 81–83]. In an iterative manner, the attacker modifies attack samples (constrained such that the original malicious content is not lost), while the classifier is retrained to correctly classify them. The modeling is done as Nash and Stackelberg games in [84, 85]. Equilibrium state is found, such that the utility of both- the defender (aiming to obtain maximum training accuracy) and the attacker (aiming to increase evasion by minimal modifications), is balanced, with each party having complete awareness of the other. The learned model is assumed to provide security for a long period of time, as it is secure against a rational adversary, who aims to minimize its effort needed to attack models. However, such approaches have the disadvantage of needing to specify the exact payoff function for the two parties. Also, these approaches are not scalable to large amounts of data.

Exploratory attacks affect the performance of the classifier at test time, by means

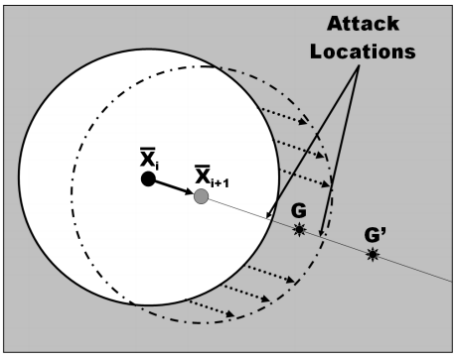


Figure 2.3: Causative attacks causing the one-class classifier to shift. G and G' are goals of the adversary, who intends to move the classifier boundary over time [1].

of crafted test samples aimed at evading the defender's model. Evasion is performed by means of partial reverse engineering of the model (using probes), to understand its working and parameters. This information is then used to modify adversarial attack samples, so that they will be classified as legitimate by the defender model [5, 51, 69, 86]. Works aimed at securing machine learning systems at test time can be divided into two categories: a) Complex learning methods and b) Disinformation methods. These categories of attacks are discussed in the following sections.

2.1.1 Security based on learning of complex models

These methods increase the complexity of the learned model, to make it harder for an attacker to effectively reverse engineer it with a limited number of probes. The complexity is increased either by using difficult to mimic features in the classification model [2], or by balancing weights across features, such that the resulting classification models are robust [2, 71, 87]. Such robust classifiers require the adversary to reverse engineer and mimic a large number of features, thereby increasing its efforts and cost, to successfully evade the classifier. A practical challenge that these methods face is the balancing of complexity with overfitting, as overfitting can lead to poor generalization performance and also make the system vulnerable to causative attacks [41].

Complexity of the models can be increased systematically, by distributing weights among informative features, to make a classifier robust [2]. In [88], feature reweighing

was used by weighting every feature inversely based on their importance. This made the classification boundary dependent on a larger number of features, thereby requiring attacks to spend more effort in trying to evade the classifier. In [89], the task of selecting a reduced feature subset in an adversarial environment was presented. Classification security was introduced as a regularization term, in the learning objective function of the defender's model. This term was optimized along with the classifier's generalization capability, during the feature selection process, making it more secure against evasion attacks on reduced feature spaces. In [90], an adversarial aware SVM was proposed, which considers two types of attacks: a) Free range attacks, where adversaries can freely move in the dataspace, and b) Restrained attacks, where an adversary incurs cost based on distance from existing samples. By incorporating possible moves of the adversary, as constraints to the SVM optimization function and by modifying its loss function, a resilient SVM model was trained. It was also shown that an overly pessimistic view of attacks can lead to poor performance, as attacks in practicality are much weaker. Robustness to random feature deletion at test time, caused by noise or by adversarial reverse engineering, was introduced in [83]. It was shown that the proposed methodology was resistant to changes caused by random feature deletion and also to, a reasonable extent, those caused by deletion based on feature importance. In [75] the predictive defense algorithm was proposed, which adds regularization terms in the min-max formulation of the objective function, to model adversary actions in aggressively reduced feature spaces. The objective function optimized in [75], is given by Equation 2.1.

$$\min_{w} \max_{a} \left[-\alpha \|a\|^3 + \beta \|w\|^3 \quad + \sum_{i} loss(y_i, w^T(x_i + a)) \right] \tag{2.1}$$

Where, the loss function represent misclassification rate and w represents the learned model of the defender. The attacker attempts to circumvent the defender by using a linear transform vector $a \in R^d$. The terms α and β denote the regularization imposed on attackers and defenders, respectively. The term α embodies the effort of an attacker, which increases based on the distance from existing samples. The term β denotes the defenders need to avoid overfitting to the data. By incorporating both regularization terms into the

optimization function, this method makes it harder for an adversary to apply a simple linear transformation on a few features of malicious samples, to get them classified as *Legitimate* by the defender model. The use of this game theoretic formulation to the problem was shown to outperform a naive Bayes classifier, used as gold standard, for the task of spam classification with reduced feature space.

While the above methods rely on adding robustness to the training of a single classifier, it was shown in [72, 87, 91] that combining classifiers trained on different subsets of the data is more suitable to the task of security. Multiple Classifier Systems (MCS) [4] allow classifiers trained on different subsets of the feature space to be combined in a natural way, to provide an overall robust classification prediction. In [72], instance bagging and random subspace methods were analyzed, as two MCS approaches, for securing against evasion attacks. It was found that both approaches make the evasion problem harder for the adversary. In particular, random subspace models are more suitable when a very small number of features exhibit high discriminating capability, whereas bagging works better when the weights are already evenly distributed among the features. These methods essentially have the effect of averaging over the feature weights, as models trained on different views of the problem space are aggregated to present the final result [87]. Related work in [5], provides an empirical evaluation of the evasion hardness, by computing adversary effort needed to launch gradient descent evasion attacks on the learning model. Theoretical proofs for effectiveness of MCS was presented in [73]. It was shown that arbitrary conjunctions and disjunctions of individual linear classifiers, makes probing attacks by an adversary exponentially expensive. It was also shown that models trained on disjoint subsets of the feature space are theoretically equivalent, in their vulnerability to evasion attacks, as compared to linear models. Heterogeneous combination of one class and binary classifiers, to produce an evasion resistant model, has shown to balance high accuracy with improved security, in works of [92].

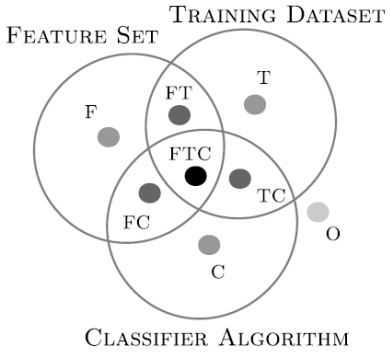


Figure 2.4: Amount of information known by an adversary. F denotes Feature space information, C is information about Classifier used and T is information about training set. O is outlier information [2].

2.1.2 Security based on disinformation

These methods rely on the concept of *'Security by obscurity'*, and they are aimed at confusing the adversary about the internal state of the system [66]. The inability of the adversary to accurately ascertain the internal state would then result in increased delay/effort for performing the attack. Popular principles for applying disinformation based security are: Randomization [48], Use of honeypots [93], Information hiding [94] and Obfuscation [1]. An example obfuscation technique used in case of a classifier filter, is one where the classifier would output correct prediction with a probability of 0.8, in an attempt to sacrifice accuracy to provide security. Ideally, hiding all information about the feature space and classification model could result in complete security. However, this goes against the Kerchkoff's principles of information security [95,96], which states that security should not rely overly on obscurity and that one should always assume that the adversary knows the system. The amount of information known by the adversary was formalized in [2], as shown in Figure 2.4. Here, F denotes information about the feature space, C denotes classifier parameters learned and T denotes the training dataset. The amount of information known by an adversary is given by the different regions of Figure 2.4, and it can range from limited information, such as only the feature information known (F), to complete symmetry between the defender and the adversary (FTC, where feature space,

training data and classifier trained is known by the adversary). While, the training set and classifier information can be assumed to be kept safe, by advanced database security and encryption methods [97], information of the feature space should be assumed to be known by the adversary. This information could be ascertained from publicly available information from related applications, or by educated guessing, by the adversary. The information distribution representation of Figure 2.4, provides a formal mechanism to specify constraints within which the adversarial analysis is done, for systematic evaluation and assessment of application needs and defense capabilities.

When disinformation relies on obfuscation, the amount of information made available to the adversary is not limited, but obtaining the information is made harder [74]. Randomization is a popular approach to ensure that information presented is garbled. A simple strategy for introducing randomization into the classification process was proposed in [1], where it was suggested that randomness be introduced in placing the boundary of the classifier. This was made possible by using the confidence estimate given by a probabilistic classifier, to randomly pick the class labels. While this increases the evasion efforts necessary, it leads to a drop in accuracy in non adversarial environments. A detailed analysis of this tradeoff was presented in [6], where a family of robust SVM models were learned and used to increase reverse engineering effort. A semi definite programming formulation was used to learn a family of SVM classifiers, wherein any single classifier could be picked at random to perform the classification with high accuracy, at any given time. It was shown that a high variance in the distribution of classifiers is possible without incurring loss in predictive performance of the classifier. Using evasion accuracy and deviation in the learned model, it was shown that the proposed model was able to mislead the adversary better than [1], without any significant drop in accuracy in normal adversarial environments.

The work in [6] showed that choosing a single classifier is a suboptimal strategy, and that drawing classifiers from a distribution with large variance can improve resistance to reverse engineering, at little cost to the generalization performance of the model. This idea has also been used in multiple classifier systems, where a random classifier is selected

at any given time, from a bag of trained classifiers, to perform the classification. The moving target approach of [98], divides features in K subsets and trains one classifier on each of these subsets. One of these classifiers is chosen, according to a scheduling policy, to perform the classification of an incoming sample x. It was shown that using a random uniform scheduling policy provides the best protection against an active adversary. This was formally proven in [76], where it was shown that the optimal randomization strategy is to select classifiers randomly with equal probabilities. In case of targeted attacks, selecting uniformly was shown to be the best strategy, whereas no randomization was the optimal strategy in the face of indiscriminate attacks. Although the ideas of [98] and [76] are applicable to multiple classifier systems, experimental results were shown only on a set of two separately trained classifiers. Obtaining multiple sets of classifiers, with high accuracy, was not discussed. The work of [91] uses the multiple classification system formulation, by maintaining a large set of classifier models and with randomization being performed, by assigning a different weighting scheme to the individual classifier's prediction, for every iteration. Experimentation was performed on the Spamassassin dataset, by pre-selecting 100 weighting schemes and allowing the classifier to choose any one of them with equal probability, over different iterations. This resulted in increased hardness of evasion of the classifier under simulated attacks.

While randomization is a popular method for incorporating obscurity against evasion attacks, other methods have been suggested [66]. Limiting the feedback or providing incorrect feedback, to combat probing attacks have been suggested in [1]. Use of honeypots is another promising direction, wherein systems are designed for the sole purpose of being attacked and thereby providing information about attackers [93]. Use of social honeypots was suggested in [99], as a means of collecting adversarial samples about social spam. Bots which behave as humans are deployed in the social cyber-space, specifically tested on Myspace and Twitter data in [99], where they collect information about potential spammers.

Although a lot of recent work has been performed in the domain of security of machine learning, there is still a long way to go before security is ingrained into the learning

task. A move towards *'security by design'* [52] is necessary to ensure that the algorithms are secure by their very nature, instead of external shields introduced as a last resort. While traditional machine learning is concerned only with a good generalization over the data, this might not be an optimal strategy if adversarial activity is expected. As shown in [37], a simpler good fit to a small set of homogeneous data generates high accuracy (for the application of detecting malicious crowdsourcing campaigns), but also makes the same model vulnerable and easier to evade. On the other hand, work in [100] has shown that over-fitted models leads to data leakage and privacy issues. There is a need to balance generalization with security, especially in the domain of cybersecurity, where machine learning is introduced primarily to ensure security. Work done by [6, 101], are first steps towards understanding the accuracy-security trade-off and the practicality of applying machine learning in an adversarial environment.

2.2 Related work on reactive cybersecurity

Adversarial learning is a cyclic process, as described in Figure 2.1 and in [49]. Attacks are a question of *when* and not *if*. Proactive measures of security delay the occurrence of attack, but eventually every system is vulnerable to attack. These measures do not provide any direction or solution for dealing with attacks and for fixing the system for future use, after it has been attacked. They are static one shot solutions, which drop the ball as soon as a system is compromised. Reactive system on the other hand, can deal with attacks after their onset and aim to fix the system as soon as possible. Work in [68] shows that, in the absence of prior information about adversaries, reactive system performs as good as a proactive security setting and is a suggested security infrastructure for industrial scale systems. Reactive systems can also ensure that attacks are recognized and alerted, to take appropriate counter measures.

Reactive methods have primarily been studied in the domain of streaming data mining, where changes to the data, called concept drift, are detected and the system is adapted to maintain performance over time [58]. In these techniques, the system is agnostic

to the cause of change and treats any degradation in its performance in a similar way. As such they are reactive but not necessarily adversary aware [81]. The work in [59] summarizes the major frameworks, methodologies and algorithms developed for handling concept drift. A trigger based concept drift handling approach [60] consists of two important components: a) A drift detection module, and b) A drift adaptation module. The drift detection module is responsible for detecting changes in the data and to signal further evaluation or adaptation based on the detected changes. The adaptation module then launches retraining of the system, by collecting new labeled data and updating the model. Ensemble techniques are a popular choice for drift handling systems, as they allow modular retraining of the system [102]. Ensemble methods have been proposed for drift detection [103] and for learning with concept drift [78, 79].

An important consideration while applying drift handling techniques to the domain of cybersecurity, is the need to work with limited labeled data. Labeling is expensive, time consuming and in some cases not a possibility at all. The scale of modern day machine learning system makes excessive dependence on labeled samples unpractical [61]. Several methods in the concept drift literature have worked on this problem by suggesting partially labeled approaches, which selectively label samples (based on a labeling budget) while still maintaining high performance. These techniques have been studied under the online active learning domain, where the task of identifying and labeling the most informative samples is shown to produce good prediction performance with low labeling rates [104–106]. Use of semi supervised methods in [103], emphasize the effective retraining of classifiers after obtaining a few labeled samples from the active learning approaches. In a parallel line of work, methods in [64, 107] aim at reducing label dependence by making the drift detection phase totally unsupervised. While labeling cannot be avoided for retraining, drift detection from unlabeled data is shown to be an effective surrogate to approaches relying on labeled data [63]. Recent works on using unlabeled approaches to handle retraining, as well, have been proposed in [108]. However, these techniques make limiting assumptions on the type and nature of drifts, to make it applicable to such latent updates. Only drift in existing

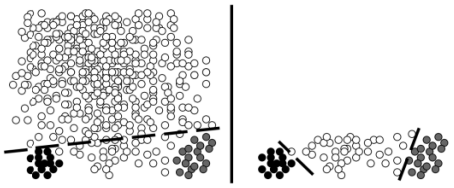


Figure 2.5: Cascaded classifier. Majority of good samples are filtered out by a high recall coarse model (left) and the per class classifier then detects specific hard to detect malicious classes(right) [3].

classes with slow changes over time, are possible to detect using these approaches [109].

Another consideration in applying streaming data techniques to cybersecurity, is the ability to deal with imbalanced data. In real world applications, data about normal behavior is plentiful, while abnormal/malicious behavior tends to be sparse [3]. This could be natural or intentional, as an attacker wants to avoid being detected. Dealing with imbalanced data is not an easy task [110], and it is made challenging with the added constraints on labeling and the effects of concept drift. Methods for dealing with streaming imbalanced data, combine active learning with efficient archiving of labeled samples, to ensure drifts are detected and new models can be retrained [111, 112]. The Reduced Labeled Sampling (RLS) technique of [112] labels samples based on their distance to the support vectors and also to existing minority class samples, to detect drift. Balancing class distributions, during retraining, is done using the Synthetic Minority Oversampling (SMOTE) approach. SMOTE performs convex combination of minority class samples, to generate additional samples. The techniques of [111, 112] performs rebalancing by adding archived minority class samples, closest to the current samples, into the training set. The Google's system for detecting adversarial web advertisements [3], tackles the problem of data skewness in a multiclass imbalanced data environment. By using a one-vs-good scheme, a coarse model is first used to identify if a samples is malicious or legitimate, and then a fine grained model separates each of the difficult classes in data. A cascaded layout of the coarse model (Figure 2.5), to weed out definitely good samples, followed by per class filters, was found to be effective. As in Figure 2.5, the coarse model is able to account for a majority of the samples, but fine grained distributions of the minority class are better captured by the local

models. Additionally in [3], expert feedback was incorporated, for detecting emerging bad ads, by using uncertainty based active learning [104]. Active learning on imbalanced data, can be unfruitful in detecting minority class samples. As such, the proposed system used a search based interface to allow experts to perform guided searches in real time, to find minority samples denoting new attack vectors, based on their intuition. A similar search based algorithm was presented in [113], which divides the data space into grids and then performs uniform guided sampling within grids to detect minority class samples.

At the confluence of concept drift and adversarial learning, are the approaches of adversarial drift [114] and adversarial active learning [115]. In [114], concept drift caused as a result of attacks, was discussed. The need for a responsive system, with the integration of traditional blacklists and whitelists, alongside an ensemble of classifiers trained for every class of malicious activity, was proposed. Additionally, emphasis was made on closing the semantic gap between classification and actionable feedback, by allowing attacks and changes to be described and explained, not just classified. Isolation of malicious campaign and techniques to ensure zero training error, while still maintaining generalization, were extensions suggested for traditional classification systems when used in an adversarial environment. The vulnerabilities of the labeling process were analyzed in [115], which introduces the concept of adversarial active learning. Adversarial environments can affect the selective labeling process, by means of providing malicious labels to mislead the detection and retraining process. The Security oriented Active Learning Test bed (SALT) was proposed in [115], to ensure effective drift management, human integration and querying strategies, in an adversarial environment. Use of concept drift tracking within malware families was analyzed in [116], to show the temporal nature of adversarial samples. Metafeatures (higher order difficult to evade) were suggested, to detect malicious activity. Use of an ensemble of classifier to detect spam emails was suggested in [117], where mutual agreement of pairs of classifiers in the ensemble was tracked and concept drift was detected if the agreement drops. In the event of a drift detection, poorly performing pairs of classifiers are selected to be retrained. Extension of this work was proposed in [53], where the entire ensemble's

disagreement score distribution was tracked. A sudden increase in the overall disagreement was used to indicate an attack on the system, without using external labeled data. Here, feature bagging [118] was found to be an effective ensemble strategy to detect evasion attacks, such as mimicry and reverse mimicry attacks, on the task of classifying malicious pdf documents. These works provide initial ideas for using machine learning in an adversarial environment, where the data is streaming and the attacks-defense is a cyclic never ending process.

The reactive approaches have been well studied towards an online never ending learning scheme. When applied to the domain of cybersecurity, the adversarial cycle demands reactivity, to ensure that attacks do not leave the system totally useless. Security is dynamic (Figure 2.1), as such an holistic detection, updating and learning pipeline is needed. However, the current methods for reactive security do not explicitly consider an adversarial environment, even though a dynamic environment is considered. In adversarial domains, concept drift (attacks) are a function of the deployed model itself, as reverse engineering is based on what model is learned in the first place [1]. This information can be used to design more suited reactive systems in adversarial domains. A combination of proactive and reactive approaches are necessary when dealing with such domains, as decisions made in the design phase could ultimately make future steps down the pipeline easier. For instance, a) Honeypots can make detection and subsequent retraining easier; b) Multiple Classifier Systems (MCS), where many disjoint models are trained simultaneously, can be used to quickly replace a degraded model at any given time; and c) Disinformation can be used to mislead attackers, by misrepresenting the model. Further research in this area will need a comprehensive look at the learning process, with adversarial effects considered at every step of the process, towards an adversarial aware dynamic learning system. This will require newer metrics of measuring system performance, beyond accuracy and f-measure, towards metrics such as recovery time and data separability [119], which are necessary for subsequent cycles of the process.

2.3 Multiple Classifier Systems (MCS) in cybersecurity

According to the *'No free lunch'* theorem [120], in the absence of any specific domain information, no particular classification algorithm is better than the other, on average. In such situations, ensemble of classifiers have shown to provide better accuracy than using any single model. Multiple classifier systems (MCS), commonly known as *Classifier Ensembles*, combine predictions of several independent and diverse models, which are trained on subset of the original training dataset. These combined classifiers are found to have superior accuracy than a single monolithic classifier, provided the base models are unstable and each model has accuracy better than random guessing [121]. The multiple classifier systems have found various application in stream data mining and in cybersecurity applications, owing to their versatility and good generalization performance [72, 121, 122].

Typically, MCS need to perform the following steps: Generate a large pool of classifiers, select subset of classifier and then aggregate results of the classifiers [4]. Generating diverse classifiers, which make independent test errors, can be done by- dividing samples, as in instance bagging, or by dividing features, as in feature bagging [123]. Selecting subset of classifiers can be done at random or dynamically at test time, based on the temporal performance of the models [124]. Fusion of results is either via majority voting, sequentially- as in boosting, via stacking or by Bayesian fusion techniques [125]. The layout of a basic MCS is shown in Figure 2.6. The incoming data samples is given to a pool of classifiers in the ensemble, which make independent predictions on the sample. Based on the fusion strategy, these individual predictions are then combined to provide the final prediction of the system.

There are several advantages to using a MCS instead of a single trained monolithic classifier. The following points elucidate the advantages of MCS as applied to a *Dynamic-Adversarial* domain.

a) *Improved generalization performance:* An ensemble of models trained with independent classifiers, which make uncorrelated errors and have accuracy>0.5, when

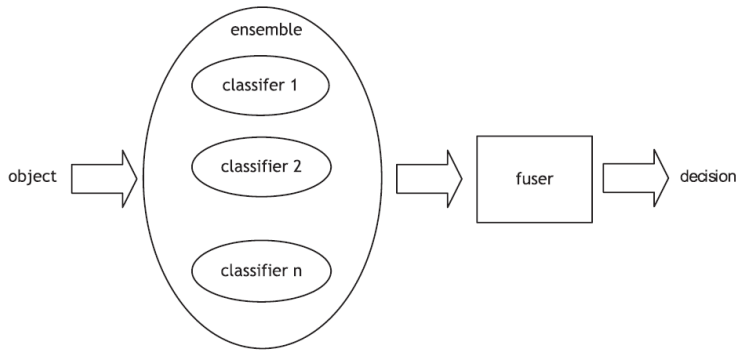


Figure 2.6: General setup of Multiple Classifier Systems (MCS) [4].

combined will lead to improved performance, than a single monolithic model trained on the entire data [121]. To understand this effect, consider an ensemble for binary classification, which combines 50 individual classifiers using majority voting. If the 50 models are different from each other and each has an error rate of 0.4, the ensemble's error rate ($\epsilon_{Ensemble}$) is given by Equation 2.2 (derived from [121]). It is seen that the obtained error of 0.07, is significantly lower than an single model, which makes an error of 0.4.

$$\epsilon_{Ensemble} = \sum_{i=26}^{50} \binom{50}{i} \epsilon^i (1-\epsilon)^{50-i} = 0.07 \tag{2.2}$$

b) *Improved stability-plasticity balance:* While operating in an incremental online setting, models are faced with the *stability-plasticity* dilemma, where models can either retain existing knowledge or adapt to new data. Ensemble methods enable maintaining a balance, by allowing selected classifiers to be forgotten or added to the ensemble, to maintain reactivity or stability as desired [60]. Ensemble models can also dynamically redistribute weights between the classifiers, in the fusion section, where classifier can be assigned weights based on their local and temporal characteristics to the current data in the stream [126].

c) *Modular retraining and learning:* In the face of concept drift, only a subset of the models might degrade in accuracy. Using an ensemble model, it is possible to selectively retrain these classifiers, while keeping the others untouched. This modular

structure also ensures that novel sub-populations in the data can be incorporated into the existing ensemble. This is especially true for malicious campaigns, which usually have multiple sub-populations of attacks within them. By incorporating and tracking these populations separately, they can be effectively learned at a granular level [78, 117].

d) *Combining heterogeneous data sources:* Heterogeneous features can be combined in a natural way in MCS. For multimodal biometric systems, models can be created for face recognition metrics, mouse movement metrics and keystroke patterns, separately, and then weighted and combined to produce a final prediction system [127]. The use of MCS allows these features to be trained by their own individual suitable model. For example, mouse movement patters may be better learned by liner model, while face recognition could benefit from using neural networks. MCS allows flexibility in learning these individual traits and then emphasis is made on combination of these results, which serve as expert reviews on the same problem.

e) *Robustness:* Unstable classifiers, such as decision trees and neural networks, are sensitive to small changes in the training data [123]. This makes them vulnerable to causative attacks, where the attacker tries to modify a few samples to mislead the classification system [128]. MCS techniques such as instance bagging distributes weights among the training samples, to ensure that no individual samples has a large effect on the classification outcome [72]. This also makes the classification resistant to noise and stray changes. Additionally, feature bagging methods adds robustness against evasion attacks, by distributing weights among features. This will ensure that an adversary will have to mimic a large number of features, to cause prediction errors for the defender.

f) *Detection of changes:* Classifier models when trained on individual aspects of the feature space, as in feature bagging [123], act as a committee of independent experts who learn the same problem with different perspectives. Any changes in the expert

opinions is indicative of suspicious activities, as it indicates that a few of the features have been manipulated by an adversary. This information has been used in Query By Committee(QBC) active learning methods, to label informative samples in order to detect changes and retrain models. This disagreement was also discussed in [53, 117], as a method to detect distribution changes from unlabeled data.

g) *Flexible defense:* MCS allows for flexible implementation of defense strategies. Randomization was discussed as a disinformation strategy for MCS. Randomization can be easily implemented in MCS, by training multiple independent classifiers and randomly picking one classifier at any given time to perform the prediction. In [98], the feature set was divided into two groups and models were trained on each half. Random uniform selection of the classifiers, led to better resistance against evasion attacks. Similarly, maintaining a distribution of classifiers and picking one at random, led to better resistance to reverse engineering in [6]. MCS methods could also be used to maintain a reservoir of models, so that a degraded model can be replaced with a pre-trained well performing model, to ensure attacks are swiftly dealt with.

h) *Parallel implementation:* Ensemble methods also allow for computational advantages, as multiple models can be trained on subsets of the training data, in parallel, and can be deployed in a map reduce framework to perform scalable distributed computation [129, 130]. Learning can also be localized to data spaces, by combining clustering with classification, to learn a different pattern for differnt regions of the data [126]. In this way, simpler local patterns can be used to represent complex global patterns, in a manageable way. Incoming samples can be mapped to their clusters and then the local model can be used to perform the computation on it.

Owing to these manifold advantages, the MCS systems are promising for use in *Dynamic-Adversarial* environments, for detecting and handling changes caused by an adversary.

2.4 Towards adversary aware stream data mining

From the works on security in machine learning, it can be seen that there is a need to re-evaluate the perceived security of learning based systems, before they can be safely deployed in vulnerable applications, such as in the domain of Cybersecurity. While existing works on *Proactive* and on *Reactive* security have largely been carried out in isolation from each other, there is a need for a combined dynamic view of the problem. Proactive steps can be taken in model development, to make subsequent detection and retraining easier. Similarly, honeypots and randomization, can be used effectively to delay the onset of attacks.

The study of adversarial drift, as a specific case of the larger field of concept drift, is important to understand how adversarial behavior is affected by design decisions of the learned model. As adversarial drift is dependent on the defender's model (which the attacker is trying to reverse engineer and evade), it could be possible to train models to mislead adversary, and not just improve generalization. Also, there is a need for a framework which generates such concept drift attacks, based on the characteristics of real world datasets, so as to test the developed security measures.

Applying streaming data drift handling techniques, in adversarial environments, will require the development of drift detection techniques which can operate with scarcely labeled data. These drift detectors need to have a high attack detection rate and a low false alarm rate. This is necessary, to use human expertise efficiently and also to build trust towards the drift signaling system. Subsequent retraining and deployment is also an area which needs further evaluation.

Lastly, the use of multiple classifier systems (MCS) was seen to be ubiquitous in the adversarial learning domain. Developing improvements that can use MCS to provide dynamic security, will allow back-compatibility with the previous works and the ability to integrate different existing proactive and reactive solutions to security, under a common framework.

CHAPTER 3

VULNERABILITY OF CLASSIFICATION SYSTEMS IN ADVERSARIAL ENVIRONMENTS - A DATA DRIVEN FRAMEWORK FOR SIMULATING EXPLORATORY ATTACKS

In order to understand the security of a system, it is essential to understand its vulnerabilities. As Sun Tzu says in *The Art of War*- "*To know your Enemy, you must become your Enemy*" [131]. This chapter analyses the adversary's point of view of the machine learning system. In particular, attacks on classification systems are considered, where the attacker tries to evade the deployed classifier, by crafting samples which violate integrity of the cybersecurity system. This chapter aims to thoroughly analyze the problem of adversarial evasion, to serve as a basis for proposing and evaluating security measures for machine learning based classification systems.

3.1 Exploratory attacks on black box classifiers

Machine learning systems deployed in cybersecurity domains, operate in an adversarial environment, susceptible to various types of malicious activities. Exploratory attacks are the most common and non intrusive attacks, where client systems can submit queries (probe samples), observe the system's response and then understand its behavior [37]. The attacker in this case is considered to have the same access as that of any other user of the system, and as such can perceive the system only as a black box, to which it can submit samples, and then analyze the response on the submitted samples. In order to craft evasive samples, the adversary starts with a reconnaissance phase, wherein it tries to learn about the deployed system by probing it. Once the adversary has learned enough information,

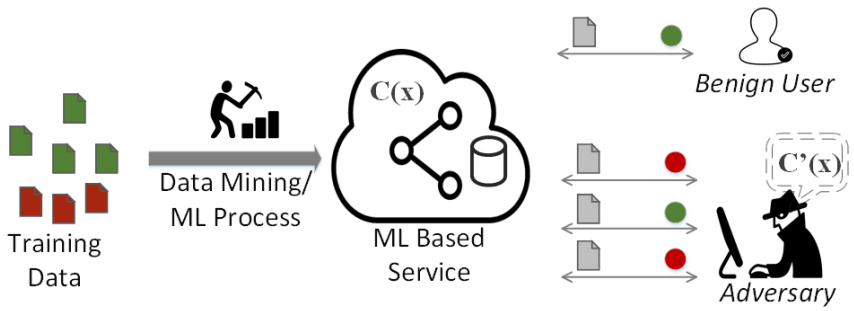
Parts of this chapter are published in [8].

it launches an attack by creating samples which the defender classifies as *Legimitate*, even though they are *Malicious*. These attacks affect the machine learning model at test time and as such fall under the category of exploratory attacks [52]. Exploratory attacks are non intrusive and unpredictable, as they affect a deployed system and can be a result of multiple adversaries, with different goals and resources, trying to break in. This chapter analyzes the vulnerability of the black box machine learning based classification systems, to these types of Exploratory - Integrity violating attacks.

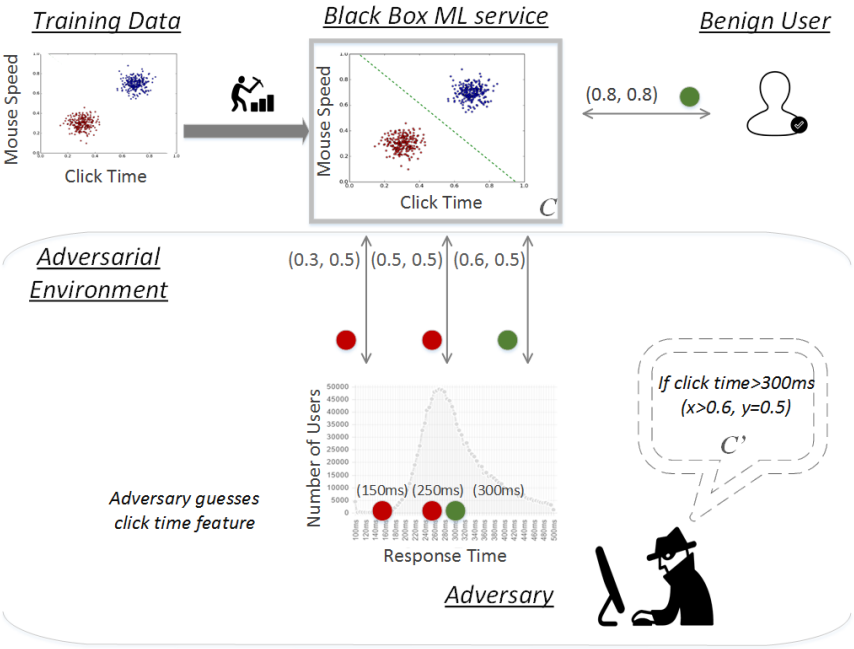
Exploratory test time attacks are illustrated in Figure 3.1a), where the deployed black box model of the defender (C), is left vulnerable to adversarial activity. The defender starts by learning from the training data and then deploying the classifier C, to provide services to the client users. Once deployed, the model C is vulnerable to adversaries, who try to learn the behavior of the defender's classifier by submitting probes as input samples, masquerading as client users. The adversary views the defender's classifier only as a black box system, capable of providing tacit *Accept/Reject* feedback on the submitted samples. An adversary, backed by the knowledge and understanding of machine learning, can use this feedback to reverse engineer the model C (as C'). It can then avoid detection on future attack samples, by accordingly perturbing the input samples. These evasion attacks are non-intrusive in nature and difficult to eliminate by traditional encryption/security techniques, because they use the same access channels as regular input samples and they see the same black box view of the system. From the classification perspective, these attacks occur at test time and are aimed at increasing the false negative rate of the model (i.e., increase the number of *Malicious* samples classified as *Legitimate* by C [1,5]). Recent works in [46], have shown that deep neural networks are vulnerable to such adversarial perturbations. Also, cloud based machine learning services (such as Amazon AWS Machine Learning[1] and Google Cloud Platform[2]), which provide APIs for accessing predictive analytics as a service, are also shown to be vulnerable to similar black box attacks in [47]. As such, a timely analysis of the adversarial vulnerabilities of classification systems is necessary.

[1] `https://aws.amazon.com/machine-learning/`

[2] `cloud.google.com/machine-learning`



(a) An adversary making probes to the black box model C, can learn it as C', using active learning.



(b) Example task of attacking behavioral CAPTCHA. Black box model C, based on Mouse Speed and Click Time features, is used to detect benign users from bots. Adversary can reverse engineer C as C', by guessing click time feature and making probes based on the human response time chart[2], using the same input channels as regular users.

Figure 3.1: Classifiers in adversarial environment, a) shows the general adversarial nature of the problem and b) shows an example considering a behavioral CAPTCHA system.

An example of the aforementioned adversarial environment is illustrated in Figure 3.1b), where a simple implementation of a behavioral mouse dynamics based CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) system, is considered. This system uses mouse movement data, to distinguish humans from bots, and they provide a user friendly way of doing this, by not forcing users to guess com-

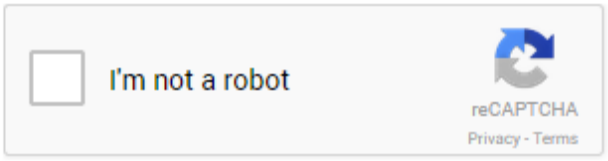


Figure 3.2: Google's reCAPTCHA system which uses mouse movement and click behavior to identify humans[1].

plicated garbled text, as in traditional text based CAPTCHA systems [31]. A prototype system developed and analyzed in [31], used machine learning and showed over 99% accuracy in detection against bots. A similar approach was used by Google, in developing its reCAPTCHA system, shown in Figure 3.2[1], which relies on a simple mouse movement and click, to provide security. In these systems, the mouse movement and click patterns are recorded and are used to distinguish if the client request originated form a human or a bot. The illustrative 2D model of Figure 3.1 b), shows a linear classifier trained on the two features of - Mouse movement speed and Click time, by the defender. An adversary, aiming to evade detection by this classifier, starts by guessing the click time as a key feature (intuitive in this setting), and then proceeds to makes probes to the black box model C, to learn its behavior. Probes are made by going through the spectrum of average reaction times for humans[2], guided by the *Accept*(green)/*Reject*(red) feedback from the CAPTCHA server. The average reaction time for humans is 272ms[2], with the full spectrum of response times shown in Figure 3.1b). A bot designed to evade this classification system, could try the different reaction times in the range of 10ms-1000ms, based on a binary search, till it lands on a critical region which is accepted by the classifier. The *Accept/Reject* feedback from the CAPTCHA server, serves as a signal to guide the search process. Once this critical region has been identified, the subsequent bots can all use this information to avoid detection, thereby rendering the classifier useless in securing the system. This is a simplistic example, since it considers only a single feature to be learned and evaded. Practical

[1] `www.google.com/recaptcha`

[2] `www.humanbenchmark.com/tests/reactiontime`

deployed classifiers tend to be more complex, non linear and multidimensional. However, the same reasoning and approach can be used to evade complex systems, as shown in this chapter.

Exploratory attacks on deployed classification systems, presents a symmetric flip side of the learning problem. Instead of learning from data to generate a model, the task of attacks is to learn about the model to generate evasive data samples [94]. As such, exploratory attacks generation is a learning process which can be analyzed from a purely data perspective, without incorporating specific domain knowledge. With this intuitive motivation, the Seed-Explore-Exploit (SEE) framework is presented in this chapter, which envisions attacks as a search problem, guided by information from the deployed classifier system. Information from the system, is obtained by crafting and submitting samples, to observe the system's response (*Accept/Reject*). In doing so, the classifier itself is considered to be a black box, which can be probed by the adversary, as shown in Figure 3.1. It is assumed that an adversary has a limited probing budget, which it can use to learn and then launch attacks. This is a practical assumption, as probes require crafting of samples which can be time taking and expensive, and also, a large number of probes could be detected and blocked out by the cybersecurity system. Based on the probing budget available, simple evasive attacks to more complex reverse engineering attacks can be launched under the proposed SEE framework.

In this chapter, vulnerability to exploratory attacks is analyzed, for binary classifiers. It is shown that, only information about the feature space (can be obtained from publications/ educated guessing) is sufficient to launch attacks, which can render the classifiers unusable in an adversarial environment. Additionally, the SEE framework is developed and experimentally evaluated to understand adversarial behavior, goals and effects. The following claims are tested in this chapter: a) Attacks can be launched on classifier, with knowledge of the feature space only, irrespective of the type of classification model of the defender, the training dataset and the domain of application, b) Classification accuracy, as a metric of predictive performance, has little significance in adversarial domains, c) The

proposed SEE framework can be used to simulate exploratory attacks on machine learning systems, ranging from simple evasion to reverse engineering of the classification model, d) Cloud based machine-learning-as-a-service providers, providing off-the-shelf predictive capability, are also not safe, as they can be attacked by non intrusive probing.

The rest of the chapter is organized as follows: Section 3.2 presents background work on simulation of exploratory attacks on classifications systems. Section 3.3 presents the SEE attack framework with two specific implementations: a) the Anchor Points attack and b) the Reverse Engineering attack. Experimental evaluation and results are presented in Section 3.4. Section 3.5 presents experiments demonstrating vulnerability of remote black box prediction services, particularly the Google Cloud Platform's Prediction service. Section 3.6 presents further discussion on the nature of attacks and the need for diversity in attack samples. Chapter summary is presented in Section 3.7.

3.2 Related work on exploratory attacks

Once a classifier model is trained and deployed in an application, it is vulnerable to exploratory attacks. These attacks are non intrusive and are aimed at gaining information about the system, which is then exploited to craft evasive samples, to circumvent the system's security. From the classification perspective, these attacks occur at test time and are aimed at increasing the false negative rate of the model (i.e., increase the number of *Malicious* samples classified as *Legitimate* [1]). These attacks are universal and are difficult to eliminate by traditional encryption/security techniques, because they use the same access channels as regular input samples and see the same blackbox view of the system.

Exploratory attacks are classified as being either: *Targeted* or *Indiscriminate*, based on the specificity of the attacks [1]. Targeted attacks aims at modifying a specific set of malicious input samples, minimally, to disguise them as legitimate. Indiscriminate attacks are more general in their goals, as they aim to produce any sample which will result in the defender's model causing a false negative. Most work on exploratory attacks are concentrated on the targeted case, considering it as a constrained form of indiscriminate attacks,

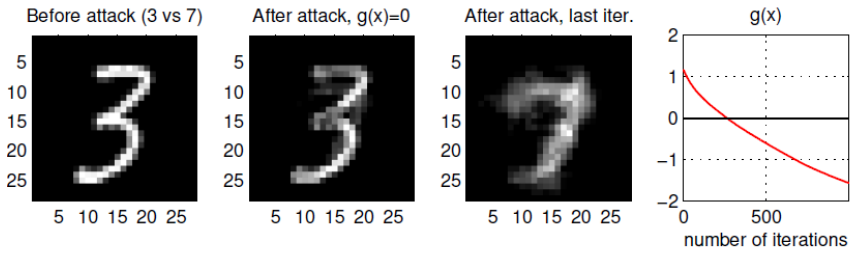


Figure 3.3: Gradient descent evasion attack over 500 iterations. Left- Initial image of digit 3, Center- Image which first gets classified as 7, Right- Image after 500 iteration [5].

with the goal of starting with a malicious sample and making minimal modifications to them, to avoid detection [5, 56, 92, 132, 133]. This idea was formalized in [51], where the Minimal Adversarial Cost (MAC) of a genre of classifiers was introduced, to denote the ease with which classifiers of a particular type can be evaded. The hardness of evasion was given in term of the number of probes needed, to obtain a low cost evasive sample. A sample's cost was given in terms of the accuracy of evasion and the distance from the original targeted sample from which it was evolved. A classifier is easy to evade if making a few optimal modifications to a set of samples, results in a high accuracy of evasion. Work in [134], shows that linear and convex inducing classifier are all vulnerable to probing based attacks, and [135] presents efficient probing strategies to carry out these attacks.

Particular strategies developed for performing exploratory attacks vary based on the amount of information available to the adversary, with a broad classification presented in [6] as: a) Evasion attacks and b) Reverse Engineering attacks. Evasion attacks are used when limited information about the system is available, such as a few legitimate samples only. These legitimate samples are exploited by masking techniques such as- mimicking and spoofing, which masquerade malicious content within the legitimate samples. The mimicry attack was presented in [53], where the Mimicus tool[1] was developed, to implement evasion attacks on pdf documents by hiding malicious code within benign documents. The good words attacks on spam emails uses a similar technique. A spam email is inserted with benign looking words, to evade detection. For example, the word *SALE* when replaced with *S4LE*

[1] `www.github.com/srndic/mimicus/blob/master/mimicus/attacks/mimicry.py`

looks visually similar and can avoid being flagged by the detector [56]. Similarly, spoofing attacks are common in biometrics [136] and for phishing websites [36], where visual similarity can be achieved with totally different content. A general purpose - domain independent technique for evasion was presented in [133]. Here, using genetic programming, variants of a set of malicious samples were generated, as per a monotonically increasing fitness function which denoted success of evasion. This is an attractive technique due to its generic approach, but limited probing budgets and lack of a graded fitness function, are some of its practical limitations. In the presence of a higher probing budget or specific information about the defender's classifier model, the gradient descent evasion attack of [5], can be used. This attacks relies on knowing the exact classifier function used by the defender, or the ability to reverse engineer it using a sufficient number of probes. Once information about the classifier is known, the attack uses a gradient descent strategy to find an optimal low cost evasion sample for the classifier. Search strategies were developed for a wide range of classifiers with differentiable decision functions, including neural networks, non-linear support vector machines, one class classifiers and for classifiers operating in discrete feature spaces. An illustration of the gradient descent attacks for masquerading a sample is shown in Figure 3.3, where the image 3 is modified to be classified as 7 over 500 iterations of gradient descent.

Reverse engineering the defender's model provides avenues for sophisticated exploratory attacks, as it exposes features important to the classifier, to be used for mimicry attacks or large scale indiscriminate attacks. Perfect reverse engineering is not needed, as an adversary is interested only in identifying the portion of the data space which will be classified as legitimate. Reverse engineering was first employed in [51], where a sign witness test was used to see if a particular feature had a positive or negative impact on the decision. Reverse engineering of a decision tree classifier, as a symmetric model for defender and adversary, was presented in [69]. [133] used genetic programming as a general purpose reverse engineering tool, under the assumption of known training data distribution and feature construction algorithm. The genetic programming output, because of its intu-

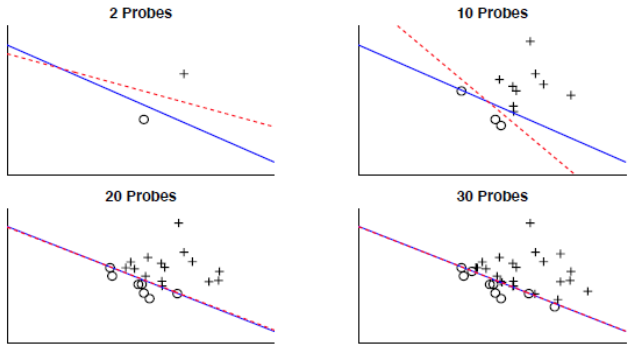


Figure 3.4: Illustration of reverse engineering task. Blue line marks defender's boundary and Red marks the adversary's reverse engineered model [6].

itive tree structure, was then used to guide evasion attacks on Intrusion Detection Systems. The idea of reverse engineering was linked to that of active learning via query generation in [6], where the robustness of SVM to reverse engineering is tested using active learning techniques of random sampling, uncertainty sampling and selective sampling. Samples were generated by iteratively generating random samples between two known seed samples, and then selecting samples based on the aforementioned active learning techniques. A sample illustrative reverse engineering task is shown in Figure 3.4.

Recent works on attacking black box classifiers have been proposed in [46, 47, 100, 137]. In [46], the vulnerability of deep neural networks (DNN) was demonstrated. An adversary starts by probing the black box classifier, and then trains a substitute DNN, as its perception of the original data space. From the trained model, adversarial samples are crafted such that the modifications needed for misclassification are minimized. The fast gradient sign method [138] and the Jacobian based source-target misclassification attacks [46], were used to generate adversarial samples. The attack strategy was shown to generate a mis-classification rate of >88% on Amazon Web Service's machine learning models[1] and the Google Cloud Platform's prediction model[2]. The membership inference attack was developed in [100], as a strategy to attack black box classification models. Leakage of training data was demonstrated, by analyzing the differences in the model's predictions on

[1] `https://aws.amazon.com/machine-learning/`

[2] `cloud.google.com/machine-learning`

inputs that it trained on, versus the ones it did not train on. This was done by means of training a set of surrogate inference models, to analyze the target black box model. The implications of privacy were analyzed, by demonstrating the leakage on a hospital discharge dataset, while using models trained on the Google[2] and Amazon's[1] cloud prediction services. Model extraction attacks were presented in [47], which provide a mechanism for reverse engineering models trained using *ML-as-a-service* providers. A generic equation solving methodology was introduced for models of logistic regression, and a path-finding algorithm was introduced for reverse engineering of decision tree models. For other class of non linear models, like neural networks and non linear SVM, a retraining approach was discussed. Here, the adversary collects samples using active learning, and then retrains a local model as a surrogate to the targeted models. These models are then used as white box oracles for attacking using the sign witness test proposed in [51]. It was shown that a model which returns only binary output, as opposed to confidence values, is more secure to model inversion attacks. However, further analysis of this claim was left as future work.

In the above mentioned works of targeted-exploratory attacks, it is assumed that if an evasion is expensive (far from the original malicious sample), the adversary will give up. The above techniques are not designed for a determined adversary, who is willing to launch indiscriminate attacks. An adversary who wants to launch an indiscriminate attack will not bother with the near optimal evasion problem [134]. These type of attacks have been largely ignored, with the only mention we found was in [90], where it is termed - the free range attack, as an adversary is free to move about in the data space. In such attacks, the adversary will first analyze the vulnerabilities of the model, looking for prediction blind spots, before attempting an attack. Analyzing performance of models under such attack scenarios is essential to understanding its vulnerabilities in a more general real world situation, where all types of attacks are possible. Also, most recent methodologies develop attacks as an experimental tool to test their safety mechanisms, there are a few works [5,43,46,47,69,100,132,137,139], which have attempted to study the attack generation process itself. Our proposed work analyzes *Indiscriminate-Exploratory-Integrity* violating

attacks, in a data driven framework, with different adversarial goals and their impact on classification security. We analyze the attacks from an adversary's point of view, considering the adversarial samples generation process, so as to understand the vulnerabilities of classifiers and to motivate the development of secure machine learning architectures.

3.3 Framework for simulating data driven attacks on classifiers

The proposed Seed-Explore-Exploit (SEE) framework, for generating exploratory attacks, is a machine learning driven framework which operates under the assumption of minimal shared knowledge between the adversary and the defender. Only the feature space information is shared between the two parties, as both operate on the same features and are aware of: the number, type and range of features. All other information, about training data and learned model, is kept hidden by the defender [2]. Both the adversary and the defender are assumed to be capable data scientists, who are equipped with the tools of machine learning and use a data driven approach to best suit their goals. The SEE framework is presented in Section 3.3.1. Two specific attack strategies developed using the SEE framework, the Anchor Points attacks (AP) and the Reverse Engineering attacks (RE), are presented in Section 3.3.2 and Section 3.3.3, respectively.

3.3.1 The Seed-Explore-Exploit (SEE) framework

The SEE framework provides guideline for implementing a search based methodology, to generate attacks against classification systems. The idea of Exploration-Exploitation is common in search based optimization techniques, where the goal is to learn the data space and then emphasize only on the promising directions [140]. Formally, the defender's classifier model C, is considered to divide the entire data space into the *Legitimate* class and the *Malicious* class, with the aim of maximizing prediction metrics, such as accuracy. A data driven adversary, will gather information about C by using probing samples, which it crafts to best *explore* the behavior of C. This information is then *exploited* by the adversary to generate a set of attack samples D'_{Attack}, which increases C's false negative rate. The

Figure 3.5: Overview of the SEE framework.

formal model of an adversary based on its knowledge, goals and resources [66] is presented below:

- **Knowledge**: Adversary is aware of the number, type and range of feature values. These could be approximated from publications, publicly available case studies or by educated guessing [2]. For example, in case of spam emails, the domain is the dictionary of words, which is publicly well known. No information about classifier type or training data is known.

- **Goals**: Adversary intends to produce false negatives for C. Additionally, the attacker wants to avoid being detected and stopped by simple blacklisting techniques [141]. This is ensured by generating attack samples set (D'_{Attack}) with high diversity. While, repeating a single attack sample, over and over, leads to ensured false negative, such attacks are easily detected and stopped. Diversity ensures variability in attack samples. Only serious attackers, who aim to force redesign of system, are considered.

- **Resources**: The attacker has access to defense system, as a client user. The attacker can submit samples to be labeled by C, up to a probing budget $B_{Explore}$, without being detected.

Based on this model of the adversary, the SEE framework is presented in Figure 3.5. The framework begins with a seed phase, where it receives a small set of samples from either class. In the exploration phase, the attacker performs systematic search inorder to obtain maximum information from a limited probing budget. Once the exploration phase is complete, the attacker will launch an exploratory attack against C, where it will send the set of attack samples (D'_{Attack}) to the classifier. The adversary's goal is to have a high false negative rate for the D'_{Attack} set and to make it have high variability. An example of the

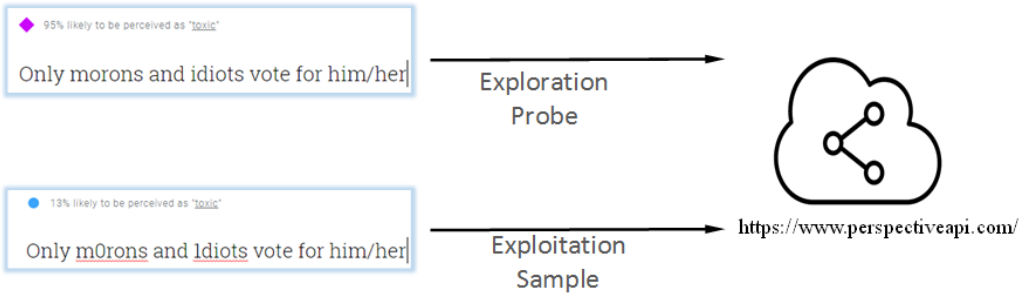


Figure 3.6: Depiction of Exploration and Exploitation steps in evading the Perspective API.[1]

exploration and exploitation operations is illustrated in Figure 3.6. The Perspective API[1], which measures toxicity of online comments, is used for the evaluation. The API provides a simple text box interface to an end user, to enter the comments and assess its toxicity. As such it is a black box and remote ML based system. In Figure 3.6, it is seen that an adversary realizing the toxicity of the keywords *'morons'* and *'iditos'*, is able to modify them to be *'m0ron'* and *'1diots'*, respectively, to enable evasion. These modifications retain the original intent of the comment, as humans can infer these modifications due to their visual similarities, but a text analysis based ML model is easily fooled by such perturbations. The SEE framework provides a systematic methodology to enable exploration and exploitation, on black box classifiers. The specific steps of the framework are explained below:

1. **Seed**: An attack starts with a seed phase, where it acquires a legitimate sample (and a malicious sample), to form the seed set D'_{Seed}. This seed sample can be acquired either by random sampling in the feature space, by guessing a few of the feature values, or from an external data source of a comparable application. Random guessing is possible only when the prediction surface is not too restrictive and when the attack domain is small (difficult in case of one class classifiers, which occupy a small portion of the feature space) [37]. However, in most cases, acquiring a sample by guessing or from an external source is not difficult. For a behavioral CAPTCHA system [31], the adversary could solve it themselves and record their movements to serve as seed.

[1] `https://www.perspectiveapi.com/`

Similarly, picking any email from ones personal inbox would be a functional legitimate seed.

2. **Explore**: The seed set D'_{Seed}, serves as a basis for starting the exploration of the feature space. Exploration is a reconnaissance task, where the goal is to obtain maximum diverse information, to understand the coverage and extent of the space of legitimately classified samples, without being detected. As such, a budget constraint $B_{Explore}$ is introduced, to limit the number of times the defense model C can be queried without being thwarted or detected. To avoid detection, it is natural that the adversary will spread out these attacks over time and space, in which case the $B_{Explore}$ term denotes the time/resources an attacker has at its disposal. Exploration can be performed to either directly reverse engineer the model's boundary, or to obtain a set of benign samples which serve as ground truth attack points. The exploration phase results in a set of labeled samples $D'_{Explore}$, and the goal of the adversary is to best choose this set based on its strategy.

3. **Exploit**: The information gathered from the Exploration phase, is used in this phase to generate a set of attack samples- the D'_{Attack} set, to be used by the adversary in their current attack campaign. D'_{Attack} should have a high evasion rate and high diversity to be effective.

The SEE framework provides a generic way of defining attacks on machine learning systems. It serves as a blueprint, based on which a data driven attack can be generated on a classification system. Specific instantiations of the Seed, Explore and Exploit phases can be developed, to suit one's needs or simulation goals. We provide two specific strategies based on the SEE framework: the Anchor Points (AP) approach and the Reverse Engineering (RE) approach, which embody different levels of sophistication of the adversary, the amount of damage they wish to cause, and the $B_{Explore}$ they have at their disposal.

3.3.2 The Anchor Points (AP) attack

The Anchor Points attack is suited for adversaries which have a limited probing budget ($B_{Explore}$), and have a primary aim of generating evasive samples for immediate benefits. An example would be zero day exploits [142], where the adversary wants to quickly exploit a new found vulnerability, before it's fixed. From a data perspective, these attacks start by obtaining a set of diverse samples, classified as legitimate by the defender's model C. These points, called Anchor Points, then serve as gold standards, denoting the space of legitimately classified data. Attacks are generated by exploiting this ground truth and by generating attacks around them. For example, an adversary would start with a set of benign personal emails and then generate attacks by modifying certain words in these emails so as to impart malice. By the nature of these attacks, they could be thwarted by blacklists, which use approximate matching [141]. Nevertheless, they serve as a quick way to generate attacks, in complex classification spaces. The implementation of these attacks as per the SEE framework is presented here:

1. **Seed**: The AP attack needs only a single legitimate sample as seed, to serve as the initial anchor point. This forms the seed set D'_{Seed}.

2. **Explore**: The exploration phase proceeds by generating variants of the seed, by performing a neighborhood search, to produce samples which would have a high chance of being classified as legitimate, as shown in Algorithm 3.1. The exploration phase generates the set of anchor points $D'_{Explore}$, to be used in the exploitation phase. The exploration phase is described in Algorithm 3.1, and is a radius based incremental neighborhood search technique, around the seed samples, guided by the feedback from the black box model C. Diversity of search is maintained by dynamically adjusting the search radius (R_{Neigh}), based on the amount of ground truth obtained so far. The parameter R_{Neigh} denotes the neighborhood radius to be used in the exploration, and is adaptively fixed to ensure a balance between accuracy and diversity. This is done by systematically reducing the radius, if the number of legitimate samples discovered

Algorithm 3.1: AP- Exploration Phase

Input : Seed Set D'_{Seed}, Defender black box model C. *Parameters*: Exploration budget $B_{Explore}$, Radius neighborhood- [$R_{Neigh-min}$, $R_{Neigh-max}$]

Output: Explored Anchor Points Set $D'_{Explore}$

```
1  D'_Explore ← D'_Seed
2  count_ legitimate=0
3  for  i = 1 .. B_Explore do
4      sample_i ← Select random sample from D'_Explore
5      R_Neigh = (R_Neigh−max − R_Neigh−min) ∗ (count_legitimate/i) + R_Neigh−min
6                                              ▷ Dynamic neighborhood search
7      perturbed_ sample← Perturb(sample_i , R_Neigh)
8      if C.predict(perturbed_ sample) is Legitimate then
9          D'_Explore ∪ perturbed_ sample
10         count_ legitimate ++
11 Procedure Perturb(sample, R_Neigh)
12   return sample+=random(mean=0, std=R_Neigh)
```

while probing drops, and vice-versa, as shown in Equation 3.1. For a 0-1 normalization scale, a $R_{Neigh-min}$ of 0.1 and a $R_{Neigh-max}$ of 0.5 is reasonable.

$$R_{Neigh} = (R_{Neigh-max} - R_{Neigh-min}) * \frac{\#legitimate \quad samples}{\#samples \quad probed \quad so \quad far} + R_{Neigh-min} \tag{3.1}$$

The exploration phase begins with the initial set of seed samples D'_{Seed}. It then populates the anchor points set $D'_{Explore}$, by generating additional samples and verifying if they also belong to the legitimate class, as seen in Algorithm 3.1. Additional points are generated by perturbing the already identified benign set of points. Perturbation is done by moving the sample randomly within the radius R_{Neigh}, along every dimension. This radius is dynamically adjusted based on the number of correct samples, as shown in Line 5. The algorithm returns $D'_{Explore}$, which forms the set of explored points, which were correctly classified by C.

3. **Exploit**: The exploitation phase uses the $D'_{Explore}$ to generate the D'_{Attack} set, which form the current campaign's attack samples. The AP method combines the following

two techniques to ensure high diversity and accuracy of attack samples.

(a) *Simple perturbation*: This technique (Line 5) uses perturbation similar to the *Perturb()* procedure in Algorithm 3.1. However, the size of neighborhood is reduced from the exploration phase, to ensure attacks points are close to the ground truth anchor points. The exploitation radius is denoted by $R_{Exploit}$, and is used to generate perturbed samples to populate the D'_{Attack} set.

(b) *Convex combination*: In this technique (Line 8), the attack samples are generated by performing a convex combination of samples randomly drawn from $D'_{Explore}$. This has the advantage of being able to produce high accuracy of attack rate. But the diversity of the samples is limited, as the samples start to get cluttered, in case where the $D'_{Explore}$ is small in size.

The exploitation phase combines both these techniques as shown in Algorithm 3.2. The algorithm picks two random points from $D'_{Explore}$, and then perturbs them by adding a random generated number in the range [0, $R_{Exploit}$], along every dimension. The perturbed samples are then combined using a convex combination, as shown in Line 8.

The final output of Algorithm 3.2, comprises of the D'_{Attack} set which is submitted as an attack to model C. This culminates the adversarial round of the attack-defense cycle and one iteration of the SEE approach. Usually, the size of D'_{Attack} is much larger than $B_{Explore}$, to justify budget expenditure.

The Anchor Points (AP) attacks is illustrated on a synthetic 2D dataset in Figure 3.7. The initial defender's classifier is probed using the radius search technique, with a neighborhood radius of R_{Neigh}. The resulting set of exploration samples which are classified as *Legitimate* by C, forms the anchor points set, to perform the exploitation. The combination of random perturbation and convex combination is illustrated in Figure 3.7. The final set of red points, is the attack set D'_{Attack}, which evades the classifier C.

Algorithm 3.2: AP- Exploitation Phase

Input : Anchor Points Set $D'_{Explore}$, Number of attacks N_{Attack}, Radius of Exploitation $R_{Exploit}$
Output: Attacks set D'_{Attack}

$D'_{Attack} \leftarrow []$
for $i = 1$ .. N_{Attack} **do**
  $sample_1 \leftarrow$ Select random sample from $D'_{Explore}$
  $sample_2 \leftarrow$ Select random sample from $D'_{Explore}$
  perturbed_sample1 $\leftarrow Perturb(sample_1, R_{Exploit})$ ▷ **Random perturbation**
  perturbed_sample2 $\leftarrow Perturb(sample_2, R_{Exploit})$
  $\lambda = random \quad in \quad [0, 1]$
  $attack\_sample_i \leftarrow$ perturbed_sample1 * λ + (1-λ)*perturbed_sample2
  ▷ **Convex combination**
  $D'_{Attack} \cup attack\_sample_i$
Procedure Perturb(sample, $R_{Exploit}$)
  return sample+=random(mean=0, std=$R_{Exploit}$)

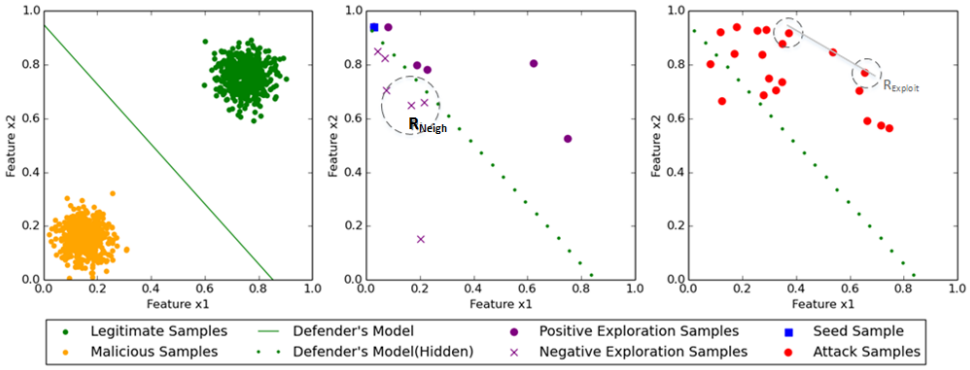


Figure 3.7: Illustration of AP attacks on 2D synthetic data. *(Left - Right)*: The defender's model from its training data. The Exploration phase depicting the seed (blue) and the anchor points samples (purple). The Exploitation attack phase samples (red) generated based on the anchor points.

The performance of the Anchor Points approach is largely dependent on probes collected in the initial seed and exploration phase, and as such maintaining diversity in search is key. Larger coverage ensures more diversity and flexibility in the attack phase. However, since this technique is suited as an adhoc quick exploit, its efficacy lies on quick impact and spread before the defender has time to respond.

3.3.3 The Reverse Engineering (RE) attack

In case of a sophisticated attacker, with a large $B_{Explore}$, direct reverse engineering of the classification boundary is more advantageous. It helps understanding the entire prediction landscape quicker and allows launching of optimal evasion attacks based on gradient descent, as was described in [5]. Reverse engineering in itself could be the end goal in some cases, as it provides information about features and their importance to the classification task [51]. Reverse engineering also allows the launching of large scale availability attacks, which can cause retraining using the same features impossible [114]. However, a reverse engineering approach is affected by the type of model being used for C, the dimensionality of the data space and the number of exploration probes possible. Nevertheless, the goal of the attacker is not to exactly fit the decision boundary, but to generate highly accurate and diverse attacks. In accordance with this, a linear approximation of a nonlinear boundary and a partial reverse engineering attempt should be sufficient. A linear approximation is not a major limitation due to the following reasons: i) most large scale and dynamic data mining systems resort to linear models, to prevent over fitting of the data and to save training time on the fly [143], and ii) a linear approximation of a nonlinear boundary would still provide some degree of accuracy, which should be sufficient to launch a massive attack to compensate for reduced accuracy. Based on this intuition, a reverse engineering strategy to learn a linear classifier from the probes is developed here.

Since effective reverse engineering relies on the availability of informative samples, it is necessary to use the available probing budget efficiently. Random sampling often results in wasted probes which add no additional information, and as such is not a preferred choice in this scenario. The query synthesis strategy of [144], generates samples close to the decision boundary and spreads the samples along the boundary, to provide a good understanding of the decision landscape. While query synthesis was used for active labeling of samples in [144], we modify it to be used for the task of reverse engineering, here. Based on the queries(probes), an attacker learns a surrogate classifier C', which is an approximation of C, to be used for exploitation in the attack phase. The SEE implementation of the RE

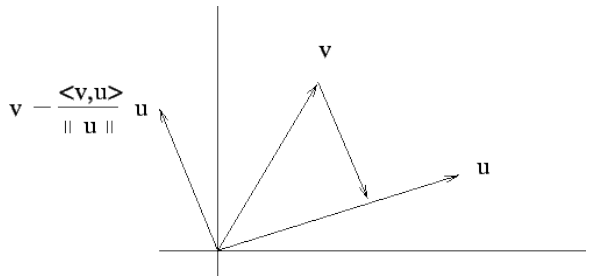


Figure 3.8: The Gram-Schmidt process of orthonormalization[1]. Randomly chosen vector v is made orthonormal to vector u.

approach is presented here.

1. **Seed**: The seed set consists of one legitimate and one malicious sample.

2. **Explore**: The exploration phase uses the query synthesis technique of [144], to learn a surrogate classifier C', as shown in Algorithm 3.3. The algorithm uses the Gram-Schmidt process to generate orthonormal points, near the midpoint of any two randomly selected points of opposite classes. In every iteration, a Legitimate X_L and a Malicious sample X_M, are selected from the already probed samples. Then a randomly selected vector X_R, is made orthonormal to the line joining the vectors X_L and X_M. This orthonormal point is then made equal to the magnitude λ_i (selected randomly between 0 and λ) and moved to the midpoint of the line. This process allows selection of points which will have high informative content and spread across the boundary for better reverse engineering a model.

   The orthonormalization process is depicted in Figure 3.8[1], where the vector v, selected randomly, is made orthonormal to u, using the Gram-Schmidt process. The projection of the vector v onto u, denoted by $Proj_u v$, is computed as $< v, u > / \|u\|$. The resulting orthogonal unit vector is given as $v - Proj_u v$, which forms the orthonormal basis for the vectors v and u.

   Algorithm 3.3 allows for effective reverse engineering, in a principled way. In case of complex class boundaries, this approach will only provide an approximation of the

[1] http://nptel.ac.in/courses/122104018/node49.html

Algorithm 3.3: Reverse Engineering by query synthesis

Input : Seed Set D'_{Seed}, Defender model black box C, Exploration budget $B_{Explore}$, Magnitude λ

Output: Exploration query Set $D'_{Explore}$, Surrogate classifier C'

$D'_{Explore\_L}$=Legitimate samples from D'_{Seed}
$D'_{Explore\_M}$=Malicious samples from D'_{Seed}
for *i = 1 .. $B_{Explore}$* **do**
    $X_L \leftarrow$ Select random samples from $D'_{Explore\_L}$
    $X_M \leftarrow$ Select random samples from $D'_{Explore\_M}$
    $X_0 = X_L - X_M$
    Generate random vector X_R
    $X_R = X_R - \frac{<X_R, X_0>}{<X_0, X_0>} * X_0$
        ▷ Using Gram-Schmidt process to make X_R orthogonal to X_0
    $\lambda_i = random(0, \lambda)$
    $X_R = \frac{\lambda_i}{norm(X_R)} * X_R$
        ▷ Set magnitude of orthogonal midperpendicular vector
    $X_S = X_R + (X_L + X_M)/2$
        ▷ Set X_R to midpoint
    if *C.predict(X_S) is Legitimate* **then**
        $D'_{Explore\_L} \cup X_S$
    else
        $D'_{Explore\_M} \cup X_S$
    $D'_{Explore} = D'_{Explore\_L} \cup D'_{Explore\_M}$
    Train C' using $D'_{Explore}$
        ▷ Training can be based on linear classifier of choice

surface. As stated, the goal here is to have a rough understanding of the area of the data space which is classified as Legitimate. The surrogate model C' obtained, is used by the exploitation phase to generate attacks.

3. **Exploit**: The Reverse Engineered model C', can be used to generate attacks with high diversity and accuracy, provided the reverse engineering was effective. Ideally, a set of random points can be generated and verified against the surrogate classifier C', before adding them to the attack set - D'_{Attack}. However, a practical and effective way would be to use the exploration set $D'_{Explore}$ of Algorithm 3.3, as a seed set to generate a set of anchor points as in Algorithm 3.1, with the exception that we use C' instead of C. This process essentially allows us to make a large number of probes at

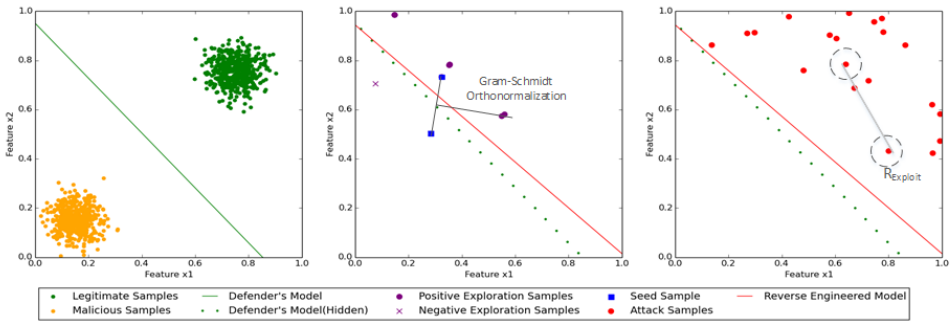


Figure 3.9: Illustration of RE attacks on 2D synthetic data.*(Left - Right)*: The defender's model based on training data. The Exploration phase depicting reverse engineering(red) using the Gram-Schmidt orthonormalization process. The Exploitation attack phase samples generated after validation from the surrogate classifier (red samples).

theoretically 0 cost. The anchor points can then be perturbed to generate more attack samples. The perturbation and search of neighborhood points can be done using a large neighborhood radius(R_{Neigh}), as we can now verify samples against C', before sending them out. The increased R_{Neigh}, allows samples to have high diversity at a high confidence, as opposed to the AP approach, where radius needs to be constrained to stay close to the ground truth points.

The Reverse Engineering (RE) attacks is illustrated on a synthetic 2D dataset in Figure 3.9. The initial defender's classifier is probed using the Gram-Schmidt technique, to learn a surrogate model (red) from the original defender's model (green). The surrogate classifier is used to provide additonal validation, before submitting the attack samples (red), which are used to evade C.

Both the AP and RE can attack classification models effectively. The fundamental difference between the two approaches is that reverse engineering places its confidence on its understanding of the separating function while the former approach places its confidence only on the obtained anchor points.

Note on query synthesis and binary features:

Since most of these approaches rely on numerical operations on data, their applicability to binary feature data might seem limited. However, this is not the case. The essential task of neighborhood point's generation (in the AP and RE approaches), can be applied

to categorical data by selecting a fraction R_{Neigh} of dimensions and flipping their feature values. Similarly the task of convex combination in Algorithm 3.2 and that of finding the orthogonal point in Algorithm 3.3, can be performed by finding the nearest feasible point in the binary vector space to the generated points. This will allow approximation of the above mentioned approaches when applied to datasets with binary feature values, as is common in spam classification applications. Additionally, similar approaches can be used in the task of generating the actual query which is to be submitted to the system. The above query synthesis approaches do not consider how the numeric values generated can be related to an actual real world probe which is submitted to the blackbox model. As suggested in [48], the task of generating the actual input from the data sample, is a challenging one. However, the real world input creation can be approximated by using the generated sample, as a suggestion, and by closely mimicking its properties.

3.4 Experimental analysis of the SEE framework

This section presents experimental analysis of the AP and the RE attacks, on classifiers trained on seven real world datasets. Additionally evaluation on three datasets from cybersecurity domains is presented to demonstrate the vulnerabilities of machine learning systems in domains where its primary purpose is to ensure security. These experiments are presented from an adversary's point of view and the metrics used for evaluation: Accuracy and Diversity, are used for determining efficacy of an attack launched by the attacker. Section 3.4.1 presents the datasets, metrics and experimental protocol for performing the experiments. Results on real and synthetic datasets and sensitivity analysis is presented in Section 3.4.2.

3.4.1 Experimental methods: Datasets, Metrics and Setup

3.4.1.1 Description of datasets

Experimentation is performed using 10 real world datasets, the details of which are presented in Table 3.1. The first 7 datasets were chosen from the UCI machine learning

TABLE 3.1

Description of datasets used for experimentation of SEE framework.

Dataset	#Instances	#Dimensions
Digits08	1500	16
Credit	1000	61
Cancer	699	10
Qsar	1055	41
Sonar	208	60
Theorem	3060	51
Diabetes	768	8
Spambase	4600	57
KDD99	494021	41
CAPTCHA	1885	26

repository [145] and are popularly used for binary classification tasks, in literature. These datasets do not traditionally embody any security risks, but are attractive to evaluate the classification evasion results, as they represent different application domains and types of data. The Spambase[1] [145], KDD99[2] [145] and the CAPTHCA [31] datasets are binary classification tasks, which represent 3 different cybersecurity domains that use machine learning as a core technique. The Spambase dataset, contains data about spam emails (such as fraud schemes, ads, etc) and legitimate personal and work emails. The KDD99 dataset is a popular network intrusion detection dataset, which is aimed at developing a model to classify normal connections from different classes of attack connections. The CAPTCHA dataset was developed recently at the University of Louisville [31], for the task of blocking bots from human users, based on their mouse movement patterns, while solving a visual image based behavioral CAPTCHA puzzle. The datasets contains mouse movement and click patterns from 16 different human users, and bot data, from an external agent aimed at subverting this system, by simulating human behavior.

All datasets were preprocessed by first reducing them to a binary class problem, if it was multi-class originally. The Digits dataset was reduced to have samples of the digit 0 and 8 only, KDD99 was reduced to represent only two classes - attacks and normal. The

[1] `https://archive.ics.uci.edu/ml/datasets/Spambase`

[2] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

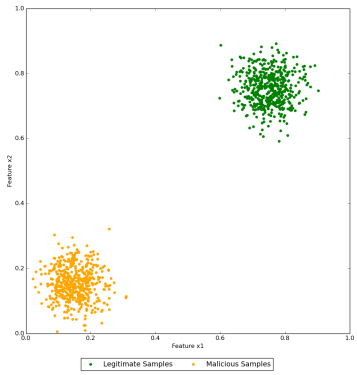


Figure 3.10: Synthetic 2D dataset generated for illustrative experimentation. The Legitimate samples are in green and Malicious samples are in orange.

dataset was then converted to contain only numerical values, by transforming categorical and nominal features to binary variables. The resulting number of features is shown in Table 3.1. The data is then cleaned by filling in missing values by feature means and by normalizing the range of each feature to [0,1]. Instances were shuffled to remove any bias due to inherent concept drift. In all datasets, the class label 1 is taken to be the *Malicious* class and 0 is taken as the *Legitimate* class, as a convention.

A 2D synthetic data was developed to illustrate the experimental behavior of the different approaches, in order to better understand their internal working, as shown in Figure 3.10. This 2D dataset has 1000 samples equally balanced between the Malicious class (orange) and the Legitimate class (green).

3.4.1.2 Adversary's metrics for attack quality

In this section, we analyze the effectiveness of attacks from an adversary's perspective. We do this to better understand the adversarial perspective and capabilities, to be able to better design countermeasures against them. An adversary aiming to create maximum impact through its attack samples, needs to make the D'_{Attack} set- *Accurate* and *Diverse*. Accuracy ensures that the attack samples will cause an increase in the false negative rate of the defender's model C, while diversity ensures variability in attacks, so that it can go unnoticed for a long time. Based on this intuition, we define 4 quality metrics which an

attacker can use to measure/compare effectiveness of different attack strategies. The Effective Attack Rate (EAR) measures the accuracy of the attacks, and the 3 metrics: Standard deviation of attacks (σ_{EA}), K-Nearest Neighbor distance (KNN-dist) and the Minimum Spanning Tree distance (MST-dist), collectively represent diversity of the attack samples. These metrics are based on the following definition of Effective Attacks (EA):

$$EA \quad = \quad \left\{x: \quad C(x) = Legitimate \quad \wedge \quad x \in D'_{Attack}\right\} \tag{3.2}$$

An attack is effective only is it is classified as Legitimate by the defender model C, as per Equation 3.2. The 4 adversary quality metrics compute quality over the set EA of effective attacks.

a) *Effective Attack Rate (EAR)*: This is a measure of the accuracy of the attacks and it is the ratio of attack samples which successfully evade the defenders classifier, given by Equation 3.3. A value of 1 denotes perfect evasion.

$$EAR \quad = \quad \frac{|EA|}{\left|D'_{Attack}\right|} \tag{3.3}$$

b) *Standard deviation of effective attacks (σ_{EA})*: This is a measure of diversity, which computes the spread of data around its mean, given by Equation 3.4.

$$\sigma_{EA} \quad = \quad \sqrt{\frac{1}{|EA-1|}\sum_{x \epsilon EA}\left(x-\mu_{EA}\right)^2} \tag{3.4}$$

Where, μ_{EA} indicates the Euclidean mean of samples in the effective attack set EA. A large value of σ_{EA} indicates that the data is more spread out and covers a larger area of the data space, therefore having high diversity.

c) *K-Nearest Neighbor distance of effective attacks($KNN-dist_{EA}$)*: This measure of diversity computes local density information of the data samples (motivated by [146]). The measure is computed by finding the average distance of the K-nearest neighbors of a sample, for all samples and then averaging this value, as given by Equation 3.5.

$$KNN-dist_{EA} \quad = \quad \frac{\sum_{x \epsilon EA}\sum_{i=1}^{K} \quad dist(x, NN_i(x))}{K. \quad |EA|} \tag{3.5}$$

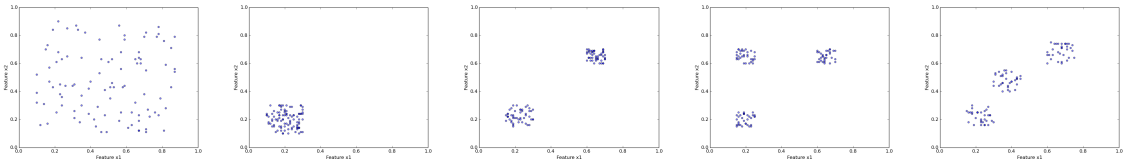
Where, the *dist(.)* function computed Euclidean distance between two vectors. and $NN_i(x)$, gives the i^{th} nearest neighbor of a the sample x. A higher value of KNN-dist, indicates that data are relatively far from each other and that every sample is in a locally sparse region of space, indicating high diversity. A value of K=5 is chosen for experimentation.

d) *Minimum Spanning Tree distance of effective attacks ($MST - dist_{EA}$)*: This is also a measure of diversity,which is computed by finding the length of the minimum spanning tree over the set of EA samples, as per Equation 3.6 [147]. This is a measure which promotes ectropy or collocation of points, in an attempt to obtain a more global uniform and diverse spread of samples. This is especially useful in recognizing multiple locally dense clusters which are far from each other.

$$MST - dist_{EA} \quad = \quad \frac{length(MST(EA))}{|EA| - 1} \tag{3.6}$$

The MST measure computes cluster separation only once, as opposed to pairwise distance metrics which calculate distance between one point and every other point. Thus the MST provides a better sense of global diversity, by allowing sub groups of data to have less diversity. A high value of MST-distance, will indicate high diversity.

Diversity is computed using the three metrics of : σ, $KNN{-}dist$, $MST{-}dist$. While each of these metrics capture variability in the dataset, the have different characteristics, being sensitive to different data distributions. Standard deviation captures the overall spread of data and is severely affected by outliers. A high spread of data leads to a higher σ, as shown in Figure 3.11 a) and b), where a) has a σ of 0.228 as it covers the entire feature space, while b) being concentrated in a small portion of the space has a σ of 0.058. Standard deviation captures this global notion of spread effectively, but fails at capturing local data characteristics. The Figure 3.11 c), represents data cluttered around two clusters and it has a similar σ (=0.218) as Figure 3.11 a) (=0.228). These datasets are nearly identical from the σ perspective, but their difference can be effectively caught by observing the KNN-dist values which is 0.071 for a) but 0.017 for c), as data in c) is locally more closely packed to



(a) σ=0.228; KNN-dist=0.071; MST-dist=0.056

(b) σ=0.058; KNN-dist=0.018; MST-dist=0.015

(c) σ=0.218; KNN-dist=0.017; MST-dist=0.037

(d) σ=0.226; KNN-dist=0.017; MST-dist=0.063

(e) σ=0.194; KNN-dist=0.025; MST-dist=0.069

Figure 3.11: Values of σ, KNN-dist and MST-dist for different data distributions over 100 test points.

its neighbors. KNN-dist captures this individual characteristic of points pointing out if the locality is sparse or dense. However, it does not account for disjoint clusters in space, as it is myopic in its scope. This distinction is caught by the MST-dist metric, which combines a local and global scope by identifying variability caused by multiple disjoint spread out clusters. The Figure 3.11 c) and d) have the same KNN-dist values of 0.017, but the MST values show significant differences, being 0.037 and 0.063 for c) and d) respectively. The MST measure is effective in case of attacks, especially the anchor points attacks, which concentrate samples around ground truth values, but the ground truth points are themselves spread out in space. However, using just MST alone is not the best indicator of the data characteristics as depicted in Figure 3.11, where the MST values are similar to d) but the data has a higher local spread. This distinction is caught by the KNN-dist metric, which sees an increase from 0.017 to 0.025.

The experiments of Figure 3.11, indicate the need to consider all three diversity metrics to have a holistic view of the data diversity. As such, our experiments will report all 3 diversity metrics in an attempt to understand data distribution in high dimensional data spaces, where visual analysis of data is not practical. In our future reference, the subscript EA will be skipped while representing the diversity measures. Diversity will be computed only on the correctly classified attack samples, unless otherwise mentioned.

3.4.1.3 Experimental protocol and setup

Experiments in this section follow the following protocol:

a) Experiments begin with a seed sample, which is obtained by random sampling in the feature space upto a maximum of 1000 trials. If no seed is found after 1000 trials, a manual seed is provided by randomly picking a sample from the training data. The AP approach needs only one legitimate sample as seed while RE requires one legitimate and one malicious sample in its seed set.

b) The exploration phase comprises of $B_{Explore}$ probes to the defender model C, to learn from it. The exploration radius of the AP approach is fixed as: $[R_{Neigh-min}, R_{Neigh-max}] = [0.1, 0.5]$. The magnitude for the reverse engineering model λ (Algorithm 3.3) is fixed at 0.25, and it was found that changing it has little effect on the results.

c) The defenders model is taken to be a linear SVM with regularization c=1, unless otherwise mentioned [148]. The reverse engineered model for the adversary is taken to be a linear kernel SVM with a higher regularization constant c = 10. A high c value ensures that fitting to the defenders feedback is given higher weight than generalizing from the samples. This is necessary, as the obtained feedback is limited, and inadequate to try and generalize over the entire space. However, arbitrarily increasing c is not a good strategy, as the defender could mislead the attacker, by providing incorrect feedback for some of the probes. Thus c=10 is taken as a rule of thumb.

d) In the exploitation phase, N_{Attack} samples are generated, which are to be submitted as the adversary attack samples. Unless otherwise mentioned, the $R_{Exploit}$ for AP is taken to be 0.1 and for RE it is taken to be 0.5. A higher value for RE is possible due to the availability of additional information, via the surrogate trained model. Effects of varying these values are also presented.

All experimentation was performed using Python 2.7[1] and the scikit-learn [149] machine learning library. The results presented are averaged over 30 runs for every experiment.

[1] `www.python.org`

3.4.2 Experimental results and analysis

Experimental results are presetned here, by considering different models for the defender's black box, and measuring its impact on the adversary's effectiveness. Section 3.4.2.1 presents illustrative analysis on a 2D synthetic datasets, with a linear defender model. Results of a linear defender model on real world datasets is presented in Section 3.4.2.2. Effects of varying the $B_{Explore}$ and the $R_{Exploit}$, on the accuracy and diversity of the AP and RE attacks, is presented in Section 3.4.2.3. Effects of choosing 4 different non-linear models, for the defedner, is demonstrated in Section 3.4.2.4.

3.4.2.1 Experiments on 2D synthetic dataset

To illustrate the AP and RE approaches and to understand their behavior, experimentation is performed on the synthetic 2D dataset of Figure 3.10. The exploration budget $B_{Explore}$ was taken as 10 probes and the number of attacks to be generated $N_{Attacks}$ was taken to be 20 samples. The experimentation protocol was followed as per Section 3.4.1.3. The results of the seed and exploration phases of the experiment are shown in Table 3.2. The defender's initial accuracy is its perception of safety, which it derives from training its model C and cross validating it on the original training set of malicious and legitimate samples. In this experiment the defender has a perceived safety of 100%. It took only 3 samples on average to find a seed sample, which is classified as legitimate by C. The Explored anchor points value of 0.8 indicates that 80% of the $B_{Explore}$ probes were found to be classified as legitimate by C. These samples form the Anchor Points in the AP approach, and a high value indicates a high AP coverage. The RE accuracy, is computed by testing the initial training dataset (defender's) on the reverse engineered model. This serves as a measure of goodness of reverse engineering, as a high accuracy implies a similarity in the C and the C' hypothesis. A near perfect reverse engineering is seen with 10 exploration samples, as seen by the 99.97% accuracy of reverse engineering, in Table 3.2.

The attack phase follows the exploration phase and is aimed at generating $N_{Attack} =$ 20 samples of high accuracy and diversity. The Effective Attack Rate (EAR)(Equation 3.3)

TABLE 3.2

Results of Seed and Exploration phases on 2D dataset.

Defender's Initial Accuracy	Random probes to find seed	Explored Anchor Points (AP)/$B_{Explore}$	Accuracy of RE model
100%	2.98±1.44	0.8±0.13	99.97%

TABLE 3.3

Results of accuracy and diversity of AP and RE attacks on 2D synthetic data.

	EAR	Diversity of effective attack samples		
		σ	KNN-dist	MST-dist
AP ($R_{Exploit}$=0.1)	0.977±0.03	0.18±0.04	0.116±0.02	0.077±0.01
AP ($R_{Exploit}$=0.5)	0.797±0.12	0.189±0.03	0.169±0.03	0.087±0.01
RE ($R_{Exploit}$=0.1)	0.973±0.05	0.21±0.03	0.137±0.02	0.089±0.02
RE ($R_{Exploit}$=0.5)	0.937±0.08	0.21±0.03	0.173±0.02	0.108±0.01

metric for accuracy and the diversity metrics of -σ, $KNN-dist$, $MST-dist$, are computed for the AP and RE methods of attacks, with two different exploitation radius $R_{Exploit}$= 0.1 and 0.5. The AP approach has a high attack effectiveness at low exploitation radius, with 97.7% of the attacks being classified as legitimate by the defender model C, as shown in Table 3.3. The AP approach has low diversity, as the radius of exploit limits movement away from the originally obtained anchor points. Increasing $R_{Exploit}$ to 0.5 leads to an increase in the diversity metrics, but causes the EAR of the AP approach to drop to 79.7%. This is attributed to the lack of confidence in samples far from the Anchor Points. The RE approach provides high diversity and the diversity can be increased by increasing the exploitation radius, with a much more acceptable drop in EAR value (∼4%). This is a result of the effectiveness of the reverse engineered model which has a RE accuracy of 99.9%, as shown in Table 3.2. The exploited samples can be checked to see if they will be classified as legitimate, by verifying them on the reverse engineered model. This allows the attacker to use a large exploitation radius and still be confident in its attacks. A point to be noted is that exact learning of the decision boundary is not needed, a partial reverse engineering is sufficient while generating a large number of diverse attack samples.

The standard deviation, seen from Table 3.3, is less useful in identifying the diversity of samples, as there is no statistical difference (t-test at significance of 0.05) between the σ of an approach at the two levels of $R_{Exploit}$. Significance changes are seen in the KNN-dist and MST-dist metrics, for the $R_{Exploit}$ of 0.1 and 0.5, indicating their efficacy as a diversity metric. The σ is statistically different between the AP and RE approach, indicating the overall spread of the data, which is higher for RE as it covers more space by distributing the attacks more uniformly, as illustrated by Figure 3.12. In this figure, the AP and RE attacks at $R_{Exploit}$= 0.1 and at 0.5 are presented. The initial learned model (green) shows good separation and a wide margin in separating the malicious (orange) and the legitimate samples (green). Attacks begins with the seed sample shown in blue, and then proceeds to generate probes upto 10 samples (purple). Probes in case of RE are shown to be concentrated near the boundary, as this attack is trying to learn the operating boundary. In case of AP, the attacks are concentrated on the legitimate data space, to obtain more anchor points. Attacks (red) are concentrated near anchor points for $R_{Exploit}$=0.1 as shown in Figure 3.12a). In case of increasing the $R_{Exploit}$ to 0.5, the effectiveness of attacks reduced, seen as samples falling in the malicious side of C (Figure 3.12b)).The reverse engineered model (red) in Figure 3.12 c) and d) shows a close fit with the defender's hidden model, which results in a large spread of attack samples, all of which lie on the legitimate side only. A well reverse engineered model allows the attacker to spread out the attack in the space, making its detection and subsequent fixing harder, for the defender. A sophisticated adversary would prefer the RE approach, as it provides him with the ability to hide his tracks and to spread out attack over time with high confidence, or to blend his attacks with noise. While the quick exploitation of a new found vulnerability is possible with the AP approach.

3.4.2.2 Experiments on real world dataset

Experiments on the 10 real world datasets, were performed by taking a $B_{Explore}$=1000 to generate N_{Attack}=2000 samples. The $R_{Exploit}$ was taken as 0.1 for the AP approach and

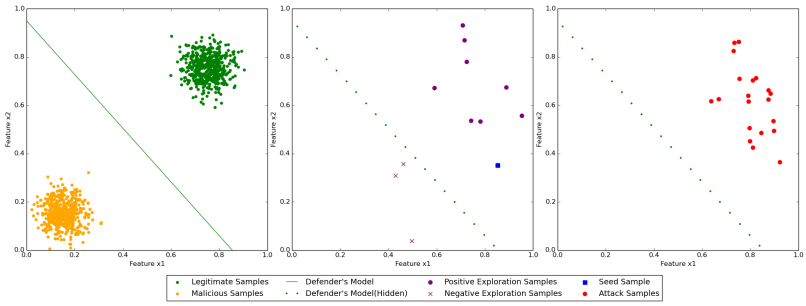
TABLE 3.4

Results of Seed and Exploration phases on real world datasets, with a linear defender's model.

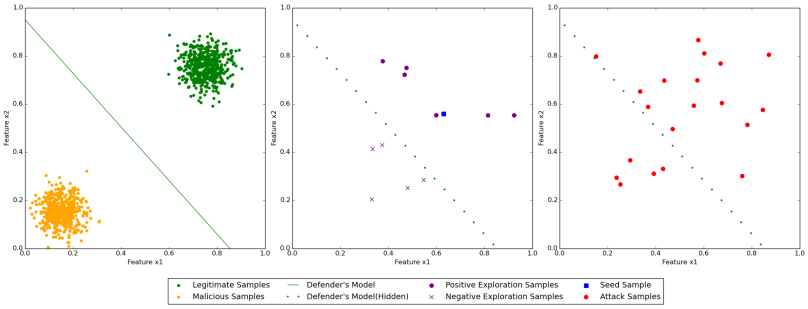
Dataset	Defender's Initial Accuracy	Random probes to find seed	Explored Anchor Points (AP)/$B_{Explore}$	Accuracy of RE model
Digits08	98%	4.6±2.63	0.63±0.01	92%
Credit	79%	3.13±1.89	0.71±0.01	71%
Cancer	97%	42.91±29.36	0.99±0.01	95%
Qsar	87%	49.5±28.81	0.99±0.01	42%
Sonar	88%	24.03±18.92	0.98±0.01	61%
Theorem	72%	4.07±2.52	0.67±0.02	57%
Diabetes	78%	2.93±1.23	0.50±0.02	71%
Spambase	91%	20.64±12.93	0.50±0.02	59%
KDD99	99%	6.07±4.23	0.91±0.01	55%
CAPTCHA	100%	7.27±5.35	0.92±0.01	91%

0.5 for the RE approach, as a high radius for AP leads to drastic decrease in the effectiveness of attacks, as discussed in Section 3.4.2.1.

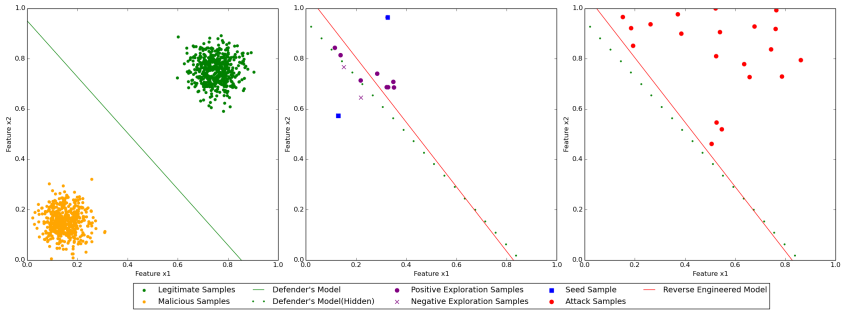
The results of the seed and exploration phases are shown in Table 3.4. No more than 50 samples, on average, were needed for finding a seed and starting the attack process. Over 500 anchor point, on average, were obtained from the exploration phase of AP. For the RE approach, the average accuracy was close to the defender's accuracy for Digits08, Credit, Cancer, Diabetes and the CAPTCHA datasets, while the other datasets showed a lower value.



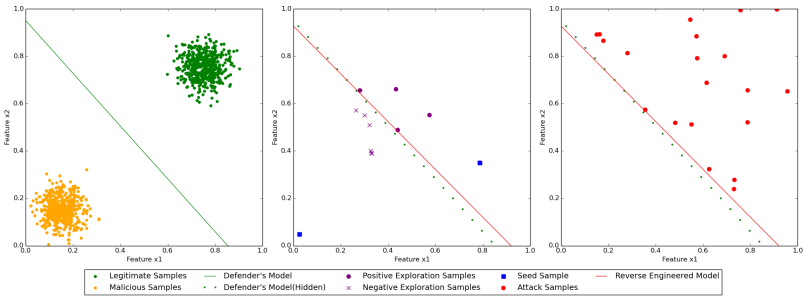
(a) AP attack with $R_{Exploit}$=0.1



(b) AP attack with $R_{Exploit}$=0.5



(c) RE attack with $R_{Exploit}$=0.1



(d) RE attack with $R_{Exploit}$=0.5

Figure 3.12: Illustration of AP and RE attacks on 2D synthetic data. *left*- initial defender's view of data (green: legitimate and orange: malicious), *center*- seed (blue) and exploration phase probes (purple)/ reverse engineered model (red line), *right*- attack samples generated(red).

Attacks were launched by generating 2000 samples after the exploration phase. The results of the AP and RE attacks indicate a high EAR over all datasets (over 70%, with a majority over 90%), as shown in Table 3.5. This indicates that both approaches are effective in attacking classification systems, irrespective of the application domain. Even for datasets where the RE accuracy was found to be low, the EAR is high. Thus confirming our claim that a partial reverse engineering is sufficient for effective attack generation, when attacks are large and diverse. The diversity of the reverse engineering attacks is higher than AP, on all three metrics, indicating a larger spread of attacks, lower collocation of points and a uniform spread the attack space. This high diversity is obtained while still maintaining a high EAR. The AP approach, produces lower diversity but has a high EAR in all cases. This is because the number of effective explored anchor points in Table 3.4 was 50% of the $B_{Explore}$, allowing a large scale AP attack to be possible. The AP approach is therefore an adhoc quick attack mechanism, irrespective of the data domain, application type and model used. Using the proposed hybrid randomization and convex combination of the AP approach's exploit phase, the diversity obtained is higher than obtained by pure random perturbation of anchor points, and at the same time a high evasion rate is possible.

The RE approach's EAR is low for the Credit, Theorem and the Spambase datasets. In case of the Credit and Theorem dataset, the defender's accuracy is low, indicating a nonlinear separation/ inseparability of samples. The RE accuracy approaches close to the defender's accuracy, but since the original model D has limited accuracy, the reverse engineered model can only be so good. For the Spambase dataset, the majority of the features follow a heavy tailed distribution as shown for Feature #5 in Figure 3.13. In such distributions, random sampling in the range [0,1] on each features is not the best choice. Integrating domain information which is commonly known, as in the case of spam datasets having heavy tails, can be beneficial. However, following a domain agnostic approach here, we are still able to achieve a 71% attack rate indicating the viability of such attacks. One thing to note is that although the RE accuracy for the Qsar dataset was low, the attack accuracy was close to 1. This is because, generating a high accuracy on the training dataset

TABLE 3.5

Results of accuracy and diversity of AP and RE attacks on real world datasets, with linear defender's model.

Dataset	Method	EAR	σ	KNN-dist	MST-dist
Digits08	AP	0.96±0.01	0.23±0.002	0.48±0.01	0.41±0.01
	RE	0.93±0.06	0.273±0.009	0.76±0.01	0.65±0.04
Credit	AP	0.98±0.01	0.218±0.001	1.19±0.02	1.01±0.02
	RE	0.80±0.15	0.265±0.001	2.22±0.02	1.72±0.31
Cancer	AP	0.99±0.01	0.215±0.001	0.38±0.01	0.33±0.01
	RE	0.99±0.01	0.263±0.001	0.5±0.01	0.45±0.01
Qsar	AP	1	0.216±0.001	1.1±0.01	0.94±0.01
	RE	0.99+0.01	0.264±0.001	1.71±0.01	1.64±0.01
Sonar	AP	0.99±0.01	0.215±0.001	1.37±0.01	1.16±0.01
	RE	0.98±0.01	0.265±0.001	2.22±0.01	2.1±0.015
Theorem	AP	0.97±0.01	0.219±0.002	1.05±0.02	0.89±0.02
	RE	0.87±0.08	0.267±0.002	1.96±0.02	1.64±0.15
Diabetes	AP	0.98±0.01	0.217±0.003	0.27±0.01	0.23±0.01
	RE	0.95±0.04	0.262±0.001	0.36±0.01	0.31±0.01
Spambase	AP	0.93±0.01	0.233±0.003	0.96±0.02	0.79±0.02
	RE	0.71±0.2	0.273±0.004	2.04±0.06	1.39±0.4
KDD99	AP	0.99±0.01	0.215±0.001	1.06±0.01	0.91±0.01
	RE	0.93±0.04	0.263±0.001	1.71±0.01	1.53±0.06
CAPTCHA	AP	0.99±0.01	0.215±0.001	0.80±0.01	0.68±0.01
	RE	0.97±0.02	0.264±0.001	1.22±0.01	1.12±0.03

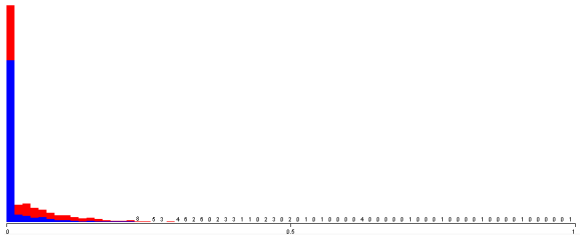


Figure 3.13: Distribution of Feature #5 for Spambase dataset, showing a heavy tail.

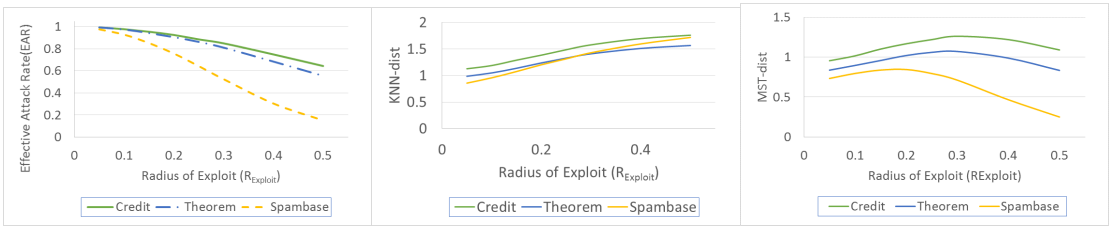
is not the goal of the RE approach. It is more concerned with generating a large number of diverse attack samples which would be classified as legitimate. This is possible even with partial reverse engineering. While a high RE accuracy indicates a high EAR, it is not a required condition for the RE attack, making it of practical use in high dimensional spaces.

3.4.2.3 Effects of varying exploration budget B_{Expore} and radius of exploitation $R_{Exploit}$

In evaluating the AP and RE approaches, the $R_{Exploit}$ was kept fixed at 0.1 for AP and 0.5 for RE. This was intuitively motivated, as confidence in attacks would reduce as distance from anchor points increases, as they are the only ground truth information available to the attackers. Effect of increasing the $R_{Exploit}$, on the accuracy and diversity of AP attacks, is shown in Figure 3.14. The Credit, Theorem and the Spambase datasets were chosen for these evaluation. These datasets are the ones where RE has a low EAR as seen in Table 3.5. Effect of increasing $R_{Exploit}$ to increase diversity of AP attacks, and increasing $B_{Explore}$ to increase EAR of RE attacks, as viable alternatives to improve performance over these three datasets is analyzed and presented.

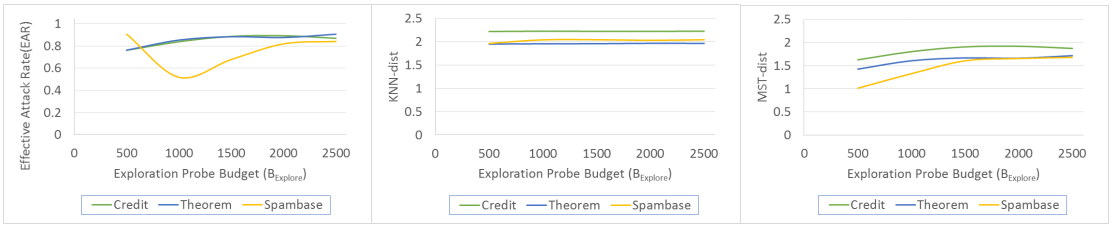
The effective attack rate (EAR) reduces as the radius increases, as seen in Figure 3.14a), as samples move away from the anchor points. There is an associated increase in the diversity using both KNN-dist and MST-dist measures, as shown in b) and d). Comparison of diversity and EAR at radius=0.5 for the RE and AP approach, shows that for increasing diversity it is much better to switch to the RE approach instead of increasing $R_{Exploit}$ arbitrarily, as the effectiveness of attacks starts dropping rapidly with increased radius. The drop in MST-dist in Figure 3.14 c) is due to the reduction of the size of the Effective Attack set (EA).

Since, increasing diversity for AP approach leads to a drop in EAR, we now try to investigate if we can increase the EAR of the RE approach while maintaining its high diversity. Increasing the explore budget increases the EAR as this leads to better training by allowing presenting more labeled samples. The increase in EAR ultimately plateaus, as per the Probably Approximate Learning(PAC) principles [150], indicating that we do not need to arbitrarily keep increasing this budget. The knee point is seen in the Figure 3.15(around 1500 for all datasets). After the knee point, the EAR of the all three datasets is 85%, and adding more probing budget has little impact on the EAR or the diversity. This indicates the need for a larger exploration budget to allow effective reverse engineering in complex



(a) Effect of $R_{Exploit}$ on EAR (b) Effect of $R_{Exploit}$ on KNN-dist(c) Effect of $R_{Exploit}$ on MST-dist

Figure 3.14: Effect of changing $R_{Exploit}$ on the Effective Attack Rate (EAR) and Diversity (KNN-dist and MST-dist), for the AP approach.



(a) Effect of $B_{Explore}$ on EAR (b) Effect of $B_{Explore}$ on KNN-dist(c) Effect of $B_{Explore}$ on MST-dist

Figure 3.15: Effect of changing $B_{Explore}$ on the Effective Attack Rate (EAR) and Diversity (KNN-dist and MST-dist), for the RE approach.

data spaces. This extra effort provides long term benefits as it leads to increased diversity of attacks. RE is suitable for patient and sophisticated adversaries, who want to apply data science in breaking the system. Increasing attack rate without having a high budget is possible by reducing the radius of exploitation. However, this would affect the diversity and as such is not desirable. In case of a low budget, it is better to stick to the AP approach, but with RE, the assumption is that the adversary wants to spend time to learn the system before attempting an attack

3.4.2.4 Experiments with non-linear defender model

While, experiments in this section so far have considered that the defender uses a linear classifier, this is not a limitation/oversimplification of the attack framework. The RE and AP approach are essentially data space search techniques independent of the underlying model type of the defender. The efficacy of these attacks with different defender model is presented here. Particularly, the following defender models were evaluated: K-Nearest

TABLE 3.6

Effective Attack Rate (EAR) of AP and RE, with non linear defender's model (Low EAR values are italicized).

	KNN		***SVM-RBF***		***DT***		***RF***	
Dataset	***AP***	***RE***	***AP***	***RE***	***AP***	***RE***	***AP***	***RE***
Digits08	0.89	0.96	0.97	0.89	0.87	0.63	0.85	*0.48*
Credit	0.96	0.78	0.94	0.53	0.79	*0.42*	0.79	*0.33*
Cancer	0.99	0.99	0.99	0.99	0.97	0.89	0.99	0.98
Qsar	1	0.99	0.99	0.99	0.96	0.76	0.99	0.99
Sonar	0.99	0.98	1	1	0.97	0.62	0.99	0.95
Theorem	0.97	0.813	0.95	0.5	0.95	0.79	0.62	0.78
Diabetes	0.99	0.935	0.99	0.9	0.83	0.63	0.88	0.61
Spambase	0.93	0.99	*0.48*	0.84	*0.08*	*0.11*	0.99	0.98
KDD99	0.99	0.93	1	0.99	0.89	0.54	0.92	*0.27*
Captcha	0.99	0.92	0.99	0.92	0.97	0.83	0.93	0.89

Neighbors classifier with K=3 (KNN) [151], SVM with an radial basis function kernel with gamma of 0.1 (SVM-RBF) [152], C4.5 Decision Tree (DT) [153], and a Random Forest of 50 models (RF) [154], as shown in Table 3.6. The attacker's model is kept the same as before and the experiments are repeated for each of the defender's model. Average values of EAR over 30 runs are reported in Table 3.6.

The AP approach is minimally affected by the choice of defender's model. The drop in case of Spambase, is attributed to the heavy tailed distributions as explained in Figure 3.13. In case of the decision trees, the model trained for Spambase, focuses only on a few key features to perform the classification. Random probing attacks space out attacks across dimensions without their feature importance, this leads to skipping over the key features in the attack generation. This makes the attack is less effective. Although, this could be compensated by performing partial reverse engineering and using a smaller exploitation radius. The reverse engineering results are significantly dependent on the defender's choice of model. In case of nonlinear data separation, as in the Credit and the Theorem datasets, the linear approximation is a bad choice and this is reflected in the low attack rate. In all other cases, the low attack rate is attributed to the over simplification of the understanding of the models, which in case of the decision tree and random forest tend to

be complicated in high dimensional spaces. However, in a majority of the cases it is seen that a 50% attack rate is still possible with the same linear SVM model used by the adversary. This makes the SEE framework generally applicable to attack classification systems, without explicit assumptions about model types, the training data or the parameters of classification. This makes it a purely data driven approach requiring only feature space information.

3.5 Analyzing vulnerability of ML-as-a-service solutions

The ubiquity and widespread applicability of machine learning has driven the development of Machine Learning as a Service (ML-as-a-service) systems, which allow developers to use off-the-shelf solutions to integrate predictive capabilities to their applications. Amazon's Machine Learning service is offered via its web services platform (AWS[1]). Google also offers a prediction API for its own cloud service - the Google Cloud Platform (GCP[2]). These services provide an easy way for developers, with little background into developing ML models, to easily upload their data and get a trained black box for their predictive needs. The training and tuning of models is done by the service providers, providing the ease of usage for developers, via APIs. This is a step towards mainstream access to machine learning methodologies, without the need for domain expertise. While Amazon has mentioned that AWS-ML relies on using a logistic regression model, GCP has not mentioned the details about its predictive model, to the best of our knowledge. This makes GCP a true black box model, and an ideal test ground to demonstrate the efficacy of the AP and the RE techniques, on a real world black box model. Here, the defender's model is remote, accessed from a client, and the adversary has no information about the internal workings of the model [155]. We use the API's Python client library to access the cloud service, and the results on the three cybersecurity datasets are shown in Table 3.7.

The results of the experiment demonstrate that the AP and the RE attacks are effective in attacking the defender's classifier, by generating a high EAR over all datasets. The diversity of the RE approach is seen to be higher for the RE attacks, on all three

[1] `https://aws.amazon.com/machine-learning/`

[2] `cloud.google.com/machine-learning`

TABLE 3.7: Results of AP and RE attacks using the Google Cloud Prediction API, as the defender's black box.

	Spambase		KDD99		CAPTCHA	
Training Accuracy	93%		99%		100%	
Attack Metrics						
	AP	*RE*	*AP*	*RE*	*AP*	*RE*
EAR	1	1	1	1	0.99	0.97
σ	0.216	0.264	0.218	0.265	0.218	0.265
KNN-dist	1.324	2.148	1.105	1.714	0.813	1.228
MST-dist	1.127	2.078	0.944	1.645	0.695	1.131
Accuracy of RE model C'	48.1%		97.2%		100%	

metrics of σ, $KNN-dist$ and $MST-dist$, indicating the variability of attacks achieved using the RE approach, in a real world setting. Furthermore, the RE accuracy in case of the Spambase dataset (48.1%) highlights that linear approximation and partial reverse engineering are sufficient to launch an effective RE attack (EAR=1). These experiments use the same exploration budget ($B_{Explore}$=1000) as the previous sections, to generate attacks of high accuracy and high diversity. In a truly blind-folded setting, where the adversary has no prior information about the defender's classifier, a budget of 1000 ($\approx$ \$0.5)[1] samples indicates the relative ease with which classifiers can be evaded and the need for a more comprehensive defense strategy, beyond a static machine learning model.

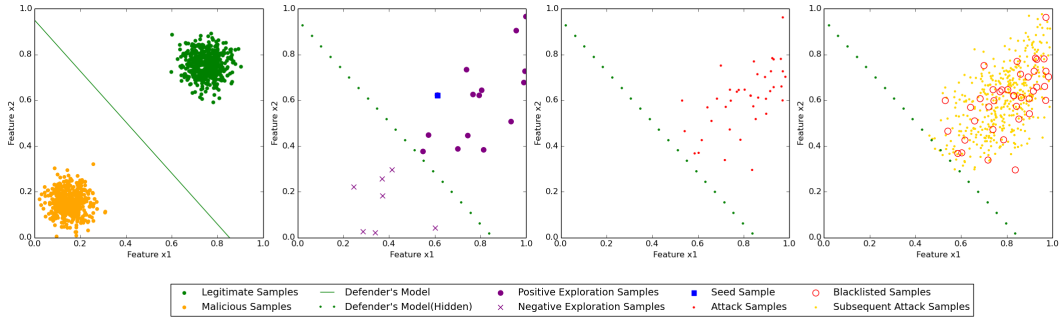
The results of Table 3.7, highlight the vulnerability of black box models, to exploratory attacks. The need to incorporate adversarial thinking and dynamics into the system is necessary. While developers are excited to use a no hassle black box service, it is necessary for them to understand the possibilities at test time, and be prepared for model degradation/attacks. Accuracy and over-fitting are not the only concerns of the model designers, in adversarial environments, and a comprehensive risk analysis is paramount before making strategic integration with critical applications.

[1] `https://cloud.google.com/prediction/pricing`

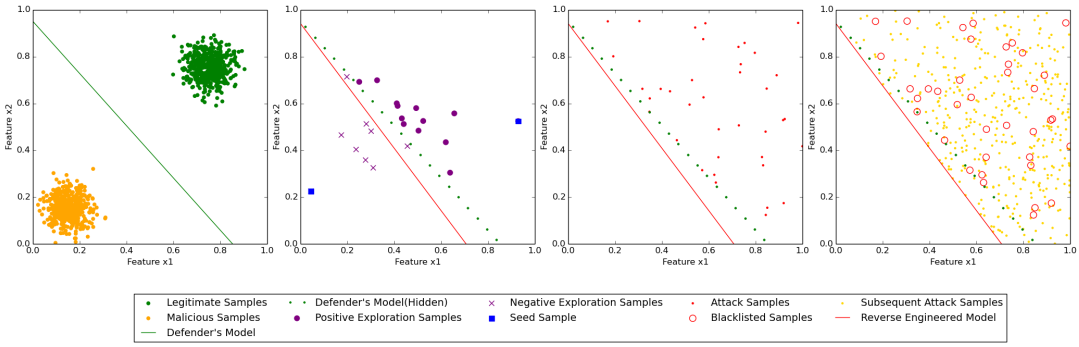
3.6 Why diversity is an important consideration in designing attacks?

Diversity has been considered an important goal for the attacker, in the SEE model. An intuitive explanation for this is: Diversity ensures that attacks have enough variation, causing their detection and prevention to become harder. An ubiquitous approach for blocking attacks is the use of *Blacklists*, to filter out known malicious samples. Modern blacklists are implemented using approximate search methods, such as locality sensitive hashes, which are able to perform approximate matching against attack signatures [141]. With such a blacklist in place, perturbations to attacks are recognized and flagged for further inspection. The goal of an attacker is to avoid detection by these blacklists, as they can make a lot of the attack samples unusable with a quick filtering step. With enough diversity, it is unlikely that blacklisting a few samples will cause the attack to stop. In case of a diverse attack campaign, the defender will have to resort to choosing between maintaining a huge blacklist of samples, or to remodel the machine learning system, both of which are expensive tasks and require time.

To empirically evaluate the effect of diversity on blacklisting, a synthetic blacklisting experiment is presented, which simulates the effect of approximate matching blacklist filters. The blacklist is maintained as a list B, of previously seen attack samples with an associated approximation factor: ϵ. An attack is detected if a new sample falls within ϵ distance to any sample in the blacklist B. The entire blacklisting process is simulated as follows: i) the attackers use the SEE framework to generate $N_{Attacks}$ attack samples which are submitted to the defender model C, ii) the defender is assumed to gain information over time about these $N_{Attacks}$ samples and then proceeds to blacklist them by storing them in B, iii) The attacker, still unaware of the blacklisting, continues to use its existing explored information (AP or RE model) to generate additionally more $N_{Attacks\_new}$ attack samples. The effectiveness of the blacklisting process is computed as the number of effective attacks in $N_{Attacks\_new}$, which are detected by B. The *% of attacks stopped* indicates effectiveness of blacklists and consequently the effect of diversity. A small rate would indicate that blacklisting is not effective in stopping such attacks.



(a) Effect of blacklisting on AP attacks with $R_{Exploit}$=0.1



(b) Effect of blacklisting on RE attacks with $R_{Exploit}$=0.5

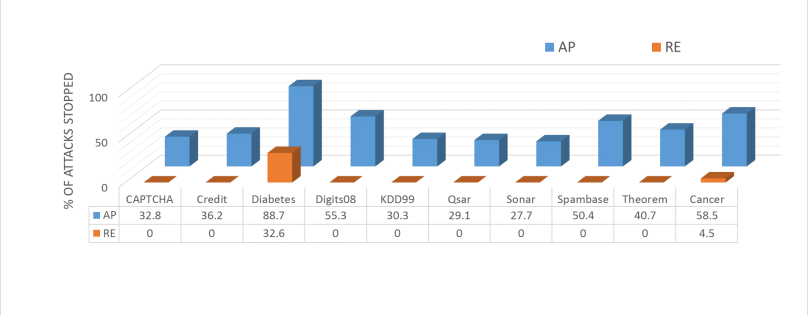
Figure 3.16: Effect of diversity on blacklisting of attacks. Left to Right - The initial training data of the defender and learned model C (green), the exploration phase anchor points (purple) and reverse engineered model C' (red), attack points submitted (red), blacklists (red circles) deployed to stop attacks and attack samples from second round of attacks (yellow).

TABLE 3.8

Effect of blacklisting on AP and RE attacks with ϵ=0.01.

Attack Method	% of Attacks Stopped	% False Alarms
AP	94%	5.3%
RE	44.5%	5.5%

With an exploration budget of 20, exploitation N_{Attack} of 40 and N_{Attack_new} of 400, the experiments are first performed on the 2D synthetic dataset. The results with an approximation factor for the blacklist ϵ=0.01, is shown in Table 3.8. It is observed that blacklisting stops 94% of the AP attacks, which is consistent with its lower diversity observation. This indicates that, on average 376 attacks were stopped by just using 40 blacklisting samples, making the AP attack ineffective. This is illustrated in Figure 3.16 a),



	CAPTCHA	Credit	Diabetes	Digits08	KDD99	Qsar	Sonar	Spambase	Theorem	Cancer
AP	32.8	36.2	88.7	55.3	30.3	29.1	27.7	50.4	40.7	58.5
RE	0	0	32.6	0	0	0	0	0	0	4.5

Figure 3.17: *% of attacks stopped* by blacklist with ϵ=0.1, on real world datasets.

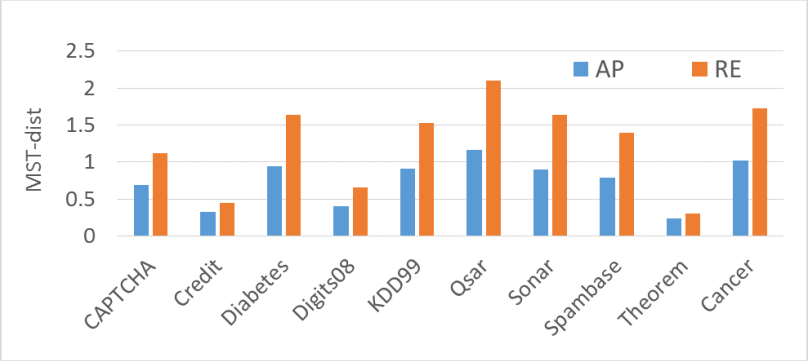


Figure 3.18: MST-dist of AP and RE attacks on real world datasets.

where the AP methodology tends to be concentrated in a smaller portion of the data space, causing subsequent attack samples to be packed more closely. In case of RE, the attacks are more spread out (Figure 3.16 b)) and only 44% of the attacks were stopped, indicating that it can still be used to generate subsequent attack campaigns, without having to repeat the exploration and learning phases. An additional point to be noted is that ϵ cannot be arbitrarily raised to increase the effectiveness of blacklisting. Increasing the approximation factor leads to an increase in the number of false positives, which causes legitimate samples to be misclassified as attacks, increasing the false alarm rate. False alarms are reported in Table 3.8, and are computed by passing the legitimate class training samples through the blacklist and seeing if they are blocked. Monitoring false positives is essential as a blacklist which causes too many false alarms will be impractical.

Results of the blacklisting experiment, with an $\epsilon = 0.1$, on the UCI datasets is shown in Figure 3.17. In order to balance effects of approximation across datasets, the approximation factor is multiplied by $\sqrt{d}$, where d is the number of dimensions of the

dataset. In all experiments, the exploration budget is kept fixed at 1000, the exploitation samples N_{Attack}=2000 and additional 2000 samples(N_{Attack_new}) are generated to test the blacklisting effects. The AP approach has a higher *% of attacks stopped* than the RE approach, across all datasets. This directly indicates the effects of diversity on effectiveness of attacks, as the Figure 3.18 shows. In these cases, increasing the ϵ, as a countermeasure to stop attacks, is not a viable option, due to additional false alarms caused. A higher diversity in RE would force the reevaluation of the security system, leading to redesign, feature engineering and collection of additional labeled samples. All these are time taking and expensive efforts, making the RE approach effective as an attack strategy. As such, if an attacker is sophisticated and has enough probing budget $B_{Explore}$, it can launch diverse attack campaigns, which are harder to stop, by adhoc security measures.

3.7 Chapter summary

This chapter presents the attacker's view of the machine learning based cybersecurity system. Using machine learning has been advantageous to the task of stopping malicious samples, by learning patterns from existing data of normal and abnormal use. However, machine learning is a double edged sword, as it can be used by the attacker as well, to circumvent the security of the system. With minimal knowledge about the learning process, an adversary was shown to be able to attack binary classifiers, with high accuracy. The goal of this chapter was to demonstrate the ease by which classifiers can be evaded, and the need for a new paradigm in using machine learning, when applied to adversarial environments.

The Seed-Explore-Exploit (SEE) framework was used to generate exploratory attacks on classifier systems. The Anchor Points (AP) attacks and the Reverse Engineering (RE) attacks, were presented as two implementations under the SEE framework, demonstrating different attacker goals and resources. Experimental evaluation on 10 real world classification datasets, including 3 from cybersecurity domains, showed the ability of the SEE framework to attack classifiers irrespective of the data domain. The vulnerability of black box and remote cloud based *ML-as-a-service* providers was also demonstrated, to

exploratory attacks via a client API user. Additionally, the use of diversity as a metric, to denote attack strength, was presented and empirically evaluated. Diversity was emphasized as a veritable adversarial criterion, to launch large scale attacks which are difficult to stop by adhoc blacklisting techniques. The following claims made at the start of the section are justified:

- ***Claim a:*** *Attacks can be launched on classifiers, with knowledge of the feature space only, irrespective of the type of classification model of the defender, the training dataset and the domain of application.*

  Experimentation on 10 different datasets was performed, agnostic of their application domains. While it was found that domain information can be helpful to increase the accuracy of attacks, it was possible to attack models, with a greater than 50% evasion rate, with knowledge of only the feature space. Additional experiments were performed with non linear defender's model, unknown to the attacker, and it was found that the AP attacks was effective, irrespective of the model complexity. The RE model was sensitive to the type of model, as it makes the linearity assumption of class separation. Nevertheless, a majority of the cases show a high attack rate, indicating the efficacy of the domain agnostic SEE attacks. Moreover, experimentation on a true remote black box system, the Google Cloud Platform's prediction service, demonstrated the efficacy of launching exploratory attacks in the real world.

- ***Claim b:*** *Classification accuracy, as a metric of predictive performance, has little significance in adversarial domains.*

  The SEE framework when evaluated on 3 cybersecurity datasets, showed that while each of these datasets had a perceived initial accuracy of 90%, it took less than 21 samples, on average, to find a legitimate sample which goes undetected. Based on the SEE framework, an average attack accuracy of 97% was possible for the AP approach and 87% for the RE approach. This indicates that defender's accuracy measure is not the best metric to denote security, in these applications. All these attacks were launched by probing, without manipulating the learning algorithm of the defender. In

case of the Google Cloud Platform, all models had an initial training accuracy >90%, which was perceived as a good fit for the application. However, it was observed that an attack rate of >97% was possible against these models. This makes accuracy provide a false sense of security in adversarial environments.

- ***Claim c:*** *The proposed SEE framework can be used to simulate exploratory attacks on machine learning systems, ranging from simple evasion to reverse engineering of the classification model.*

  The proposed SEE framework was used to simulate AP and RE attacks on 10 real world classification datasets. The AP attacks used ground truth legitimate points to launch attacks, while the reverse engineering model learned the classifications boundary to launch diverse attacks. The SEE framework sets the attack as a search problem, and is general in its applicability, based on the attacker goals, resources and desired effects. A quick attack with low probing budget is possible using an implementation similar to the AP attack. A long term and effective attack is possible if large number of probes can be made to the system, so as to reverse engineer the model. The SEE framework was able to demonstrate both classes of attacks, under a common generic data driven framework. This framework was shown to be effective for vulnerability analysis, under different data domains and choice of defender models, including a real world application where it is evaluated on the Google Cloud Platform's prediction model.

- ***Claim d:*** *Cloud based machine-learning-as-a-service providers, providing off-the-shelf predictive capability, are also not safe, as they can be attacked by non intrusive probing*

  Experiments on Google Cloud Platform's prediction API was performed to analyze the efficacy of the proposed SEE framwork, against a true remote black box model. We used the API as a client user, to probe the model and then launch adversarial samples against the trained model. It was seen that even when models were trained according to effective cross-validation principles, to provide a high accuracy of 97.3%, on average, the models were still evaded with an attack rate of 99.3%. This was

possible even though no information about the trained model was available to the adversary. This demonstrates the fundamental vulnerability of machine learning, which requires a domain agnostic analysis and solution.

The purpose of this chapter is to make the model designers aware of the nature of attacks that invade classification systems, from a purely data driven perspective. These attacks can change data distribution over time, violating the stationarity assumption of machine learning models, making them ineffective for providing security. Wearing the *white hat* in this chapter, we aim to use the lessons learned, in developing secure adversary aware machine learning systems.

CHAPTER 4

REACTING TO DATA DISTRIBUTION CHANGES - DETECTING CONCEPT DRIFT FROM UNLABELED STREAMING DATA USING MARGIN DENSITY

In Chapter 3, it is seen that an adversary can probe the deployed classifier model, to evade it at test time. This evasion is done by crafting samples, whose distribution is different from the training phase, for the *Malicious* class samples. Detecting and handling of these changes, is an important aspect of reactive security measures. However, most works on reactive security is contributed by research in the domain of concept drift. In this domain, dealing with changes to data distribution is emphasized. As such, we start our analysis of the dynamic aspects of attacks by developing a reliable unsupervised drift detection methodology, which can operate in a streaming environment. The methodology in this chapter is developed independently of the application domain. Since attacks cause the distribution to drift at test time, this methodology aims to detect it as a performance affecting change in the test time distribution of data. The generic nature and effectiveness of the methodology, demonstrates it to be an effective reactive strategy for detecting and handling concept drift.

4.1 Detection of concept drift from unlabeled data

Machine learning models in real world applications operate in an environment where the data distribution can change constantly. These changes, called concept drift, can cause the performance of the model to degrade over time [58]. As such, it is necessary to adopt an adaptive strategy, which can detect changes and update the model, when new data is available. While updating a model requires labeled data (for retraining), the detection process

Parts of this chapter are published in [9–11].

does not have to. Labeling is a time consuming and expensive activity, which often requires human intervention. The scale of modern day machine learning applications and the volume of requests, makes labeling a luxury which is not affordable and is impractical [61]. There is therefore a need to reduce the dependence on labeled samples, in the stream classification process. While, periodic retraining of classifiers will need labeled samples from time to time; the continuous use of labeled samples to verify model performance (in-order to detect drifts) is a wasted effort, especially when drifts are infrequent. Majority of the adaptive techniques mentioned in literature [63, 65], rely on the unhindered and infinite availability of labeled data, making their applicability to the real world, suspect. This chapter develops a reliable reactive approach to dealing with concept drift, from unlabeled streaming data. We take a domain independent and adversarial agnostic approach to drift detection, where any changes in the data leading to classification degradation, is considered relevant. This allows for a purely concept drift perspective on the problem of data distribution changes (adversarial or otherwise), making it widely applicable for different application needs.

Being able to detect drifts from unlabeled data, is necessary to ensure scalable usage of adaptive classification systems. However, the existing methods of unlabeled drift detection are essentially change detection techniques, which signal an alarm for any shift in the data distribution, irrespective of its effect on the classification process [7, 64, 156–158]. For the task of classification, change is relevant only when it causes model performance to degrade. This relevance is a function of the learned model, as illustrated in Figure 4.1, where the same data shift resulted in diametrically opposite results. In Figure 4.1a), the model performance is unaffected, while in b), there is a complete failure in the prediction capabilities of C2. The difference lies in the classier models C1 and C2, which are a result of learning on different views of the same data. The existing unlabeled techniques fail to make this distinction between the two cases, as they totally exclude the classifier from the detection process and make decisions solely on the distribution characteristics of the unlabeled data. This results in increased sensitivity to change and a large number of generated false alarms.

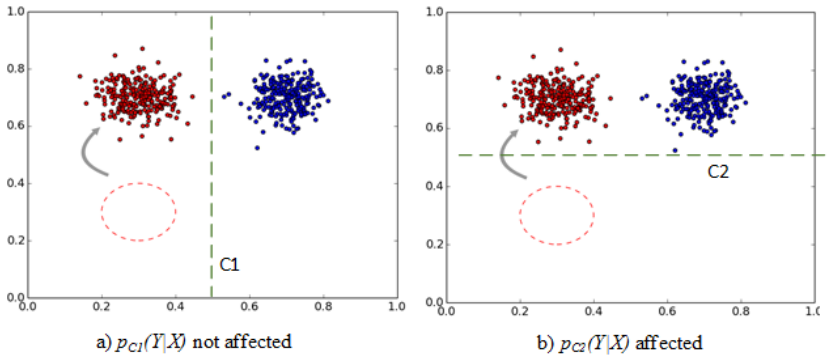


Figure 4.1: Drift as a function of the learned classifier model.

False alarms in drift detection, makes the algorithm overly paranoid and leads to wasted labeling effort, to verify if the change is relevant to the classification task. This behavior is especially undesirable in cybersecurity applications because- a) Frequent false alarms annoys experts who provide model verification, causing the detection process to loose credibility, b) An overly reactive system can be used by an adversary to manipulate learning or to cause it to spend an excessive amount of money on labeling [41] and c) increased labeling due to false alarms are expensive (even using crowd-sourcing websites at large scale, every day is expensive) and they cause delay in detection of attacks.

From a probabilistic perspective, concept drift can be seen as a change in the joint probability distribution of the data samples X and their corresponding class labels Y, as per Equation 4.1 [159]. Unlabeled change detection techniques track changes to $P(X)$, while the labeled drift detection approaches directly track $P(Y|X)$. In this chapter, an unlabeled drift detection methodology is proposed, which can vicariously track changes to $P(Y|X)$, without needing explicit labeled samples. Changes are tracked based on the distribution of samples relative to the learned classifier's boundary, to make it robust towards irrelevant changes in distribution of data.

$$P(X,Y) = P(Y|X).P(X) \tag{4.1}$$

The Margin Density Drift Detection (MD3) methodology, proposed in this chapter, monitors the number of samples in a classifier's region of uncertainty (its margin), to detect

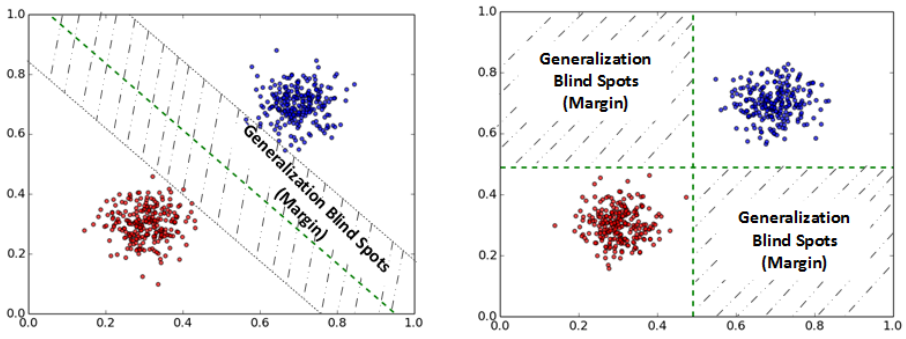


Figure 4.2: Classifier blindspots (margin) for SVM (left) and Feature-Bagged Ensemble (right).

drift. Robust classifiers, such as Support Vector Machines(SVM) [148] or a feature bagged ensemble [118], after training, have regions of uncertainty called margins as depicted in Figure 4.2. These regions are a result of the classifier's attempt to generalize over unseen data and they represent the model's best guess over that data space. A large margin width with a low density (given by number of samples) is at the core of any optimization based classification process (such as SVM). While, explicit information about class distribution is learned in the training of a classifier, an additional auxiliary information also learned and often overlooked is the margin characteristics, such as the expected margin density. This information is representative of the data state and any change in it could indicate non-stationarity. Margin is crucial to the generalization process and any changes to the margin density is worthy of further verification. Since the margin density can be computed from unlabeled data only, it could be used as a substitute to explicit-labeled drift detection techniques, for monitoring changes in $P(Y|X)$.

In this chapter, we present the proposed MD3 approach and test the following claims: a) The MD3 approach can be used as a substitute for fully labeled approaches to detect drifts, without significant reduction in the predictive accuracy over the stream, b) The MD3 approach can reduce the number of false alarms compared to other traditional change tracking unlabeled methodologies, c) The MD3 approach can perform computations incrementally in a streaming environment and can be used independently of the type of classifier

used and the application domain, and d) The robustness of MD3 leads to less labeling effort needed in the adaptation process. The MD3 algorithm is proposed as a streaming drift detection algorithm and as such it needs to operate under the constraints of streaming data, which are: Data samples need to be processed one at a time continuously, only limited amount of memory is available to store past information and response needs to be swift and in near real time. The chapter is organized as follows: Section 4.2 presents background work relevant to labeled and unlabeled drift detection approaches. Section 4.3 presents the Margin Density metric and the Margin Density Drift Detection (MD3) streaming algorithm, which uses the margin density metric. Experimental results on drift induced datasets, real world concept drift datasets from cybersecurity domains, and benchmark concept drift datasets, are presented in Section 4.4. Additional discussion regarding the efficacy of the margin based approach, when compared to other state of the art uncertainty tracking methodologies, is presented in Section 4.5. Conclusion and chapter summary is presented in Section 4.6.

4.2 Review of concept drift detection techniques

Detecting change is essential to trigger based stream adaptation strategies. Several methods in literature have been proposed recently (Table 4.1, [63]). The techniques can be divided into two categories, based on their reliance on labeled data: Explicit/Supervised drift detectors and Implicit/Unsupervised drift detectors. Explicit drift detectors rely on labeled data to compute performance metrics, such as accuracy and f-measure, which they can monitor online over time. They signal drop in performance and as such are efficient in signaling change when it matters. Implicit drift detectors rely on properties of the unlabeled data features, to signal deviations. They are prone to false alarms, but their ability to function without labeling, makes them useful in applications where labeling is expensive, time consuming or not available. Table 4.1 shows a taxonomy of the drift detection techniques with popular techniques in each category.

TABLE 4.1

Summary of drift detection approaches in literature.

Explicit drift detection (Supervised)	Sequential analysis	CUSUM [160], PHT [160], LFR [161]
	Statistical Process Control	DDM [162], EDDM [163], STEPD [164], EWMA [165]
	Window based distribution monitoring	ADWIN [166], DoD [167], Resampling [168]
Implicit drift detection (Unsupervised)	Novelty detection/ clustering methods	OLINDDA [156], MINAS [169], Woo [158], DETECTNOD [7], ECSMiner [157], GC3 [102]
	Multivariate distribution monitoring	CoC [170], HDDDM [64], PCA-detect [171, 172]
	Model dependent monitoring	A-distance [173], CDBD [174], Margin [175]

4.2.1 Explicit concept drift detection methodologies

4.2.1.1 Sequential analysis methodologies

These techniques continuously monitor the sequence of performance metrics, such as accuracy, f-measure, precision and recall; to signal a change, in the event of a significant drop in the values. The CUmulative SUM (CUSUM) approach of [160], signals an alarm when the mean of the sequence significantly deviates from 0. As per Equation 4.2, the CUSUM test monitors a metric M, at time t, on an incoming sample's performance ϵ_t, using parameters v for acceptable deviation and θ for change threshold.

$$\begin{gathered} M_0 = 0; \quad M_t = max(0, M_{t-1} + \epsilon_t - v) \\ if \quad M_t > \theta \quad then \quad 'alarm' \quad and \quad M_t = 0 \end{gathered} \tag{4.2}$$

where, M_0 is the initial metric at time t=0. M_t is the current metric computed as an accumulation of the metric so far - M_{t-1}, and the sample's performance at time t= ϵ_t. The parameter v denotes acceptable deviation from mean and θ is the change detection threshold. The max function in the above equation is used to test changes in the positive direction, for a reverse effect (i.e., to measure drop in accuracy), a min function can be used. This test is memory-less and can be used incrementally. A variant of this approach is the Page-Hinckley Test (PHT) [160], which was originally developed in the signal processing domain to detect deviation from mean of the Gaussian signal. PHT monitor the metric as an accumulated difference between its mean and current values, as per Equation 4.3.

$$\begin{aligned} &M_0 = 0; \quad M_t = M_{t-1} + (\epsilon_t - v); \quad M_{Ref} = \min(V) \\ &if \quad M_t - M_{Ref} > \theta \quad then \quad 'alarm' \quad and \quad M_t = 0 \end{aligned} \tag{4.3}$$

where, the terms have the same meaning as in case of Equation 4.2, as described above. Both the CUSUM and the PHT, are best suited for univariate change detection of a sequence of performance measures, tracked for online algorithms. A related statistical change detection was proposed in [161], to deal with imbalanced streaming data, which monitors multiple performance metrics. The technique monitors the true positive rate, false positive rate, true negative rate and false positive rate, obtained from the confusion matrix of the classification. The confusion matrix presents a detailed view of the classification performance by explicitly monitoring the number of true positives, false positive, false negative and true negatives, as shown in Figure 4.3. While traditional metrics of accuracy are biased towards the majority class, the confusion matrix presents a more detailed view, suitable for imbalance class problems.

4.2.1.2 Statistical Process Control based methodologies

The Probably Approximately Correct (PAC) learning model of machine learning, states that the error rate of a trained model will decrease with increasing number of samples, if the data distribution remains stationary [150]. The drift detection techniques based on Statistical Process Control, monitor the online trace of error rates, and detects deviations

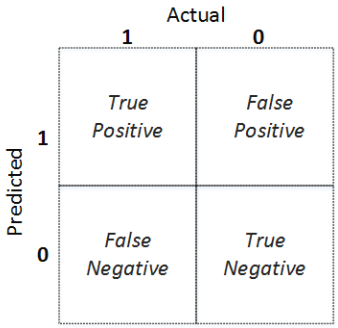


Figure 4.3: Confusion matrix showing the four classification performance metrics.

based on ideas taken from control charts. A significantly increased error rate violates the PAC model and as such is assumed to be a result of concept drift. The Drift Detection Method (DDM) [162] and the Early Drift Detection Methodology (EDDM) [163] are popular techniques in this category.

The DDM approach monitors the probability of error at time t as p_t and the standard deviation as $s_t = \sqrt{p_t(1-p_t)/i}$. When, $p_t + s_t$ reaches its minimum value, the corresponding values are stored in p_{min} and s_{min}. A warning is signaled when $p_t + s_t \geq p_{min} + 2 * s_{min}$ and a drift is signaled when $p_t + s_t \geq p_{min} + 3 * s_{min}$. The EDDM was developed as an extension of DDM, and was suitable for slow moving gradual drifts, where DDM failed. EDDM monitors the number of samples between two classification errors, as a metric to be tracked online for drift detection. Based on PAC model, it was assumed that, in stationary environments, the distance (in number of samples) between two subsequent errors would increase.

The Statistical Test of Equal Proportions (STEPD) [164], computes the accuracy of a chunk C of recent samples and compares it with the overall accuracy from the beginning of the stream, using a chi-squared test to check for deviation. An incremental approach was proposed in [165], where the Exponentially Weighted Moving Average(EWMA) was used to signal deviation in the average error rate, in terms of the number of standard deviations from the mean. The metric M (here, error rate) at time t is updated as per Equation 4.4.

$$M_0 = \mu_0; \quad M_t = \lambda * M_{t-1} + (1 - \lambda) * \epsilon_t$$
$$if \quad M_t - \mu_0 > \theta * \sigma_0 \quad then \quad 'alarm' \tag{4.4}$$

where, μ_0 and σ_0 are mean and standard deviation obtained from the train data, by random sampling. The error rate at time t is given by ϵ_t, θ is the acceptable deviation in terms of number of standard deviation from the mean, and λ is the forgetting factor which controls the effect of previous data on the current sample. The EDDM, STEPD and EWMA, also employ the warning and subsequent drift signaling system as in the DDM approach.

4.2.1.3 Window based distribution monitoring methodologies

Unlike all the methods mentioned thus far, which operate in an incremental fashion one sample at a time, window based approaches use a chunk based or sliding window approach over the recent samples, to detect changes. Deviations are computed by comparing the current chunk's distribution to a reference distribution obtained at the start of the stream, from the training dataset [166]. Window based approaches provide precise localization of change point, and are robust to noise and transient changes. However, they do need extra memory to store the two distributions over time.

The Adaptive Windowing (ADWIN) algorithm of [166], uses a variable length sliding window, whose length is computed online, according to the observed changes. In case change is present, the window is shrunk and vice-versa. Whenever, two large enough sub windows, of the current sliding window, exhibit distinct averages of the performance metric, a drift is detected. Hoeffding bounds [176] are used to determine optimal change threshold and window parameters. The ADWIN methodology was shown to provide rigorous performance guarantees, efficient memory and time complexities, and freedom from having to specify cumbersome parameter values. Another window based approach- the Degree of Drift (DoD), detects drifts by computing a distance map of all samples in the current chunk and their nearest neighbors from the previous chunk [167]. The Degree of Drift metric is computed based on a distance map and if the distance increases more than θ, a drift is signaled. The

Paired Learners approach of [177], uses a pair of *reactive learner*, trained on recent chunk of data, and a *stable learner*, trained on all previously seen data. Differences in accuracies between the two approaches is indicative of a drift. This disagreement is captured in a binary valued circular list, and an increase in the number of ones beyond a change threshold θ, is signaled as concept drift. Drift is managed by replacing the stable model with the reactive one and setting the circular disagreement list to all zeros.

A recently proposed permutation based method [168], relies on the assumption than randomly choosing training and testing data from a chunk of data should lead to similar accuracy of prediction, unless the window has nonstationary data. This method is based on the idea commonly used in classifier's cross validation evaluation [178]. In cross validation, the entire training dataset is split into K bands (sequential sets of samples). In each iteration or fold of the cross validation approach, one band is chosen as the test data and the other (K-1) bands form the training dataset. Generating a model on the training dataset and then testing on the test dataset, gives the performance on that fold of the cross validation process. The same process is repeated K times and the average performance is reported. Cross validation provides a good estimate of the generalization error, when the data is stationary. The permutation approach of [168], splits the current window into two parts to train a model on the first half and test on the second half. The window is then shuffled and the process is repeated. A significant change in the performance between the two indicates a drift, as concept drift is sensitive to the sequence of data. This method was shown to have better precision-recall values and robustness, when compared with the DDM, EDDM and the STEPD described in Section 4.2.1.2.

4.2.2 Implicit drift detection methodologies

4.2.2.1 Novelty detection / Clustering based methods

Novelty detection methods relies on using distance and/or density information to detect previously unseen data distribution patterns. These methods are capable of identifying uncertain suspicious samples which need further evaluation; and they define an additional

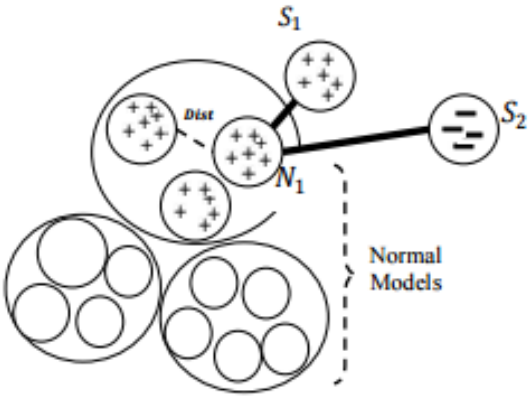


Figure 4.4: Distinction between Novelty (S2) and Drift (S1) in DETECTNOD [7].

'unknown' class label to indicate that these samples do not fit the existing view of the data [156]. Clustering and outlier based approaches are popular implementation strategies for detecting novel patterns, as they summarize current data and can use dissimilarity metrics to identify new samples [158].

The OnLIne Novelty and Drift Detection Algorithm (OLINDDA), uses K-means data clustering to continuously monitor and adapt to emerging data distributions [156]. Unknown samples are stored in a short term memory queue, and are periodically clustered and then either merged with existing similar cluster profiles or added as a novel profile to the pool of clusters. The MINAS algorithm of [169], uses micro clusters which it obtains using an incremental stream clustering algorithm- CluStream, and it extends the OLINDDAs approach to be used in a multi class problem. The DETECTNOD algorithm of [7], uses a clustering model to define the boundaries of existing known data. It relies on Discrete Cosine Transform (DCT) to build a compact representation of these clusters and uses this information to provide a nearest neighbor approximation on incoming test samples. Samples falling out of the normal mode, are clustered into k clusters and based on their similarity values to existing clusters, they are either termed as 'Novelties' or 'Drifts'. Figure 4.4 illustrates this process, where S1 is identified as a drift in the existing normal mode sub clusters and S2 is identified as a novel pattern.

The Woo ensemble [158] and the ECSMiner [157], are two techniques which rely on

the concept of micro clusters. The Woo ensemble clusters data and assigns a classifier to each of these clusters. A new sample falling outside the boundary of any existing cluster is marked suspicious and its density is monitored. An increased number of samples within the radius of suspicious samples indicates a new concept, which then triggers retraining of models and readjustment of the cluster centroid. The ECSMiner uses the concept of Filtered Outliers, which represent samples which fall outside the boundary of all existing clusters, and works similar to the Woo ensemble. The GC3 approach of [102], extends this idea of micro clusters to be used with a grid density based clustering algorithm, where novelty is determined by newly appearing dense grids in the data space.

All the above novelty detection techniques, rely on clustering to recognize new regions of space, which are previously unseen. As such they suffer from the curse of dimensionality, being distance dependent, and also the problem of dealing with binary data spaces. Additionally they are suitable to detect only specific type of *cluster-able* drifts. If the drift does not manifest itself as a new cluster or a novel region of space, these detection techniques will fail. Nevertheless, these techniques are suitable for multi-class classification problems where many classes can appear and disappear during the course of the stream.

4.2.2.2 Multivariate distribution monitoring

Multivariate distribution monitoring approaches, directly monitor the per feature distribution of the unlabeled data. These approaches are primarily chunk based, which store summarized information of the training data chunk (as histograms of binned values), as the reference distribution, to monitor changes in the current data chunk. Hellinger distance and KL-divergence are commonly used to measure differences between the two chunk distributions [179], and to signal drift in the event of a significant change.

The Change of Concept (CoC) technique [170], considers each feature as an independent stream of data, and monitors correlation between the current chunk and the reference training chunk. Change in the average correlation over the features is used as a signal of change. Pearson correlation was used, which makes the normality assumption for the

distribution. A non parametric and widely applicable unlabeled approach was proposed in [64], called the Hellinger Distance Drift Detection Methodology (HDDDM). It is a chunk based approach which uses Hellinger distance to measure change in distribution, over time. An increased Hellinger distance, between the current stream chunk and a training reference chunk, is used to signal drift. Chunk distribution is computed by making a histogram for each feature, with $\sqrt{N}$ bins where N is the number of samples in the chunk. The Hellinger distance (HD) between the reference chunk P and the current chunk Q is computed using Equation 4.5. Here d is the data dimensionality and b is the number of bins (b=$\sqrt{N}$), per feature.

$$HD(P,Q) = \frac{1}{d}\sum_{k=1}^{d}\sqrt{\sum_{i=1}^{b}\left(\sqrt{\frac{P_{i,k}}{\sum_{j=1}^{b}P_{j,k}}} - \sqrt{\frac{Q_{i,k}}{\sum_{j=1}^{b}Q_{j,k}}}\right)^2} \tag{4.5}$$

The computed Hellinger distance, which is averaged over all the features, results in a number in the range [0,$\sqrt{2}$]. A HD value of 0 indicated completely overlapping distributions while$\sqrt{2}$ indicates total divergence. The HDDDM approach in [64], was used to detect drifts in conjunction with an incremental learning algorithm, to trigger resetting of the model. Efficacy of this approach was indicated by increased accuracy, which was a result of the interventions leading to retraining, upon drifts detection.

To make the drift detection computationally efficient in high dimensional data streams, Principal Component Analysis (PCA) based feature reduction was used in [171] and [172], which reduce the set of features to be monitored. It was shown that monitoring the reduced feature space allowed to detect drifts in the original features. [171] advocates the use of the Semi Parametric Log Likelihood (SPLL) criterion to monitor changes in the data projected on the principal components. It was proposed that monitoring the principal components with the lowest 10% of Eigenvalues is sufficient for detecting effective drifts. However, the work in [172], presented contrasting results. It was shown that analyzing principal components with large Eigenvalues is more valuable, as data characteristics in the original feature space are best summarized by the top component vectors, which retain the maximum variance after the reduction.

Although, the PCA based approached are efficient in reducing the number of features to be tracked, they still suffer from significant false alarms, as do the other multivariate distribution approaches. All of these methods are sensitive to changes in any of the features, irrespective of their importance to the classification task. These methods are also not suitable for detecting concept drifts in cases where drift is not manifested by feature distribution changes ($P(Y|X)$ changes but not $P(X)$). Furthermore, in the classification of imbalanced datasets, these methods are not effective in tracking the changes to the minority class samples. Changes in these samples do not signal a significant deviation, as minority class samples comprise only a small percentage of the original dataset.

4.2.2.3 Model dependent drift detection methodologies

The methodologies of Section 4.2.2.1 and 4.2.2.2, explicitly track deviations in the feature distribution of the unlabeled data. As such, they are essentially change detection methodologies, which assume that a change in data distribution $P(X)$ will lead to changes in the classification performance $P(Y|X)$. While these methods are attractive for their independence to the type of classifier used, these methods lead to detecting a large number of false alarms (i.e., changes which do not lead to degradation of classification performance). False alarms lead to wasted human intervention effort and as such are undesirable. The model dependent approaches of [107, 173–175], directly consider the classification process by tracking the posterior probability estimates of classifiers, to detect drift. They can be used with probabilistic classifiers, which output the class probabilities $P(Y|X)$ before thresholding them to generate the final class label. By monitoring the posterior probability estimates, the drift detection task is reduced to that of monitoring a univariate stream of values, making the process computationally efficient.

The use of the Kolmogorov-Smirnov test, Wilcoxon rank sum test and the two sample t-test, was suggested in [175], to monitor the stream of posterior probability estimates. The idea of margin was introduced, by using a 1-norm SVM, and the average uncertainty of samples was monitored in lieu of the multivariate feature values. This idea was extended

by [173], to the task of detecting domain shifts in high dimensional text classification applications. A reduced false positive rate was obtained by tracking the 'A-distance', which was proposed as a measure of histogram difference obtained by binning the margin distribution of samples, between the reference and current margin samples. The Confidence Distribution Batch Detection (CDBD) approach [174], used KL-divergence to perform a similar analysis of classifier confidence output values (margin), to detect drifts over text streams. Additionally, they combine the drift detection with active learning, to further reduce the amount of labeled data requested.

These methods are attractive as they significantly reduce false alarms. However, their dependence on using probabilistic models limit their applicability. Also, these methods trigger to any change in the posterior distribution of the margin samples. Changes away from the margin of the classifier are less critical to the classification process, but none of the above mentioned approaches provide robustness against such changes.

4.2.3 Unlabeled drift detection in adversarial classification

In the domain of adversarial classification, where concept drift is initiated by an attacker intending to subvert the system, unlabeled drift detection can be extremely helpful as an automated early warning system. Ensemble based techniques have been proposed in [117] and [103], which use disagreement scores between the ensemble models, to signal changes to the data distribution. In [117], an ensemble of classifiers is used to perform email spam classification. The average pairwise mutual agreement of the classifiers in the ensemble was used to signal change and drive retraining. However, the methodology also relies on periodic checking, using labeled samples, to ensure that high agreement is not a result of undetected changes affecting most classifiers. Recent work in [53], also indicates the relationship between drift and classifier agreement scores. Feature bagging was found to be effective in characterizing adversarial activity in the task of malicious pdf classification. Drifts caused the classifier agreements to shift disproportionately towards the center of the [0,1] range, instead of being concentrated at the peripheries. The work in [53], concentrates

on an empirical analysis of this effect and provides initial experimentation specific to the pdf malware domain. A visual inspection of the change is presented. However, harnessing it as a signal for change in the context of streaming data was not explored.

The proposed Margin Density Drift Detection (MD3) technique, in this chapter, provides a way to signal drift from unlabeled data, in a reliable manner. Unlike other implicit drift detection techniques, the proposed MD3 approach is less susceptible to raising false alarms. By actively including the learned classifier's information into the decision process, the MD3 approach is able to discern changes that could adversely affect the classification performance. As such, it embodies the benefits of both the classes of drift detectors - like explicit drift detectors, it detects drifts only when they can impact classification results, and it does so using unlabeled data, saving labeling budget as with the implicit drift detectors. In doing so, the approach bridges the gap between the two categories of drift detectors, by providing the first of its kind - domain independent, cost-effective and model independent, drift detection scheme for reliably signaling change in high dimensional data streams.

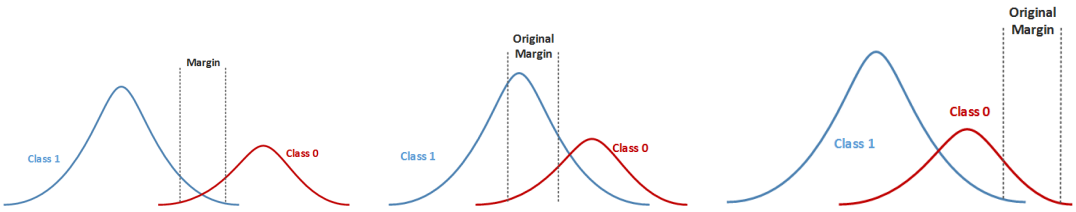
4.3 The Margin Density Drift Detection (MD3) methodology

The proposed margin density approach for detecting drifts, uses the average number of samples in a classifier's margin as a univariate signal to be tracked over time. The MD3 approach provides a distribution independent, classifier type independent, unlabeled drift detection approach, capable of detecting drifts from high dimensional streaming data; with high robustness towards stray changes and false alarms. The margin density metric along with the motivation behind its usage is presented in Section 4.3.1. Computing margin density from probabilistic and non probabilistic classifiers is presented in Section 4.3.2, along with synthetic experiments on a 2D dataset to demonstrate its change detection characteristics. Use of margin density signal in a streaming algorithm to detect concept drift and trigger retraining, is described in Section 4.3.3.

4.3.1 Motivation

The ability to generalize from the training dataset is at the core of any classification technique. This generalization effort leads to regions of space, known as margins (Figure 4.2), where the classifier is uncertain and tries to present a best guess, based on its learned information. Margin is the portion of the prediction space, which is most vulnerable to misclassification. This intuition has been used by existing works on active learning [104, 105], to develop labeling strategies based on uncertainty of data samples. The Uncertainty sampling technique [180] and the Query By Committee technique [104], are two methodologies which select samples based on the distance from the classification boundary and the disagreement between ensemble models for a given samples, respectively. These approaches explore the informativeness of the margin samples for the task of managing labeling budget efficiently. They advocate a reduced dependence on labeled samples, but still rely on continuous monitoring of stream with labeled data. The MD3 approach explores the use of margin tracking for unlabeled drift detection.

A change in the number of samples in the margin is indicative of a drift, as depicted in Figure 4.5, where the distribution of samples with respect to their distances from the classifier boundary is shown, against a fixed margin. A sudden increase or decrease in the number of samples within the margin, makes the stationary assumption of data, suspect. Classifiers define margin width and acceptable misclassifications during the training process, to avoid over fitting. As such tracking a sudden change in the margin characteristics can indicate distribution changes. Changes of data distribution relative to the classification boundary, enables tracking of the posterior probability distribution of the space $P(Y|X)$ without using labeled data, by tacitly involving the classifier in the detection process. Thus, the change relevance, which is a function of the learned model (Figure 4.1), is directly included in the detection process. By using a fixed margin and by tracking the density of the samples closest to the classification boundary, the proposed MD3 approach avoids false alarms caused by changes away from the boundary, which seldom result in any performance degradation. This makes the change detection process robust, and provides a better approach to utilizing



(a) Initial distribution of samples wrt. distance from classifier boundary.

(b) Drift causing increase in margin density.

(c) Sudden drift causing drop in margin density.

Figure 4.5: Drifting scenarios and their effects on the margin density.

the margin, than the existing margin based methods described in Section 4.2.2.3.

The idea of margin is intuitive in probabilistic classifiers, which have an explicit notion of uncertainty. However, the motivation behind its usage is much more general. A classifier's boundary is an embodiment of the set of features which it deems important to the prediction task at hand. Monitoring changes close to the boundary enables us to limit tracking to the important features only. The multivariate approaches of Section 4.2.2.2 suffer from false alarms, because they do not differentiate between changes in any of the features, giving equal weights to all features. The margin density approach tracks margin changes, which summarizes the important features and their interaction, which results in boundary formation.

In a real world high dimensional datasets, there are multiple sets of features which can provide high classification performance. A robust classifier, such as a SVM with hinge loss or a feature bagged ensemble, can utilize a majority of these features, by evenly distributing weights among them, to create a better generalization over the data [71, 123]. In doing so, the classifier model serves as a committee of experts with multiple independent perspectives on the same data. A change in any one perspective (set of features), will cause an increased disagreement and consequent uncertainty in the predicted results. Since, only relevant features can provide a good perspective on data, this uncertainty is indicative of a drift which requires attention. The robust classifier functions as a self-contained, self-monitoring prediction unit, aware of its own capabilities and deviations. In Figure 4.6, the coupled classifier $C1 \vee C2$, leads to a monitoring scheme where C1 and C2 are constantly

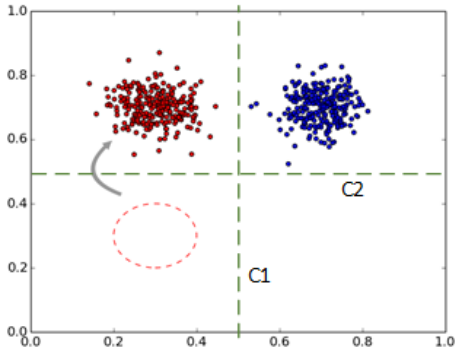


Figure 4.6: Change detected by coupled classifiers $C1 \vee C2$.

monitoring each other. This coupled detection strategy can be extended to high dimensional spaces, and it provides an unsupervised approach where changes in some relevant features are triggered by corresponding invariances in other relevant features. Using this idea, the margin density approach can be extended to classifiers such as decision trees and K- nearest neighbors, which provide explicit class labels and not probabilistic values, by using them in a feature bagging ensemble. This ensemble setup trains multiple base models, on different subset of features, and combines their results. This has the effect of distributing classification importance weights to the different features, as the features are averaged to produce the final prediction.

4.3.2 The Margin Density (MD) metric

The Margin Density (MD) is introduced as a univariate metric, which can be tracked over time, to detect drifts from unlabeled data. The MD metric is defined as:

Definition 4.3.1. ***Margin Density (MD):*** *The expected number of data samples that fall within a robust classifier's (one that distributes importance weights among its features) region of uncertainty (its margin).*

The MD metric, being a ratio, has its value in the range of [0,1]. MD can be computed from classifiers with explicit notions of margins, such as a linear kernel SVM. It can also be computed by disagreement scores of feature bagged ensembles. Here, the term

margin is taken as a notation for the regions of uncertainty of a robust classifier, where the classification importance weights are distributed among its features. For SVM, the margin is well defined by the algorithm, but for other classifiers, such as decision trees, we use the notion of a pseudo-margin given by the region of space with high disagreement based on a feature bagged ensemble. The term margin will be used to refer to the region of uncertainty for both the cases, as a notation. Section 4.3.2.1 shows how the MD can be computed for these two cases and Section 4.3.2.2 shows its ability to detect different kinds of changes in the data distribution, by experimentation on a synthetic dataset.

4.3.2.1 Computing the margin density metric

Classifiers with explicit margins

Classifiers such as Support Vector Machines (SVM) and Logistic regression, explicitly define margins in their setup [148]. A soft margin linear kernel SVM finds an optimal maximum width separating hyperplane between two classes, by allowing a few samples to enter the margin, for better generalization capability. This is made possible by the addition of slack variables (ξ) to the SVM's objective function, to allow for non separable cases and to add robustness against noisy samples close to the SVM boundary. The optimization function of a linear kernel soft margin SVM is given in Equation 4.6 [148], where w is the normal vector of the separating hyperplane given by $w.x + b = 0$ and b gives the offset from the origin, y_i is the class label of the sample x_i and C is the regularization cost parameter which controls the misclassification cost.

$$\begin{aligned} min \quad & \frac{1}{2} w^T w \quad + \quad C \sum_{i=1}^{m} \xi_i \\ s.t. \quad & y_i.({x_i}^T w + b) \geq 1 - \xi_i; \quad \xi_i \geq 0 \end{aligned} \tag{4.6}$$

The above equation learns a linear boundary, which separates the two classes with a margin of width $2/||w||$ given by $w.x \pm b = 1$, as shown in Figure 4.7. The trained SVM model, has an expected number of samples in the margin, due to its soft constraints. The margin density here, is given by the ratio of samples which fall inside the margin of the

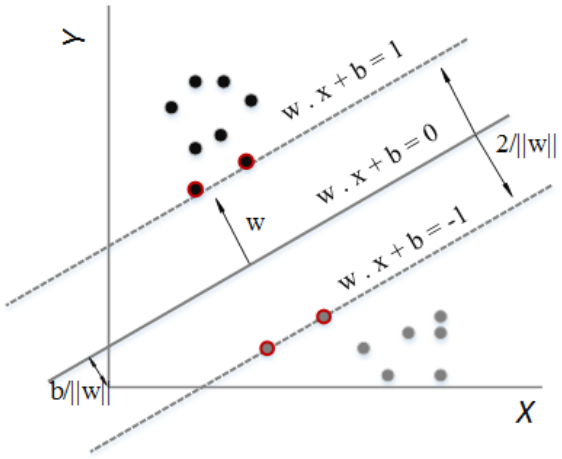


Figure 4.7: SVM with margin $w.x \pm b = 1$.

SVM, as per Equation 4.7. The signal function $S_{(w,b)}(x)$ checks if a given sample x falls within the margin of the SVM, with parameters w and b.

$$\begin{aligned} MD_{SVM} &= \frac{\sum S_{(w,b)}(x)}{|X|}; \quad \forall x \in X \\ where, \quad S_{(w,b)}(x) &= \begin{cases} 1, & if \quad |w.x + b| \quad \leq \quad 1 \\ 0, & otherwise \end{cases} \end{aligned} \tag{4.7}$$

The set of unlabeled samples is given by X and the distance from the hyperplane is given by $|w.x + b|$. This term is threshold by a $sign()$ function, to produce the final class label of +1 or -1. If the distance from the hyperplane, for a sample x, is within the margin (≤ 1), then the signal function S returns a 1; denoting that it contributes to the margin density. For other probabilistic classifiers, such as logistic regression, which return probability of class +1 or -1 as $p(y = +1|x)$ and $p(y = -1|x)$, the confidence is computed as $|p(y = +1|x) - p(y = -1|x)|$ and the threshold θ_{margin} (typically taken as 0.5) is used to specify the cutoff for uncertain samples. Samples with confidence less than θ_{margin}, contribute to the margin density.

Classifier without explicit margins

Classifiers such as decision trees [153] and K-nearest neighbors [151], return discrete

Algorithm 4.1: Random Subspace Ensemble

Training:

for *i= 1,2,3,...,K:* **do**

  Select J random features from all D features

  Construct C_i in X_J and add it to ensemble E

Classification:

Use majority voting to provide prediction on the input sample x.

$y(x) = argmax_y(votes(y))$

class labels and they do not have any intuitive notion of margin. These models are considered unstable [121], and they reduce the number of features necessary to build the models. In order to make them robust and to distribute weights across features, they are used with a feature bagging ensemble technique. Feature bagging improves generalization of unstable classifiers, by training multiple base models of the classifier on different subset of features, from the original D-dimensional data space [118]. Random subspace [123, 181] is an implementation strategy for feature bagging and is given by Algorithm 4.1. The entire feature space of D features is divided into K randomly chosen subspaces, with J features each, and a classifier is trained on each of these subspaces. The resulting ensemble E has classifiers $C_i; i = 1..K$, and majority voting is used to predict the final label y for a given sample x. By employing random subspace ensemble, any base classifier type can be made robust and the margin density signal can be extracted from it.

In Algorithm 4.1, the set of K models in the ensemble D, are trained on different views of the feature space and serve as a committee of experts with independent views on the prediction problem. An increased disagreement between the models is indicative of high uncertainty over a sample. The margin density MD for this type of models is computed by measuring the number of samples which have high uncertainty, as given by Equation 4.8. The Signal function $S_E(x)$, checks to see if the sample x has certainty less than θ_{margin} (typically taken as 0.5). In the equation below, p_E refers to the voted mean predicted class probabilities of the base estimators in the ensemble.

$$MD_{RS} = \frac{\sum S_E(x)}{|X|}; \forall x \in X$$

$$where,$$

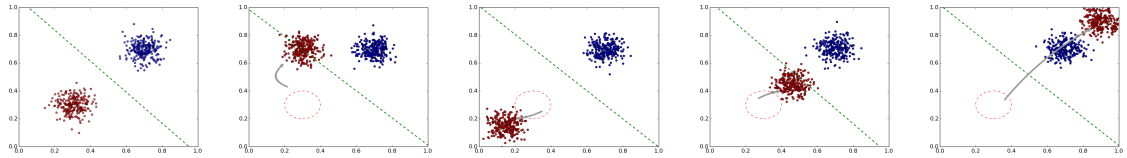
$$S_E(x) = \begin{cases} 1, & if \quad |p_E(y=-1|x) - p_E(y=+1|x)| \\ & \quad \leq \theta_{margin} \\ 0, & otherwise \end{cases} \tag{4.8}$$

The set of unlabeled samples X is collected and the ratio of samples which have critical uncertainty ($\leq \theta_{margin}$), is given as the margin density. The ensemble E can be comprised of classifier of any type (even heterogeneous classifiers can be considered), making the margin density approach applicable irrespective of the choice of the classification algorithm used.

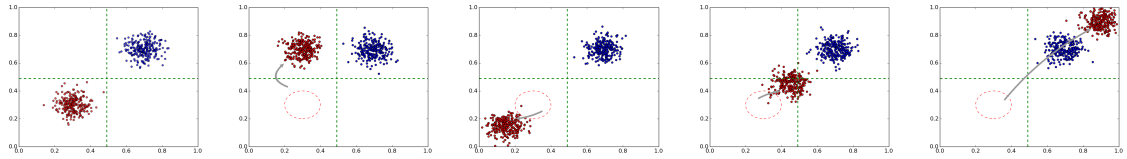
4.3.2.2 Change in margin density (ΔMD) as an indicator of drift

To understand behavior of the margin density metric and its efficacy as an indicator of drift, it is evaluated here on a synthetic dataset, under different change scenarios. A change scenario is setup by generating an initial distribution of 500 samples, used for training a model, and then generating 500 additional samples from a changed distribution, for testing the model. The change in margin density (ΔMD) is evaluated as the difference in margin densities of the training and test data: $\Delta MD = |MD_{Train} - MD_{Test}|$. By comparing ΔMD with changes in the training and testing error (ΔErr), which is representative of a metric used by fully labeled drift detectors, the effectiveness of MD to detect true drifts is evaluated. Similarly, a comparison with traditional feature based unlabeled drift detection techniques is evaluated, by checking the Hellinger distance between the distributions (ΔHD) [64]. Since ΔErr and ΔMD are within the range [0,1] and ΔHD has a range of $[0,\sqrt{2}]$, the ΔHD values were normalized to [0,1] by dividing the values by $\sqrt{2}$ in all the following experiments. The margin density metric was evaluated for a linear kernel SVM and for a Random subspace ensemble with 2 orthogonal C4.5 decision trees.

Experiments on a 2D synthetic dataset with two classes, using an SVM and a random



(a) A0 - Initial(SVM) (b) A0-A1 (c) A0-A2 (d) A0-A3 (e) A0-A4



(f) A0 - Initial(RS) (g) A0-A1 (h) A0-A2 (i) A0-A3 (j) A0-A4

Figure 4.8: Drift Scenarios with SVM (top) and random subspace (RS) (bottom) on 2D synthetic dataset. A0 is the initial distribution. A1-A4 represent different drift scenarios.

TABLE 4.2

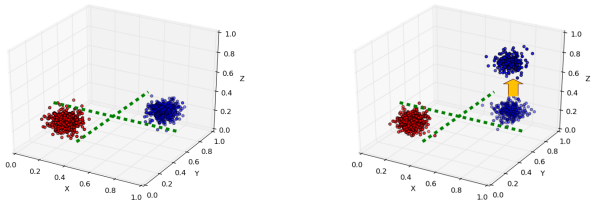
Results of change detection metrics ΔErr, ΔMD and ΔHD, on synthetic drifting scenarios.

Base Model-	**SVM**			**RS**		
Drift Scenario	**Err**	**MD**	**HD**	**Err**	**MD**	**HD**
A0-A0 (Baseline)	0	0.01	0.12	0	0	0.12
A0-A1	0.19	0.43	0.33	0.5	0.5	0.33
A0-A2	0	-0.04	0.58	0	0	0.58
A0-A3	0.05	0.42	0.58	0.1	0.17	0.58
A0-A4	0.5	-0.05	0.69	0.5	0	0.69
B0-B0 (Baseline)	0	0.03	0.1	0	0	0.1
B0-B1	0	0.02	0.27	0	0	0.27
C0-C0 (Baseline)	0.01	0.02	0.11	0.08	0.12	0.11
C0-C1	0.41	-0.72	0.82	0.41	-0.19	0.82

subspace(RS) model are depicted in Figure 4.8. A0 represents the initial training distribution of the samples and A1-A4 represent 4 different change situations. A change from A0-A0, was considered as a control experiment to denote changes due to random sampling, as shown in Table 4.2. The drift scenario A0-A1, represent changes which directly affects classification boundary due to drift in one of the features. This causes the error to increase by 19% for SVM and 50% for RS. Correspondingly, the MD changes by an average of 0.47 and the HD changes by 0.33. Since the Hellinger distance is computed from the unlabeled data, independently of the learned classifier, the ΔHD values for SVM and RS are same for

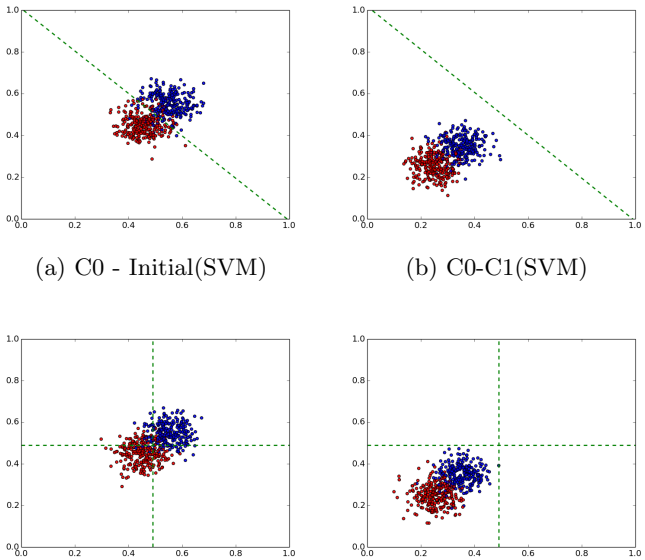
any given scenario, shown in Table 4.2. Change scenarios A0-A2 and A0-A3, represent shift of equal magnitude but opposite direction. In A0-A2, the shift is away from the margin, while in A0-A3 the shift is towards the margin, as shown in Figure 4.8c) and d) for SVM, and h) and i) for RS models. Both these scenarios result in the same change in ΔHD of 0.58, indicating shortcomings of traditional feature tracking approaches in differentiating false alarms from relevant changes. The MD metric shows no change for the A0-A2 scenario but detect the A0-A3, consistent with the error tracking approach. By using a fixed margin and tracking its density, changes away from the margin are effectively ignored, as they rarely cause performance degradation. This property of the margin density approach makes it more resistant to false alarms, compared to other margin based methods [107,175]. An extreme data distribution shift is seen in Figure 4.8e) and j), which is representative of a drastic drift affecting all features simultaneously; a situation rare in real world applications. This change occur away from the margin and goes unnoticed by the ΔMD metric. They are however, tracked by the ΔHD and ΔErr methods. These changes are rare in real world operational environments and can be more effectively caught by novelty detection methods described in Section 4.2.2.1, which are designed for such changes. The Hellinger distance metric, in an attempt to provide completeness in drift detection, leads to excessive false alarms. This effect is exacerbated in high dimensional datasets, where there are many irrelevant features, which do not contribute to the classification process. This is illustrated in Figure 4.9, where the Z-dimension is not useful to the prediction task. A drift in the Z-direction leads to a false alarm by ΔHD but is correctly ignored by ΔErr and ΔMD, as seen for the entry B0-B1 in Table 4.2.

An additional scenario C0 is presented, to indicate the need for tracking both a drop as well as a rise in the margin density. The scenario C0-C1 as shown in Figure 4.10, is common in case of non linear or tightly packed class distributions, which causes the initial margin density to be high. This drift leads to a drop in the margin density, indicated by negative values in Table 4.2, indicating the need to track the absolute value of margin density change :$|\Delta MD|$. This is in contrast to the ΔErr and ΔHD metric, where only a



(a) B0 - Initial(RS) (b) B0-B1

Figure 4.9: Drift Scenario in 3D synthetic dataset, change occurs along Z-dimension, which is irrelevant to the classification task.



(a) C0 - Initial(SVM) (b) C0-C1(SVM)

(c) C0 - Initial(RS) (d) C0-C1(RS)

Figure 4.10: Drift scenario causing drop in margin density, with SVM model (top) and random subspace model (RS) (bottom). C0 is the initial distribution of samples.

spike in error rate or Hellinger distance is considered relevant to change detection.

The analysis in this section indicates the ability to use the change in margin density (ΔMD), as a signal for drift detection. The MD metric signals change, when it is relevant to the classification process, while providing high robustness against stray changes. Change in irrelevant features, in high dimensional spaces and in regions away from the classifier's margin, are effectively filtered. This effect was observed for both - classifiers with explicit margin and by using a random subspace ensemble, for cases where margin is not explicit.

4.3.3 The Margin Density Drift Detection (MD3) algorithm

The MD3 algorithm, being a streaming data algorithm, needs to operate with limited memory of past information, to continuously process data indefinitely, and has to provide a quick response time. The change in margin density is used as a metric for detecting drift in a streaming data environment. The incremental classification process, continuously receives unlabeled samples X and predicts their class labels Y, based on the classification model C, as shown in Figure 4.11. At any given time t, the signal function $S(X_t)$, computes if the sample X_t lies within the margin of C. This computation is performed using Equations 4.7 and 4.8, based on the type of model used. This signal is used to update the expected margin density. A significant change in the margin density at time t (MD_t), signals a change which requires further inspection. Following this, the next N_{train} samples are requested to be labeled by an external Oracle. The Oracle can be any entity which can provide true labels for the unlabeled sample X_t, at a given cost. If the performance of C, on the N_{train} labeled samples, is found to have degraded, a drift is confirmed and the model is retrained using these collected labeled samples. In the MD3 approach, there is no need for continuous monitoring using labeled samples, as the drift detection process is unsupervised. Labeling is requested only when a drift is suspected, for confirmation and retraining. The MD metric reduces the need for frequent confirmation, owing to its robustness toward irrelevant changes, making the labeling process essentially for the retraining phase only.

The MD3 algorithm (Algorithm 4.2), begins with an initial trained classifier C, which is obtained by learning from the initial labeled training dataset, before the model is made online. From this initial training dataset, a reference distribution, summarizing margin and performance characteristics of the dataset, is learned. This reference distribution comprises of the expected margin density - MD_{Ref}, the acceptable deviation of the margin density metric- σ_{Ref}, expected accuracy on the training dataset - Acc_{Ref} and its deviation σ_{Acc}. These values are learned from the training dataset by using the K-fold cross validation technique, commonly used for evaluating classifiers [178]. In the cross validation method, the entire dataset is sequentially divided into K bands of samples. In the first iteration, the

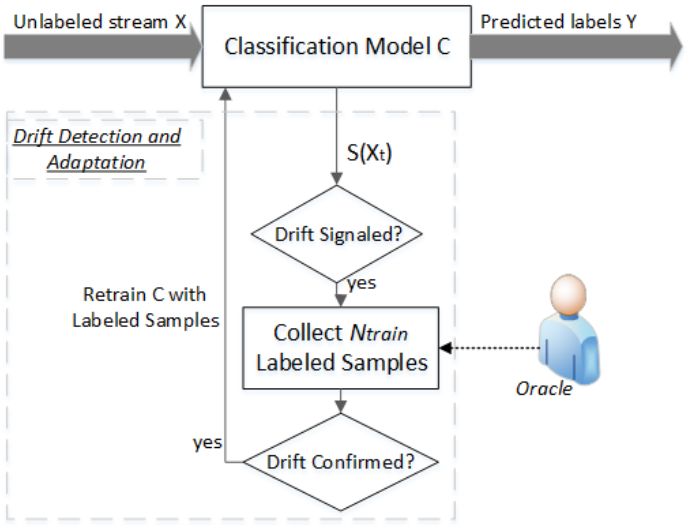


Figure 4.11: Overview of the MD3 algorithm in a stream classification setting.

first K-1 bands are used as a training dataset to learn model C and then the K^{th} band is used to test the model. The process is repeated K times, where each band functions as the test set exactly once. Accuracy and Margin density values from the K test sets are considered results of random experimentation. The average values and standard deviation, of the test accuracy and margin density, over the K iterations is used to form the reference distribution. Cross validation allows to create a population of the metric values, to better estimate their expected values and acceptable deviation. These values are then used to signal change based on the desired level of sensitivity, given by parameter θ. Change is signaled when the margin density at a time t, given by MD_t, deviates by more than θ standard deviations from the reference margin density value MD_{Ref}, as given by Equation 4.9. The same sensitivity parameter is used to detect significant drop in performance, for the obtained labeled samples, from the reference accuracy values as per Equation 4.10.

$$if \quad |MD_t - MD_{Ref}| > \theta * \sigma_{Ref} \quad \Rightarrow Drift \quad suspected \tag{4.9}$$

$$if \quad (Acc_{Ref} - Acc_{LabeledSamples}) > \theta * \sigma_{Acc} \quad \Rightarrow Drift \quad confirmed \tag{4.10}$$

Here, *LabeledSamples* is the set of N_{train} samples, which were requested to be labeled once a significant drift is suspected by Equation 4.9. A drop in accuracy confirms that

Algorithm 4.2: The MD3 algorithm

Input : Unlabeled stream X, Initially trained model C, Reference distribution (MD_{Ref}, σ_{Ref}, Acc_{Ref}, σ_{Acc}). *Parameters:* Sensitivity Θ, Stream progression $\lambda = (N-1)/N$(where N is the chunk size), N_{train}(= N by default)

Output: Predicted label stream Y

```
1  MD_0 = MD_Ref
2  currently_ drifting = False
3  LabeledSamples = ∅
4  for t= 1,2,3,...: do
5      Compute margin inclusion signal-
         S(x = X_t) = { 1,  if  X_t  in  margin
                      { 0,  otherwise
6      Update MD_t = λ * MD_{t-1} + (1 − λ) * S(X_t)
7      if |MD_t − MD_Ref|>Θ * σ_Ref then
8          currently_drifting = True                    ▷ Drift Suspected
9          LabeledSamples←Collect N_train labeled samples by querying Oracle
10     if currently_ drifting and |LabeledSamples|== N_train then
11                                  ▷ Enough labeled samples to make decision
12         if (Acc_Ref - Acc_LabeledSamples)>Θ * σ_Acc then
13             Retrain C with LabeledSamples            ▷ Drift Confirmed
14         Update Reference distribution (MD_Ref, σ_Ref, Acc_Ref, σ_Ref)
15         currently_ drifting = False
```

the change is indeed a result of concept drift and that model retraining is necessary, to update the classifier C. Once retraining is performed, a new reference distribution (MD_{Ref}, σ_{Ref}, Acc_{Ref}, σ_{Acc}), is learned from the *LabeledSamples* set, based on the K-fold cross validation technique described above. By allowing users to specify the intuitive parameter of sensitivity, suggested to be picked in the range of [0,3], the entire change detection process is made flexible to be used in different streaming environments. A larger value can be set if frequent signaling is not desired, alternatively a lower value could be used for critical applications, where small changes could be harmful if undetected.

The drift detection process, set around tracking the margin density signal MD, is made incremental by using the moving average formulation of Equation 4.11. Here, the margin density at a time t, given by MD_t, is computed incrementally by using a forgetting factor λ on - MD_{t-1}, and combining it with the signal function $S(X_t)$, which indicates if

the current sample X_t falls within the margin of the classifier C.

$$MD_t = \lambda * MD_{t-1} + (1 - \lambda) * S(X_t) \tag{4.11}$$

The parameter λ is the forgetting actor for the stream, and it can be computed by specifying the *chunk of influence* parameter-N. The λ is computed as $\lambda = (N-1)/N$. This formulation makes it applicable as a stream monitoring system, by making incremental updates to the margin density metric. It should be noted that, here incremental formulation is specified for the drift monitoring process only. This is irrespective of the stream classification algorithm used, which could process data either - incrementally, by chunk or by using a sliding window. Separating the detection and classification schemes, makes the MD metric more general in its implementation, to be used in different classification setups. This also ensures that we can perform controlled testing of the efficacy of the drift detection methodology.

4.4 Experimental results and analysis

This section presents experimental analysis and results of the proposed MD3 approach, on two sets of experiments: Section 4.4.2 presents results on drift induced datasets, to better understand drift detection characteristics of the framework in a controlled environment; Experiments with real world drifting data are presented in Section 4.4.3 and Section 4.4.5, to demonstrate its practicality. Experimental comparisons with a fully labeled drift detection technique- the AccTr approach based on EWMA [165], and the unlabeled drift detection approach of- HDDDM [64], were performed. Two variant of the MD3 approach were used. MD3-SVM uses a linear kernel SVM as the base model, while MD3-RS uses a random subspace implementation of the margin density approach. Details about experimental methods and setup are presented in Section 4.4.1. Effects of varying the margin width parameter θ_{margin} and that of varying the detection model are also presented, in Section 4.4.4.

4.4.1 Experimental methods and setup

Experimental methods used for comparative analysis

The drift detection on data streams, are evaluated and compared using the following techniques.

- *Static baseline model (**NoChange**)*: This approach assumes that data is static with no drift over time. As such, no change is signaled, model is never updated and no labeling is requested. This is the lower baseline and any approach should atleast be better than a NoChange approach.

- *Fully labeled Accuracy Tracking (**AccTr**)*: This model forms the upper baseline for the drift handling mechanisms. All data is assumed to be labeled and the explicit tracking of accuracy is used to signal change. An unlabeled drift detection mechanism is effective if its performance is close to the AccTr approach. The AccTr approach is illustrated in Figure 4.12, where every predicted sample's correct label is requested from an Oracle and the accuracy is checked to see if it has significantly deviated from the training accuracy. The accuracy is tracked incrementally by using the EWMA [165] formulation of change tracking as given by Equation 4.4.

- *MD3 using SVM model (**MD3-SVM**)*: This approach uses the SVM based implementation of MD3. A linear kernel SVM, with hinge loss is used. Margin density is computed by tracking number of samples in the classifier's margin.

- *MD3 using random subspace(RS) model (**MD3-RS**)*: The proposed random subspace drift detection technique of MD3 is used. 20 C4.5 decision trees are used as the ensemble, each one has 50% of the features randomly picked from the feature space. Margin density is given by number of samples in regions of high uncertainty (high disagreement) of the ensemble. The threshold for critical uncertainty was chosen as 0.5, and samples with confidence less than 0.5 are considered to be in the margin.

- *Hellinger Distance Drift Detection Methodology (**HDDDM**)*: The approach obtained

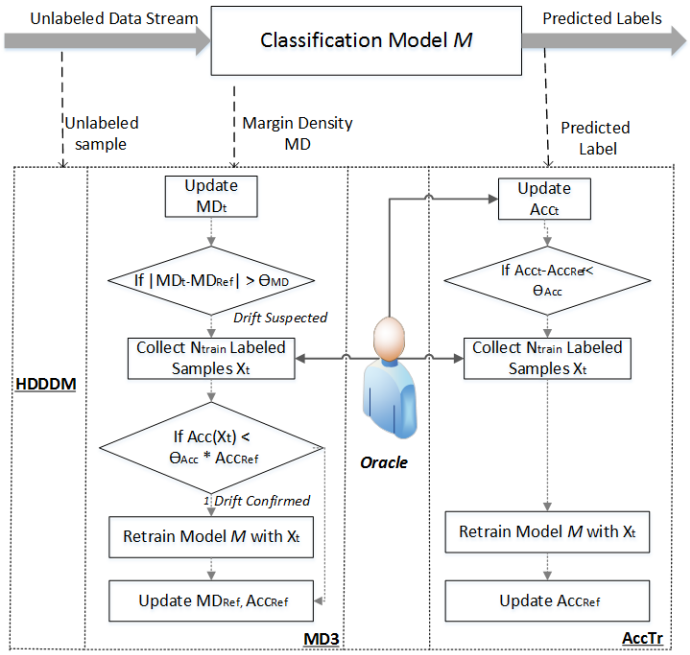


Figure 4.12: Unlabeled (HDDDM), Fully Labeled (AccTr) and the Margin Denstiy(MD3) drift detection techniques, showing portion of stream that they track.

from [64], is representative of traditional unlabeled approaches found in literature, which track changes to feature space. In particular, the HDDDM approach tracks the average Hellinger distance of all samples within two distributions and signals change when the distance increases beyond a threshold. Hellinger distance is a popular metric in streaming data research, and a comparison using this will enable us to highlight the fundamental differences between the unlabeled approaches and the MD3 approach.

These methods provide representations of the main different paradigms of drift detection: Explicit detector(labeled) and Implicit detectors(unlabeled), as presented in Section 4.2. A comparative analysis of these methods will highlight the efficacy of the MD3 approach and its place in the literature on drift detection techniques.

Experimental setup

To ensure that bias due to the underlying classification process does not affect our analysis of the detection scheme, all approaches were implemented in an incremental manner using the moving average formulation of Equation 4.12.

$$Metric_t = \lambda * Metric_{t-1} + (1 - \lambda) * S \tag{4.12}$$

The metric at time t depends on the signal function *S(.)*, which is defined based on the detection method used. The AccTr approach uses error as a signal. If predicted label is different from the correct label, the signal is 0, otherwise it is 1. For the MD3 approaches, the signal is obtained by the margin inclusion test, as given by Equation 4.7 for MD3-SVM and Equation 4.8 for the MD3-RS methods. The HDDDM approach is not incremental by nature. It is a chunk based approach, as computing histograms of data needs an entire chunk of data. We modified this approach, such that a chunk is defined incrementally, sliding at a rate of one sample. For a given time t, the chunk comprises of $t - N$ latest samples, where N is the chunk size. The λ forgetting factor in Equation 4.12 is taken as $(N - 1)/N$ for AccTr and MD3 approaches, to ensure equivalence in the drift detection evaluation as compared to the HDDDM approach.

The initial 15% of the stream is assumed to be labeled. This forms the initial training set from which the classifier model C is learned, along with the reference distribution metrics (expected metric and acceptable deviation). The reference distribution was obtained via 5 fold cross validation on the training data, as described in Section 4.3.3, for the AccTr and MD3 approaches. For the HDDDM approach, the reference distribution and acceptable deviation was obtained by using a sliding window of 3*N unlabeled samples with a slide rate of N/3 samples. The Hellinger distance between the subsequent chunks in this sliding window, formed the population for learning the expected Hellinger distance and its standard deviation. A sensitivity of θ=2 (from the suggested range of [0,3]), was chosen to balance robustness with reactivity, for all the drift detection methods. After a drift is indicated, the number of labeled samples to be requested - N_{train} is taken to be equal to N. This ensures that the same number of recent samples which indicated a drift, are used to confirm the drift and retrain the classifier.

All experiments were performed using Python 2.7 ad scikit-learn machine learning library [149]. Support Vector Machine with a linear kernel and regularization constant of

1.0, was chosen as the prediction model for all the experiments. In order to ensure that differences between detection techniques MD3-SVM and MD3-RS are not a result of the different training capabilities of the SVM and RS classifiers, the task of prediction and detection were separated for the MD3-RS approach. The training dataset was used to train two models: a linear SVM and a RS model. While, the SVM was used to provide prediction as the online classifier C (Figure 4.11), the RS model was used solely for the purpose of detection of drift and for triggering retraining of C. This setup enables us to analyze the drift detection properties of the two approaches, by blocking out their different prediction behavior.

4.4.2 Experiments on drift induced datasets

This section presents experimental evaluation of the MD3 approach on static datasets, which were induced with concept drift in a controlled manner. By controlling the location and nature of drifts in these datasets, a better understanding of the drift detection capabilities of the different approaches, in a real world setting, is obtained. Six datasets were chosen from the UCI machine learning library [145], and they were preprocessed to have only numeric and binary values, normalized in the range of [0,1]. The multiclass datasets were reduced to a binary class problems and the data instances were shuffled randomly, to remove any unintended concept drifts already in the data. The characteristics of the datasets is shown in Table 4.3. The chunk size parameter N, shown in Table 4.3, is used to process the stream based on the number of instances present in the datasets.

The drift induction process is explained next, followed by experimental results and analysis on the drift induced data.

Inducing concept drift in datasets

The drift induction process of [107], provides a way to include a single concept drift in static datasets, at a particular location in the data stream. This allows for controlled drift analysis, while at the same time retaining properties of the real world applications from which the the

TABLE 4.3

Characteristics of datasets chosen for drift induction experiments.

Dataset	#Instances	#Attributes	Chunksize N
Digits08	1499	16	150
Digits17	1557	16	150
Musk	6598	166	500
Wine	6497	12	500
Bank	45211	48	2500
Adult	48842	65	2500

dataset is derived. The dataset is first shuffled to remove any unwanted concept drift and to prepare it for the drift induction process. The drift induction process of [107], induces feature drift in the dataset, after a point in the stream, called the *ChangePoint*. Drift is induced by randomly picking a subset of the features and rotating their values, for a particular class. For example, if the feature (1,5,7) are picked for class label 0, after the *ChangePoint*, the instances belonging to class 0 have features (1,5,7) shuffled as (7,1,5). This basic approach ensures that feature drifts are induced and also, the original data properties of the dataset are maintained. This approach is however dependent on the features selected for rotation and it provides erratic results if the 'right' set of features are not selected.

Our drift induction approach, proposed here, extends the basic idea of [107], and allows for greater control over the nature of change. Instead of randomly picking a set of features, to be rotated, we pick features based on their importance to the classification task. This is done by ranking the features, based on their information gain metric [182], and then selecting features form the top or bottom of the list based on the nature of change desired. Two sets of experiments were performed: a) The *Detectability* experiments, which choose the top 25% of features from the ranked list, and b) The *False alarm* experiments, which choose the bottom 25%. This was done to test the detection capabilities and robustness to irrelevant changes, respectively. The top 25% of the ranked features have a high impact on the classification task, as these are ranked based on their information content, and modifying these features results in model degradation, which is necessary to be detected and fixed. Modifying the bottom 25% of the features, has less impact on the classification process and

TABLE 4.4

Effects of shuffling the top 25% and the bottom 25% feature, on the test accuracy.

Dataset	Train	Test-Original	Test-Top 25%	Test-Bottom 25%
Digits08	97.5	97.1	77.6	95.3
Digits17	99.5	99.6	60.1	99.9
Musk	93.9	91.9	82.7	90.1
Wine	100	100	67.6	100
Bank	83.3	83.2	56.2	84.7
Adult	85	85.2	58.6	85.4

results in false alarms, which should be ignored by the detectors.

The effect of changing the top 25% and bottom 25% of the features on the 6 UCI datasets is shown in Table 4.4. In all the datasets, the *ChangePoint* was induced at 50% of the stream. A model is trained on the data before the *ChangePoint*, and tested on the samples after this point. The similarity in accuracy of the model on the training and original test set (before induction), indicates an initial static dataset. For the *Detectability* experiments, the top 25% of the features are rotated and this results in a significant drop in the test accuracy, after *ChangePoint*, as seen in Table 4.4. This indicates true drifts, which need to be detected by a drift detection algorithm. Rotating the bottom 25% of the features in the *False alarm* experiments, does not show any significant drop in test accuracy. Although the same number of features are rotated in both cases, features have different levels of relevance, when it comes to the classification task. We perform our experimental analysis on the top 25% and bottom 25% datasets, to analyze behavior of the different algorithms, under different change conditions.

Experimental results on drift induced datasets

The NoChange, AccTr, MD3-SVM, MD3-RS and the HDDDM methodologies, were analyzed in the *Detectability* and the *False alarm* experiments. The number of drifts detected, the false alarms raised and the accuracy over the stream, is reported in Table 4.5. A drift is detected if there is a significant change in the metric being tracked. This results

in requesting of N (Table 4.3) labeled samples, to confirm if the deviation leads to a drop in the accuracy. A false alarm is the result of a change signal, which after obtaining labeled samples was found to have no significant effect on the classification performance. In case a false alarm is reported, no retraining of the classifier takes place and the labeled samples are discarded. In the *Detectability* experiments, there is exactly one true drift induced, which causes accuracy to drop after the midpoint of the stream. In case of the *False alarm* experiments, the induced change does not affect performance and as such exactly one non relevant change is introduced in these experiments.

The *Detectability* experiment causes the model's performance to drop over time, which is evident from the low accuracy of the NoChange model, as seen in Table 4.5. This indicates the need for a drift detection methodology, to deal with the induced drift. The AccTr approach, directly monitors the classification performance with labeled samples, as such it serves as the gold standard for detecting drifts, in our experiments. This approach detect exactly 1 drift in all 6 cases, indicating its robustness against false alarms. The AccTr, HDDDM and the MD3-SVM and MD3-RS, techniques are all able to detect atleast one drift in these experiments (*Detectability*) and as such are able to reach similar final accuracies. The MD3-RS and MD3-SVM are unsupervised methods and are still able to reach accuracy similar to the fully labeled AccTr approach (average difference of $<1\%$ for both cases). This indicates the ability of the MD3 approaches to be used instead of the labeled approaches, without significantly compromising on the prediction performance.

The resistance to false alarms, is shown by the number of drifts detected in the *False alarm*, in Table 4.5. Changes in these experiments do not cause a significant performance degradation. As such, a drift detected does not lead to retraining of the classifier, resulting in a false alarm. While the AccTr and both the MD3 approaches are resistant to such changes, the HDDDM approach signals it as a relevant change which needs further inspection. The HDDDM approach does not differentiate between the change in the top 25% features vs the change in the bottom 25% features, as it is a classifier agnostic technique which relies solely on tracking the changes in the raw feature values distribution. The HDDDM approach

TABLE 4.5

Detection results on drift induced datasets for the *Detectabiltiy* experiments and the *False alarm* experiments.

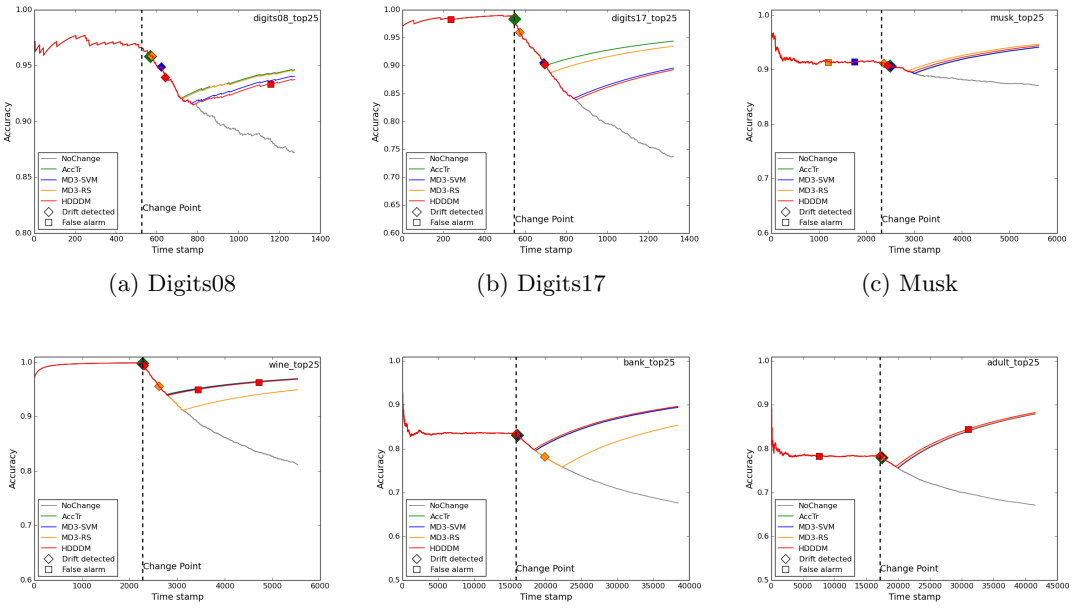
Dataset	Methodology	Top 25%- (Detectability Expts)			Bottom 25%- (False alarm Expts)
		Accuracy	Drifts detected	False Alarms	Drifts detected
Digits08	NoChange	86.4	0	0	0
	AccTr	94.4	1	0	0
	MD3-SVM	93.8	1	0	0
	MD3-RS	94.4	1	0	0
	HDDDM	93.5	2	1	1
Digits17	NoChange	71.9	0	0	0
	AccTr	94.2	1	0	0
	MD3-SVM	89	1	0	0
	MD3-RS	93.2	1	0	0
	HDDDM	88.7	2	1	3
Musk	NoChange	87	0	0	0
	AccTr	94.1	1	0	0
	MD3-SVM	94	2	1	0
	MD3-RS	94.8	2	1	0
	HDDDM	94.3	1	0	1
Wine	NoChange	80.1	0	0	0
	AccTr	96.9	1	0	0
	MD3-SVM	96.9	1	0	0
	MD3-RS	94.9	1	0	0
	HDDDM	96.9	3	2	1
Bank	NoChange	67.5	0	0	0
	AccTr	89.4	1	0	0
	MD3-SVM	89.4	1	0	0
	MD3-RS	85.3	1	0	0
	HDDDM	89.6	1	0	3
Adult	NoChange	67	0	0	0
	AccTr	87.9	1	0	0
	MD3-SVM	87.9	1	0	0
	MD3-RS	87.9	1	0	0
	HDDDM	88.2	3	2	3

causes a higher false alarm rate, than the MD3 approaches, on both experiments. Another observation is that the MD3-SVM and MD3-RS show similar behavior, on average, with only a deviation of 0.25% in accuracy and the exact same number of drifts detected. This shows the generic applicability of the margin density signal irrespective of the implementation technique used to compute it.

Accuracy over time, for the *Detectability* experiments, is shown in Figure 4.13. After

the *ChangePoint*, a significant drop in accuracy is seen in all cases. The drift detection approaches are swift in recognizing this change and after retraining (seen as point after which accuracy starts to rise again), they are able to again provide high prediction performance. The NoChange approach (gray), does not detect any drift and as such the performance continues to degrade in this case. The MD3 approaches, have accuracy trajectories close to the AccTr detector, indicating its use as a surrogate to the fully labeled approach. This accuracy is higher, in most cases, than the HDDDM approach. While the MD3 approach led to a false alarm on the Musk dataset only, the HDDDM approach is seen to trigger changes multiple times over the course of the stream. These changes are frequent (atleast 1, on average), and often occur without any correlation with the change in accuracy values. The MD3 approaches detect drifts in close proximity to the AccTr approach, seen by minimal lag between the green diamonds and- the orange and the blue diamonds. The MD3-RS approach is more robust to change, which causes it to detect drifts with a delay in the Figure 4.13 d) and e). However, the drift detection for all approaches is close to the *ChangePoint*, illustrating their ability to detect the true drift effectively.

Progression of the Accuracy and Margin Density metrics over time, for the *Detectability* experiment is shown in Figure 4.14. A drop in the accuracy (green) is seen after the *ChangePoint*. This is accompanied by significant spike in the MD3-RS (orange) metric. The metric MD3-SVM(blue), shows either a significant spike (in Figure 4.14 a), b) and d)), or a drop in margin density (in Figure 4.14 c), e) and f)). The margin density metric, similar to the accuracy metric, shows a high signal-to-noise ratio. It stays stable before the *ChangePoint* and after the retraining is performed (indicated by circles in the plots), with a significant deviation only when drift occurs. This confirms that the drift detection is a result of the informativeness of the margin density metric, in detecting drifts, and not due to random variations in the data stream.



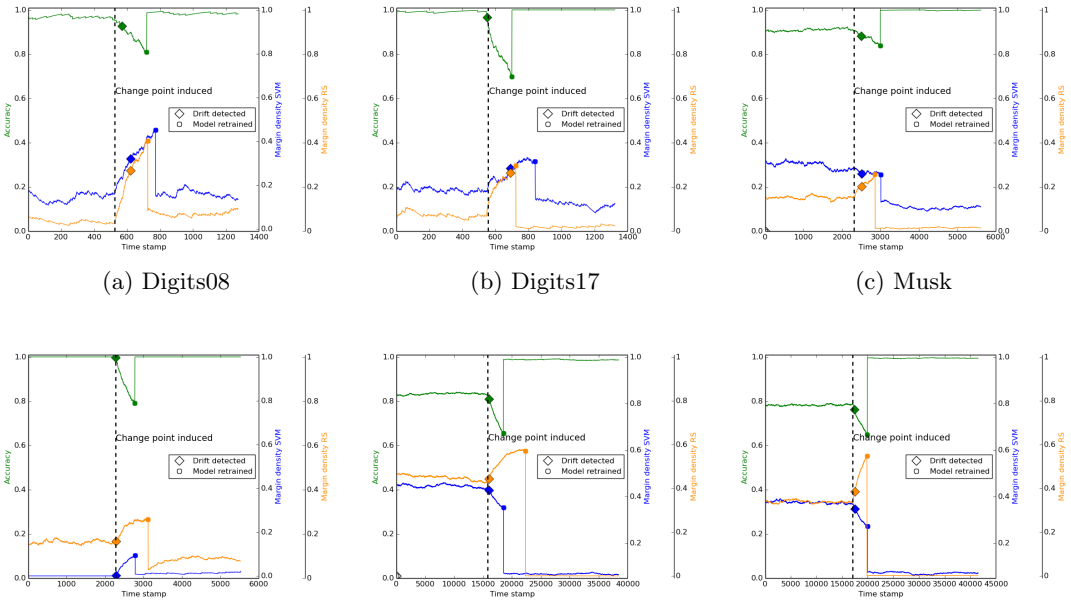
(a) Digits08 (b) Digits17 (c) Musk

(d) Wine (e) Bank (f) Adult

Figure 4.13: Accuracy over time for the NoChange (gray), AccTr (green), MD3-SVM (blue), MD3-RS (orange) and the HDDDM (red) approach on the *Detectability* experiments. True drifts detected are shown as diamonds and squares represent false alarms.

4.4.3 Experiments on real world cybersecurity datasets exhibiting concept drift

Machine learning models deployed in real world applications, operate in a dynamic environment where concept drift can occur at any time. Such drifts are not only plausible, but in fact expected and rampant in the domain of cybersecurity, where attackers are constantly trying to generate data that degrades the classifier. In this section, 4 real world concept drift datasets are chosen from the domain of cybersecurity, as presented in Table 4.6. These datasets are high dimensional and popularly used in machine learning literature, to test online binary classification models in a concept drifting environment. The spam and spamassassin datasets taken from [183,184], represent the task of separating malicious spam email from legitimate ones. Phishing [145] contains data about malicious web pages and the nsl-kdd dataset [185] is derived from the task of intrusion detection systems, which filters malicious network traffic. All the datasets were preprocessed by converting feature to numeric/binary types only, and by normalizing each feature value to the range of [0,1].



(a) Digits08 (b) Digits17 (c) Musk

(d) Wine (e) Bank (f) Adult

Figure 4.14: Accuracy (green), Margin Density for SVM (blue) and Margin Density for RS (orange), over time, in the *Detectability* experiments. Drifts detected are denoted by diamonds and circles denote the retraining point.

TABLE 4.6

Description of real world concept drift datasets, from cybersecurity domain.

Dataset	#Instances	#Attributes	Chunk Size - N
spam	6213	499	500
spamassassin	9324	499	500
phishing	11055	46	500
nsl-kdd	37041	122	2500

The final data characteristics are shown in Table 4.6. These datasets exhibit concept drift, but the exact nature and location of the drifts is not known in advance.

The NoChange, AccTr, MD3-SVM, MD3-RS and HDDDM methodologies are evaluated on the 4 datasets. The metrics used for evaluation are: Accuracy of the stream, Number of drifts signaled, Number of false alarms and the total percentage of samples which were requested to be labeled. Accuracy determines the predictive performance of the online system. Number of drifts signaled indicates sensitivity to change. False alarms occur when a drift is signaled, but upon obtaining labeled samples it was found that performance

has not significantly degraded, thus requiring no retraining. Since the location and number of drifts is not known in advance, the false alarm refers only to cases which did not lead to a retraining of the models, causing the requested labels to go wasted. False alarms are 0 for the case of the AccTr approach as this approach directly tracks drop in accuracy, unlike the other unlabeled techniques. False alarms refers to situation where Equation (4.9) is triggered due to a suspected drift, but upon receiving labeled samples we confirm that retraining is not needed (Equation (4.10)), as the accuracy has not degraded significantly. The labeling% indicates the cost expended by the methodology (in term of labels requested) and is directly related to the number of false alarms, as every alarm leads to requesting of Chunk Size- N(shown in Table 4.6) samples to be labeled. A high accuracy, high drift detection, low false alarm and low labeling% is desirable.

In all cases, the accuracy of the NoChange approach is significantly lower than the other drift detection techniques, as observed from Table 4.7. This confirms the drifting nature of the datasets and the need for drift detection. The accuracy obtained by the margin density methodologies is close to the fully labeled AccTr approach, with MD3-SVM having an average deviation of 1.3% and the MD3-RS having a 0.9% deviation, only. This indicates the ability of these techniques to detect drifts, as good as a labeled drift detection mechanism. The labeling requirement for the MD3 approaches is 88.3% less than the AccTr approach, which relies on a totally labeled stream for performing its computation.

The HDDDM approach, on average, performs poorly compared to the MD3 approaches, and also needs 8.3% more labeling. This is a result of its high false alarm rate. False alarms are harmful as it causes the system to react to every change in the data distribution, noise or otherwise, making adaptiveness a hassle than a solution. Especially in the domain of streaming cybersecurity applications, an overly responsive system is a serious problem, as it is vulnerable to malicious manipulation of the training process. Also, labeling is a time consuming and expensive task. A system which frequently requires manual intervention is less likely to be trusted and can cause experts to disregard its warnings. The HDDDM approach needed 1250 more labeled samples than the MD3 approaches, on

TABLE 4.7

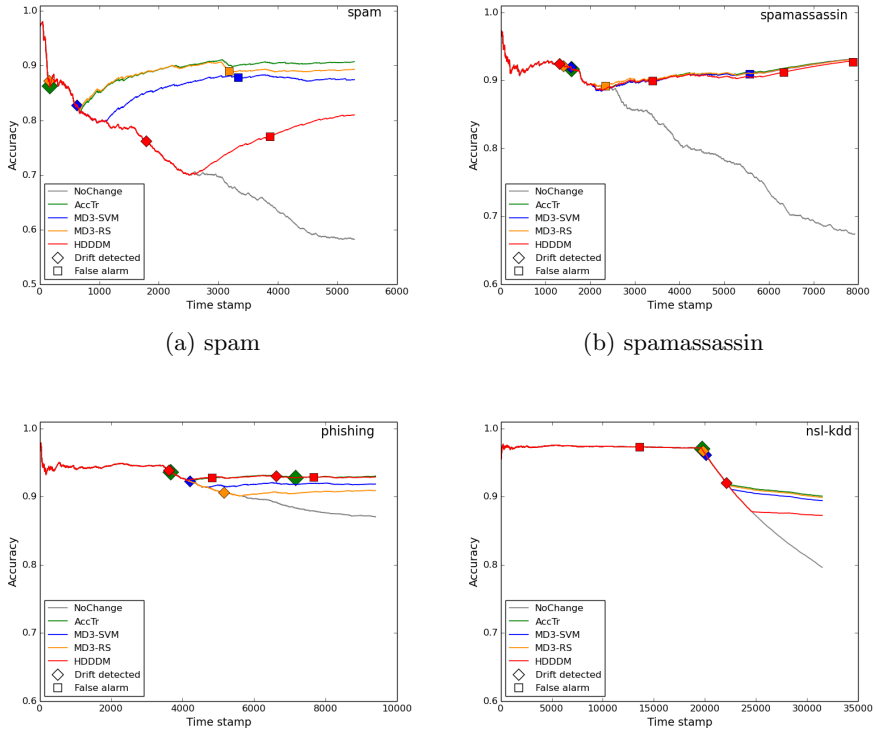
Results on real world concept drift datasets, from cybersecurity domain.

Dataset	Methodology	Accuracy	Drifts signaled	False alarms	Labeling %
spam	NoChange	57.5	0	0	0
	AccTr	90.6	1	0	100
	MD3-SVM	87.3	2	1	18.9
	MD3-RS	89.2	2	1	18.9
	HDDDM	80.7	2	1	18.9
spamassassin	NoChange	67	0	0	0
	AccTr	92.8	1	0	100
	MD3-SVM	92.8	2	1	12.6
	MD3-RS	92.8	2	1	12.6
	HDDDM	92.6	4	3	21.9
phishing	NoChange	86.9	0	0	0
	AccTr	92.9	2	0	100
	MD3-SVM	91.7	1	0	5.3
	MD3-RS	90.8	1	0	5.3
	HDDDM	92.8	4	2	21.3
nsl-kdd	NoChange	79.6	0	0	0
	AccTr	90.1	1	0	100
	MD3-SVM	89.4	1	0	7.9
	MD3-RS	89.9	1	0	7.9
	HDDDM	87.2	2	1	15.8

average, due to its increased false alarm rate. The MD3 approach on the other hand, signals change only when it would affect the system performance directly, making it suitable as a self-guided automatic monitoring system for drift detection.

Progression of accuracy over the 4 datasets, is shown in Figure 4.15. It is seen that the MD3 methodologies and the AccTr approach converge and behave similarly over time. This indicates that MD3 could be used as a replacement for a fully labeled drift detection approach. The similarity in the MD3-SVM (blue) and the MD3-RS (orange) approaches, in terms of the accuracy progression and the number of drifts/false alarms detected, indicates the general applicability of the margin density metric as a drift indicator metric. The margin density metric provides good performance irrespective of the type of machine learning technique it is implemented on, making it classification algorithm independent.

The HDDDM (red) approach performs worse than the other methods on the spam



(a) spam (b) spamassassin

(c) phishing (d) nsl

Figure 4.15: Accuracy over time for the NoChange (gray), AccTr (green), MD3-SVM (blue), MD3-RS (orange) and the HDDDM (red) approach on real world concept drift datasets. True drifts detected are shown as diamonds, and squares represent false alarms.

and the nsl-kdd datasets. This is attributed to the delay in its drift detection. In all datasets, the number of false alarms (squares) is high for the HDDDM approach, which translates to increased labeling expenditure as shown in Table 4.7. The MD3 approach, signals 1 false alarm in the spam and spamassassin datasets. In the spam dataset, the MD3-SVM and the MD3-RS approaches signal change at time-stamps 3829 and 3676 respectively (Figure 4.15a), which are reported as false alarms. However, these are not without basis, there is a small drop in the accuracy at these points, albeit not enough to warrant concept drift recovery. The margin density metric is sensitive to changes in accuracy, but is still robust compared to other feature tracking approaches (HDDDM), which seem to signal changes without any correlation to accuracy degradation. From the experiments, it can be concluded that the MD3 approach is a classification technique independent, high performing, unlabeled drift detection technique, which is robust to irrelevant data changes

TABLE 4.8

Results of using logistic regression (L1-penalty) as the detection model for MD3.

Dataset	Accuracy	Drifts signaled
spam	87.8	2
spamassassin	85.3	2
phishing	86.9	0
nsl-kdd	89.3	1

and is attractive for deployment in machine learning based cybersecurity systems, due to its reduced need for manual intervention.

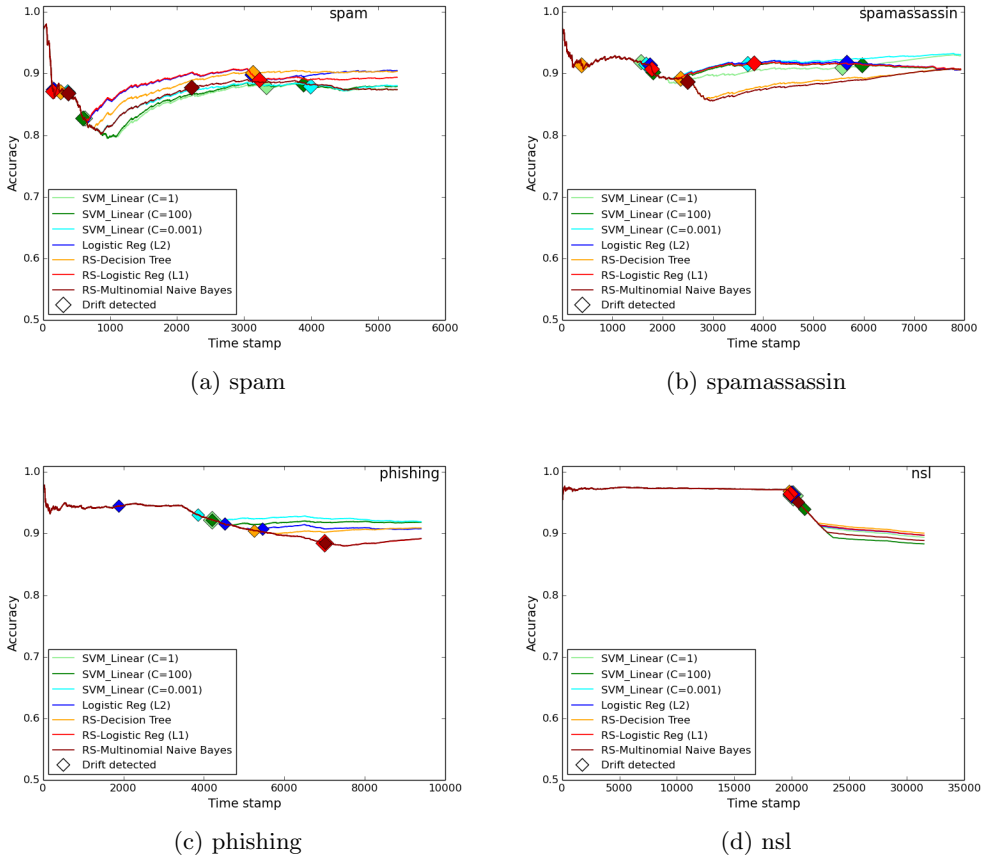
4.4.4 Effects of varying the detection model and the margin width (θ_{margin})

In all the experiments so far, the detection model was taken to be a Linear SVM with regularization parameter of C=1, for the MD3-SVM technique, and a random subspace ensemble with C4.5 Decision Trees as its base models, for the MD3-RS technique. Moreover, the margin width (θ_{margin}), was kept fixed at 0.5, based on intuition. The effects of varying these settings on the detection capabilities of the MD3 framework, are presented in this section.

4.4.4.1 Effects of varying the detection model

The 4 real world datasets from Section 4.4.3 are evaluated here using the following 5 additional detection models: linear SVM with C=100 (high regularization constant), linear SVM with C=0.001 (Low regularization constant), logistic regression with L2-penalty, random subspace ensemble with logistic regression (L1-penalty) models and random subspace ensemble with multinomial naive Bayes models. The results of these experiments are presented in Figure 4.16. All models were evaluated using the scikit-learn machine learning library [149].

Experimental results of varying the detection models are presented in Figure 4.16. It is observed that varying the underlying model has no significant effect on the detection capabilities of the MD3 methodology. A Friedman's non parametric test [186] on the final



(a) spam

(b) spamassassin

(c) phishing

(d) nsl

Figure 4.16: Effect of varying the detection model on the drift detection and the prediction accuracy, over time.

accuracy values for all the 4 datasets, showed that there is no statistically significant difference between the performance of the different models, at a p-value of 0.05. The number of drifts detected and the relative position of the detection are also observed to be close in a majority of the cases.

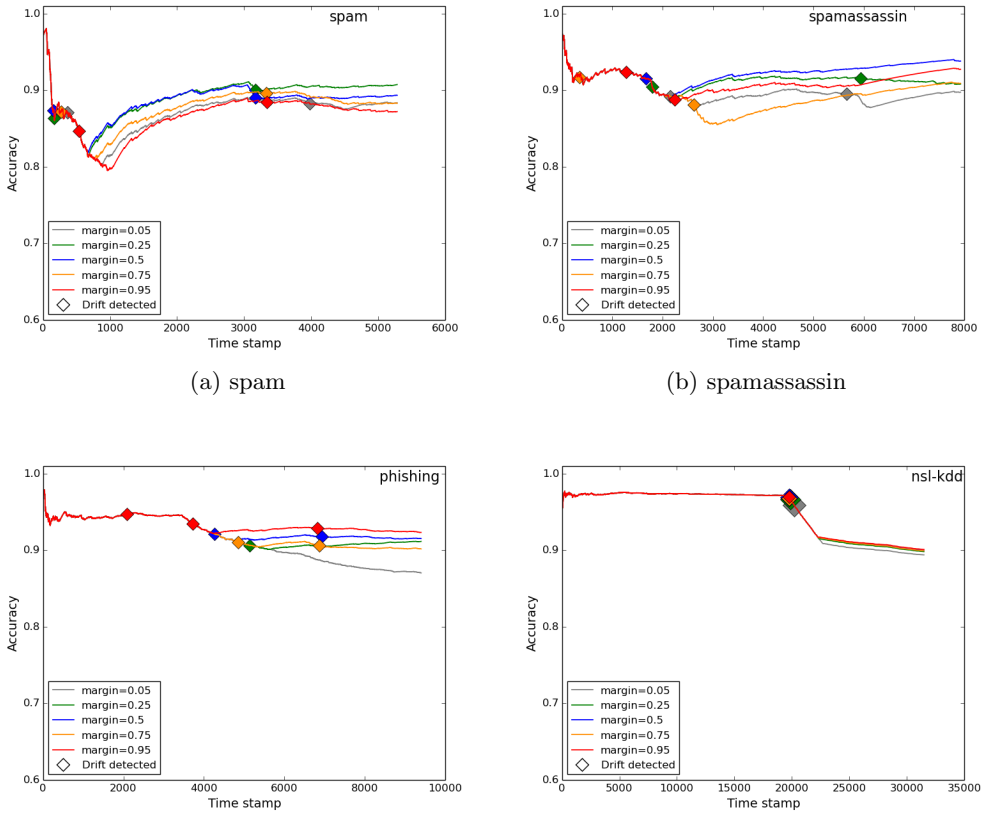
These results are inline with our intuition in Section 4.3.1, where we postulate that the margin density signal could be a general, model independent indicator of change that could be applied to any robust classifier model, which distributes classification importance weights among its features. The models in the experiments described here are all robust classifiers, which distribute feature weights, and as such perform similarly under the MD3 framework. The results of applying the MD3 methodology on a non-robust classifier is shown in Table 4.8, where a logistic regression model with L1-penalty is used. The margin

for a logistic regression model is defined as described in Section 4.3.2.1, with the probability obtained from the posterior estimates of the classifier. This classifier tends to minimize the number of features used in the final models, making it unsuitable for the MD3 methodology. The results of Table 4.8 show that the model does not detect any drifts for the phishing dataset and also performs significantly worse on the spam and the spamassassin dataset. This is because the L1-penalty model tends to minimize the number of features used, which violates the central premise of coupled features detection (Section 4.3.1), that the MD3 model relies on. However, it can be seen from Figure 4.16 that the same L1-penalty based logistic regression model when used with the random subspace ensemble can be effective for usage under the MD3 framework. The MD3 methodology, with its ability to use models of explicit and non-explicit margins, can therefore be applied as a general detection scheme irrespective of the base models used.

4.4.4.2 Effects of varying the margin width (θ_{margin})

In case of the MD3-RS approach, the concept of margin is defined by the margin width parameter θ_{margin}, which was taken as 0.5 in all the experiments thus far. Experiments in the previous section demonstrate that varying the underlying detection model has no significant effect on the detection capabilities of the MD3 framework. In this section, we evaluate the effect of varying the parameter θ_{margin} for the MD3-RS model, which uses C4.5 decision tree models as its base classifier type.

The results of the varying the θ_{margin} are shown in Figure 4.17. It is seen that the choice of the parameter θ_{margin} does not have a significant effect on the final results, as all accuracy plots follow a similar trajectory in time. The only failure case is seen in case of a θ_{margin}=0.05 for the *phishing* dataset (Figure 4.17 c) - grey). At this margin width, the samples captured are insufficient to detect drift effectively. The margin density signal is depicted in Figure 4.18, for the phishing and the nsl-kdd dataset. It is seen that, for the phishing dataset at a θ_{margin}=0.05 the signal fails to detect a drift. For all other margin values, although the absolute signal magnitude is different, they are all effective



(a) spam

(b) spamassassin

(c) phishing

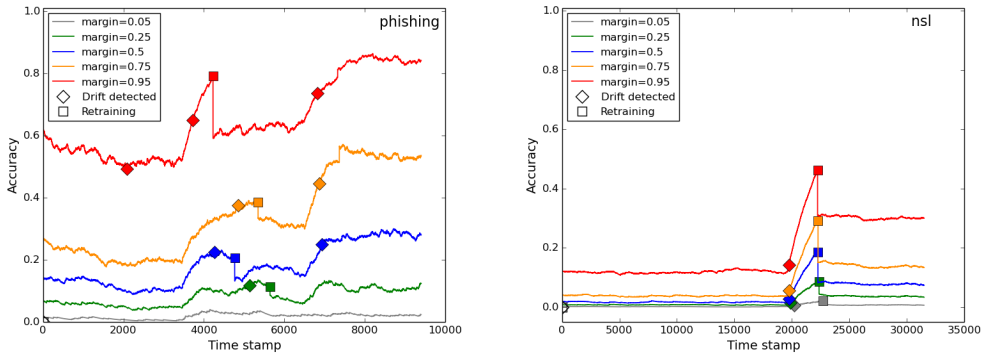
(d) nsl

Figure 4.17: Effects of varying the margin width (θ_{margin}) on the drift detection process of MD3.

in detecting change and do so at nearly the same location. This can be attributed to the reference distribution learning component of the MD3 algorithm, which learns the expected margin density via cross validation on the training dataset. Subsequent changes tracked are relative to the reference distribution, making them effective even when margin width changes. To maintain robustness to change and to ensure that drift detection is effective, a θ_{margin} in the range of 0.25-0.75 is suggested. Further tuning can be done based on the desired sensitivity required for the application.

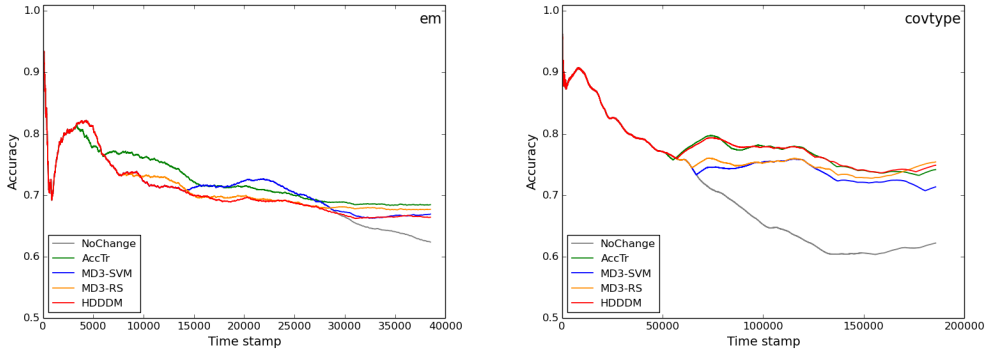
4.4.5 Experimental results on benchmark concept drift datasets

The MD3 methodology is evaluated on two popular real world data streams here - the Electricity Market (EM) dataset and the Covertype dataset (Covtype). These are widely



(a) phishing (b) nsl

Figure 4.18: Margin density metric over time, for different values of (θ_{margin}).



(a) Electricity Market (EM) dataset (b) Covertype dataset

Figure 4.19: Accuracy over time for the NoChange(gray), AccTr(green), MD3-SVM(blue), MD3-RS(orange) and the HDDDM(red) approach, on real world benchmark concept drift datasets.

used datasets for benchmark test of concept drift handling systems [63,162]. The electricity market dataset represents pricing data collected from New South Wales, Australia, which fluctuates based on the supply and the demand components of the market [63, 162]. The Covertype dataset consists of forest cover data and is a multi-class dataset [187]. This dataset was reduced to a binary class prediction problem by considering only the class labels of 1 and 2. Both datasets were pre-processed by converting features to numeric values, normalized in the range of [0,1]. Chunk size N of 2500 was chosen for evaluation of both datasets. These datasets have unknown type and location of the drift, and as such serve as real world benchmarks for evaluating concept drift techniques.

TABLE 4.9

Results on benchmark real world concept drift datasets.

Dataset	Methodology	Accuracy	Drifts signaled	Labeling %
EM (#Instances = 45312, #Features = 8)	NoChange	62.3	0	0
	AccTr	68.4	2	100
	MD3-SVM	66.9	2	13
	MD3-RS	67.7	2	13
	HDDDM	66.4	4	26
Covtype (#Instances = 218515, #Features = 54)	NoChange	62.2	0	0
	AccTr	74.2	16	100
	MD3-SVM	71.3	18	24.2
	MD3-RS	75.4	22	29.6
	HDDDM	74.9	25	33.6

The results of Table 4.9 show that although both datasets have unknown concept drift, they benefit from drift handling, as the accuracy of the *NoChange* approach is worse than that of the other techniques (Figure 4.19). The MD3 approaches were found to have similar accuracy to the fully labeled *AccTr* approach ($\Delta = 1.1\%$ for EM and $\Delta = 0.85\%$ for Covtype). The MD3 approach was also seen to signal fewer drifts than the HDDDM approach, leading to half the labeling budget in case of EM and $3/4^{th}$ in case of the Covtype dataset, for the same resulting accuracy, on average.

4.5 How the MD3 compares to other margin based drift detection techniques?

The ability of the margin density signal to effectively ignore changes to the unlabeled data, which do not affect the classification performance, makes its usage attractive as an unsupervised drift indicator. This is because unlike the feature based change detectors like HDDDM, the margin density (MD) approach implicitly includes the model in the drift detection process. Other unlabeled drift detection techniques developed in literature [107, 173–175] and described in Section 4.2.2.3, also incorporate the notion of a margin. However, these techniques differ from the MD3 approach in the signal being tracked. The MD3 technique tracks the margin density (MD) signal, which is the expected number of samples in the uncertain regions of a classifier. The other margin based techniques of

TABLE 4.10

Characteristics of synthetic data generator used for comparing effects of the different margin based change detection metrics. (Table 4.11).

Before drift (1-500 Samples): $X_{Class1} \sim \mathcal{N}(\mu = \{0.5, ...(5\ features), 0.85, 0.85, ..., 0.85\ (Upto\ 20 features)\}, \sigma = 0.1^2)$ $X_{Class2} \sim \mathcal{N}(\mu = \{0.5, ...\ (5\ features), 0.15, 0.15, ..., 0.15\ (Upto\ 20 features)\}, \sigma = 0.1^2)$
After Drift (501-1000 Samples):   *No Change:* $X_{Class1} \sim \mathcal{N}(\mu = \{0.5, ...(5\ features), 0.85, 0.85, ..., 0.85\ (Upto\ 20 features)\}, \sigma = 0.1^2)$ *Change Upto feature i:* $X_{Class2} \sim \mathcal{N}(\mu = \{0.75, ..., 0.75\ (i\ features), 0.15, ..., 0.15\ (Upto\ 20 features)\}, \sigma = 0.1^2)$

Section 4.2.2.3, essentially track the average change in the uncertainty of samples over time. The difference between the two paradigms is subtle, and the ability to specify a fixed margin, before deployment, is responsible for the efficacy of the margin density approach. To elucidate the difference between these paradigms and their implications, a synthetic experiment is designed in this section.

In order to understand the effects of the different margin based techniques, an experiment similar to that in Section 4.3.2.2 is performed. A synthetic 20 dimensional dataset is generated. The dataset has 1000 samples, with concept drift occurring at the midpoint (500 samples). The characteristics of the data are shown in Table 4.10. The dataset has 20 dimensions, the first 5 of which are made irrelevant to the classification task, by assigning them the same distributions for the two classes. After the midpoint, drift is induced in the dataset by changing Class 2's feature distribution. The feature distributions are incrementally changed upto feature i, by changing their mean values. The remaining 15 features (feature 6-20) are all relevant to the classification task and the effect of changing them gradually is analyzed via this experiment. The effects of these changes are evaluated by training a model on the first 500 samples and evaluating change metrics on the remaining 500 samples after the drift. We chose the random subspace ensemble with decision trees as its base models (MD3-RS), for the experiments here.

TABLE 4.11

Results of change detection metrics ΔErr, ΔMD, ΔUncertain and ΔHD, on varying intensities of drift on synthetic data. Features 1-5 are irrelevant to the classification. Bold entries represent first indication of change, for each of the metrics.

# of features affected (i)	ΔErr	ΔMD	ΔUncertain	ΔHD
1 (Irrelevant)	0	0	0	**0.13**
3 (Irrelevant)	0	0	0	0.17
5 (Irrelevant)	0	0	0	0.22
6	0	0	**0.05**	0.23
7	0	0	0.2	0.26
8	0	0	0.2	0.28
9	0	0	0.25	0.3
10	0	0	0.25	0.33
11	0	**0.5**	0.4	0.35
12	**0.5**	0.5	0.45	0.37
13	0.5	0.5	0.45	0.39
14	0.5	0.5	0.35	0.41
15	0.5	0.5	0.4	0.43

The effects of changing the feature values of the synthetic 20-Dimensional dataset, is presented in Table 4.11. The change in the training and testing error (ΔErr), the margin density metric (ΔMD), the average uncertainty (ΔUncertain) and the Hellinger Distance (ΔHD), are compared. Changing feature 1-5 have no significant effect on the classification error, as these features are irrelevant from a classifier's perspective. The HD metric, owing to its model agnostic measurements, is unable to see this distinction and as such a change is seen for the ΔHD metric in Table 4.11. Both the margin density technique and the uncertainty tracking techniques, are robust against such changes. The advantage of the MD technique, over the traditional uncertainty tracking techniques, is seen in case of the changes in the relevant features (6-20). A robust classifier, such as the random subspace ensemble, is inherently capable of providing high predictive performance, even if a few of the relevant features drift. Unless a majority of the features change at the same time, the robust classifier is unaffected by changes to a few features. The traditional uncertainty

tracking techniques fail to distinguish between critical changes and changes which can be inherently managed by a robust classifier. By tracking the posterior probability estimates of the classifier models, the uncertainty based techniques could effectively ignore changes to irrelevant data (similar to the MD3 approach), but they still lead to additional false alarms when compared to the margin density approach. The MD approach limits the critical area of uncertainty being monitored and as such it limits the tracking to the most critical drifts only. This is seen in Table 4.11, where the MD metric is affected after 11 features drift simultaneously, as opposed to 6 features for the *Uncertainty* tracking techniques. The actual fully labeled technique would signal drift after 12 features are affected. The MD signal comes closest to tracking the actual drift using only unlabeled data, with a high robustness to false alarms. The margin density approach is robust not only to changes in the irrelevant features, but also to changes in the relevant features, which are not critical to a robust classifier's performance.

The *Uncertain* and the *HD* metrics, presented here, are representations of two paradigms of unlabeled drift detection techniques in literature. The *HD* metric represents the fundamental behavior of the feature distribution tracking techniques (Section 4.2.2.2) [64,170–172], while the *Uncertainty* metric represents behavior of the model based posterior probabilities tracking techniques (Section 4.2.2.3) [107,173–175] . Although the actual usage and the implementation of the metrics are nuanced and different in literature, our purpose here is to demonstrate the underlying principle that makes the margin density signal more robust to false alarms. The purpose of this section is to provide motivation for future work on improving reliability of unlabeled drift detection techniques. The generation and analysis of real world datasets, with similar characteristics as the synthetic data presented here, could also be a useful contribution to better enable further research in this area.

4.6 Chapter summary

In this chapter, the Margin Density Drift Detection (MD3) methodology was proposed, to detect drifts from unlabeled data. The proposed methodology uses the number of

samples in a classifier's region of uncertainty, as a metric for detecting drift. A significant deviation in the margin density metric, was used to signal the need for labeled samples for subsequent retraining of the classifier. MD3 was shown to be independent of the classification algorithm used, robust to stray changes in data distribution and a good substitute to a labeled drift detection scheme. Experimental analysis was performed on 6 drift induced datasets, 4 real world concept drift datasets from the cybersecurity domain and on 2 benchmark concept drift datasets. The results indicated high detection and prediction performance, coupled with low false alarm rate for the MD3 approach. When compared with a state of the art unlabeled feature tracking approach - The Hellinger Distance Drift Detection Methodology (HDDDM), the MD3 algorithm resulted in a fewer labeling percentage, for the same (and sometimes higher) accuracy. The claims made at the start of the chapter are summarized below:

- ***Claim a:*** *The MD3 approach can be used as a substitute for fully labeled approaches to detect drifts, without significant reduction in the predictive accuracy over the stream.*
  The margin density approach, when tested on 6 drift induced datasets, resulted in a <1% difference in the accuracy, on average, between the fully labeled drift detection methodology-AccTr. When tested on the 6 real world datasets, the average difference was 1.1%. Additionally, the plotted accuracies over time, followed a similar trajectory for the AccTr and the MD3 approaches. The ability of the MD3 approach to detect drift when it was detected by AccTr, with minimal lag, makes it a suitable surrogate to be used in lieu of the labeled approach.

- ***Claim b:*** *The MD3 approach can reduce the number of false alarms compared to other traditional change tracking unlabeled methodologies.*
  The margin density approach was compared to the HDDDM approach, which tracks changes to the unlabeled data feature values to detect distribution changes. A drift induced experiment, which ranks features by their importance and then systematically introduces drift in the bottom 25% of the features, was used to generate changes which do not cause significant performance degradation. While the HDDDM ap-

proach signals these changes as potential drifts, the AccTr labeled approach and the MD3 approach, were both resistant to these changes and effectively ignore it. When compared on the real world cybersecurity datasets, the HDDDM approach causes four times as many false alarms as the MD3 approach, on average.

- ***Claim c:*** *The MD3 approach can perform computations incrementally in a streaming environment and can be used independently of the type of classifier used and the application domain.*

  The margin density metric was computed incrementally in the MD3 algorithm, by making use of a moving average formulation of the metric along with a forgetting factor λ. It was shown that the metric computations can be made independent of the online classification technique used. Experiments were performed in a streaming fashion, with samples appearing one at a time and limited memory and time constraints are imposed. The MD3 model was implemented using a SVM model and a random subspace ensemble of decision trees. Both models perform identical in terms of number of drifts detected and false alarms detected across all experiments. The average accuracy of these two methods showed no significant difference. As such the MD3 approach was found to be model independent. Margin density can be computed explicitly from probabilistic classifiers and for classifiers which do not have a notion of margin (e.g., Decision Trees), they could be placed in the random subspace ensemble setting. Experimentation performed by varying the underlying model of the detection, indicted that as long as the model is robust, distributes feature weights among the informative features, it can provide the benefits of margin density based detection.

- ***Claim d:*** *The robustness of MD3 leads to less labeling effort needed in the adaptation process.*

  Every drift signaled, requires labeling of samples to confirm if it leads to accuracy degradation and to retrain the classifier, in the event if drift is confirmed. By being robust towards false alarms, the MD3 approach used, on average, 8.3% less labeling than the HDDDM approach over the 4 real world cybersecurity concept drift

datasets. On the benchmark datasets, an average reduction of 9.85% in labeling is seen, between the HDDDM and the MD3 approaches. Additional proof of concept experiments, comparing MD3 with other uncertainty based drift detection approaches, indicated that margin density provides a better way to account for innate robustness of classifiers, making it more reliable in detecting significant drifts.

The MD3 algorithm showed to be an efficient, robust and practical way, of detecting model degradation in a streaming data environment, using unlabeled data. Experiments on concept drifting datasets from the cybersecurity domain indicate its capability to detect concept drift, when changes can be of an adversarial nature. However, the specific nature of the adversarial activity in these cybersecurity datasets is not known. Ability of using the ideas learned from the margin density approach and applying them to detect targeted attacks against a classifier, is an area which requires further investigation. Another direction of future work would be to use the margin density information along with active learning strategies, to selectively label samples for the confirmation and retraining phase. This would allow for further savings in terms of reduced labeling, in the online classification process.

CHAPTER 5

IMPACT OF DEFENDER'S CLASSIFIER DESIGN ON ADVERSARIAL CAPABILITIES IN *DYNAMIC - ADVERSARIAL* ENVIRONMENTS

Classifiers deployed in adversarial domains, are susceptible to evasion at test time. Chapter 3 demonstrated the nature of these attacks and established the adversarial model for testing security measures. In Chapter 4, a reliable unsupervised drift detection methodology was developed for detecting and reacting to changes in distribution, over time. However, this methodology did not consider the specific nature of adversarial changes, as it assumes that drift detection would be sufficient to deal with adversarial test time activity. Adversarial changes in data distribution are dependent on the deployed classifier, which the attacker is trying to evade. In this chapter, the relation between the defender's model design and the capabilities of an adversary at test time, is analyzed. This chapter evaluates the specific nature of the adversarial domain, which makes it different from other distribution changes. This evaluation is aimed to motivate the development of adversarial aware design of classifiers in dynamic domains.

5.1 The dynamics of securing against data driven exploratory attacks

In Chapter 3, data driven exploratory attacks on black box classifiers were demonstrated. It was seen that classifiers after deployment are vulnerable to probing based attacks, where an attacker learns the behavior of the system and then crafts samples, to evade it. These attacks lead to degradation in the prediction capabilities of the deployed model, as the attacks cause a drift in the data distribution. For a classification system to be applicable in a *Dynamic - Adversarial* environment, it needs to account for the adversarial capabilities at test time, and also needs to focus on the ability to recover from attacks, for continued

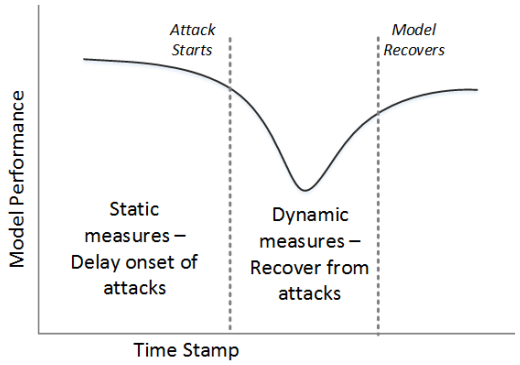


Figure 5.1: Illustration of model performance over time, indicating onset of attack and recovery from it. Static measures of security focus on delaying onset of attacks. Dynamic measure focus on detection and recover only.

operation in a dynamic environment.

Most works in adversarial machine learning, concentrate on making classifiers harder to evade. They do so by resorting to *Complex learning* strategies [2, 4, 40, 71–73, 87], which advocates integrating maximum informative features into the classifier models, or by integrating *Randomness* into the prediction process [40, 66, 74, 76, 91, 98]. The emphasis of these two strategies, is to make attacks harder/expensive to carry out. However, these methodologies approach the problem of security from a static perspective. They focus on the ability to ward of attacks, but fail to provide insights or directions regarding measures to be taken after an attack commences. This is illustrated in Figure 5.1, where the predictive performance of a classifier faced with an attack, and subsequent recovery, is shown. Static measures of *Complex learning* and *Randomness*, concentrate only on the portion of the attack-defense cycle before the attack starts. Any practical and usable system also needs reactive dynamic measures, which can deal with attacks after it has affected the system's performance.

Dynamic approaches are developed in the domain of concept drift research, where the aim is to detect and adapt to changes in the data distribution over time. One such methodology for reliable drift handling was developed in Chapter 4, as the Margin Density Drift Detection framework (MD3). However, this approach, like the other works on concept drift detection [58–60, 63], considers an adversarial agnostic view of the system. Concept

drift detection considers any change in data equally, without regard into the domain specific nature of the change. This makes it ineffective in an adversarial setting, where drifts are a result of attacks, which are in turn a function of the model deployed by the defender. As such, the ability to control drifts by adjusting the strategies of the defender, are an important strategic advantage disregarded by concept drift research. This chapter brings together the concepts of Adversarial data mining and Dynamic data mining, under a common paradigm of *Dynamic Adversarial Mining*. The aim is to look at the never ending nature of the attack-defense cycle, where myopic thinking can only help win battles, but not the war. Impact of classifier design on the detect-ability of attacks and the overall evasion resistance is analyzed.

The central theme of this chapter is to analyze the impact of different classifier design strategies, on the adversarial capabilities at test time. As an illustrative example consider the 2D synthetic data in Figure 5.2. A classifier model C is trained on the space of *Legitimate* class training samples (blue). The model C is seen to be restrictive, as it allows only a narrow margin of samples to be admitted into the system as benign samples. This strategy is considered to provide better security from a static perspective, as it would require an adversary to craft samples in a small and exact range of feature values. However, looking at the problem from a dynamic perspective, it is seen that the space of possible attacks is now highly overlapped with the training data. As such, once an attack starts, the adversary has enough information about the feature space, so as to become indistinguishable from benign traffic entering the system. These attacks are harder to detect (using unsupervised techniques), and also harder to recover from, due to the inseparability of samples. On the other hand, the simple design of Figure 5.3 a) (simple as it reduces the number of features used in the model C), has a larger space of possible attacks, making it easier to evade. However, the increased space means that an adversary evading C will not be able to completely reverse engineer the location and characteristics of the *Legitimate* training samples, due to the added uncertainty provided by the large space of possible attacks.

The illustrative example of Figure 5.2 and Figure 5.3, demonstrate the need to de-

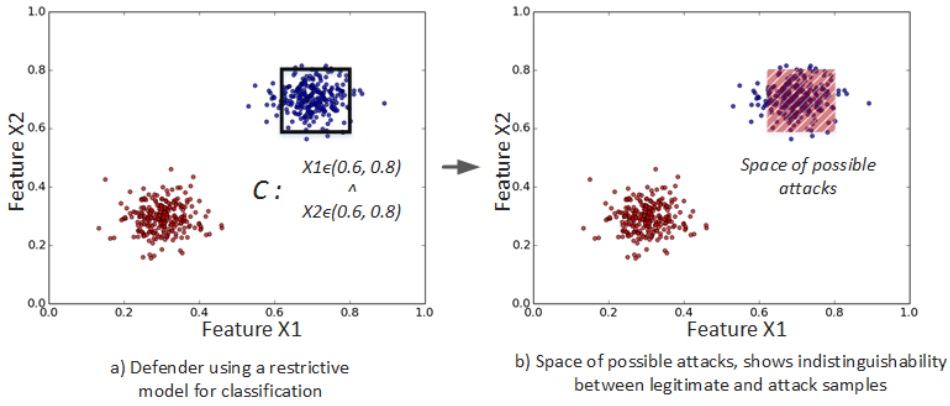


Figure 5.2: Impact of using a restrictive defender model. Attacks are harder to carry out, but will lead to inseparability from benign traffic.

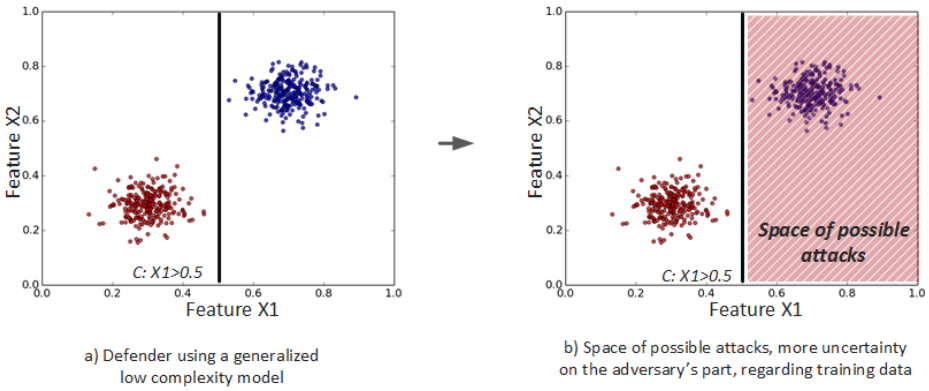


Figure 5.3: Impact of using a simple defender model. The space of attacks causes uncertainty regarding exact features of benign data.

velop countermeasures from a *Dynamic - Adversarial* perspective. In such an environment, delaying the onset of attacks, detecting attacks and recovering from them are all equally important, for the continued usage of the classification system. In this chapter, the impact of classifier design strategies on the severity of attacks launched at test time, is presented. The existing ideas of *Randomness* and *Complex learning*, for securing classifiers, are evaluated from a dynamic perspective. Also, the ability to take preemptive measures, to ensure defender's leverage in a dynamic environment, is evaluated. In particular, the ability to hide feature importance is demonstrated, as a proactive measure to enable efficient reactive security.

The following claims are evaluated in this chapter: a) Restrictive one-class classifiers,

Complex Learning and Randomization, are not effective as long term security measures, b) Unsupervised Drift Detection (particularly MD3), can be actively evaded by an adversary at test time and c) Using a reduced feature set in the defender's model can improve the detectability of attacks and prevent information leakage to the adversary. The analysis in this chapter is based on intuitive representative methodologies, which demonstrate the inherent vulnerabilities of different classification based systems, as they often tend to be evaluated only on a strict set of predefined objectives. In a real world environment, attacks can take any shape and can almost always circumvent defenses, over time. In such environments, staying ahead of the attacker at every step is more crucial than hoping for an iron clad design [49]. By understanding the nature of probing based attacks and the requirements of a dynamic system, the proposed solution aims to provide better long term security in *Dynamic - Adversarial* environments.

The rest of this chapter is organized as follows: Section 5.2 presents related work in the area of countermeasures against exploratory attacks. Section 5.3 presents the idea of adversarial uncertainty, as a novel way of analyzing the impact of classifier design strategies in dynamic environments. Impact of classifier design on adversarial capabilities is also discussed in this section. Section 5.4 experimentally evaluates vulnerabilities of Randomness, Complex Learning and One-class classifiers, from the perspective of providing long term security. The proposed hidden classifier design is presented and evaluated in Section 5.5. Chapter summary is presented in Section 5.6.

5.2 Background work on the security of machine learning against exploratory attacks

The vulnerability of machine learning to adversarial activity, has received considerable attention in recent times [40, 66, 100, 137]. Countermeasures have been developed to thwart attacks or to make them otherwise difficult to carryout, by training resilient models. Most work on securing classifier have focused on targeted-exploratory attacks, where an adversary starts with an attack payload, which it wants to optimally modify, so as to

evade the defender's model [72, 73, 87]. Attack cost is given by the distance from the initial adversarial samples, to the samples which first cause evasion, commonly measured using L1-Euclidean norm for numeric data spaces. Under this framework, countermeasures have been developed, evaluated and compared. The methodologies can be broadly divided into two classes: a) *Obfuscation* approaches, which rely on randomness to mislead an adversary [2, 4, 40, 71–73, 87], and b) *Complex learning* approaches, which rely on integrating orthogonal information into the classifier model [40, 66, 74, 76, 91, 98], to make it harder to mimic.

Obfuscation relies on randomization to make attacks harder. By training multiple models over the hypothesis space of the training data, several alternate decision strategies are made available at test time [72, 76, 91, 98]. Randomly choosing among these strategies have shown to increase the cost of adversarial evasion [76]. An example of such an approach is a multi-modal biometric system [136] which alternates between fingerprint recognition and voice recognition to provide authentication services. As such, an adversary will be confused about the influence of a particular biometric, on the final outcome of the system, making evasion by mimicking, harder to achieve. Randomization was found to be effective in a multiple classifier system (MCS) setting to make evasion harder [98], and was also shown to be effective against reverse engineering attacks on Support Vector Machines [6]. In [91], randomization of the class boundary was used to demonstrate increased difficulty of evading the *SpamAssassin* spam filtering system. Providing misleading feedback on input samples was proposed in [76], as an effective measure against targeted evasion attacks. However, in the same work, randomization was found to be ineffective against attacks of indiscriminate nature, where any samples evading the classifier is considered a valid attack payload. The reason for this shortcoming is due to the nature of randomization based security, where all information is available to be probed, but information needs additional validation (more probes) to determine its veracity.

Complex learning methods relies on increasing the robustness of classifiers, by evenly distributing importance weights among the informative features of the model. Considering

the earlier multi-modal biometric example, a system which uses both fingerprint and voice recognition is harder to evade, as an adversary needs to successfully evade both sets of features, to gain access. Feature bagging was proposed as a way to increase robustness in a MCS setting in [72]. In [75], modifying the min-max formulation of the loss function was proposed, to ensure robustness in a game theoretic formulation of the attack-defense problem. An adversarial aware SVM model was proposed in [90], by adding additional terms to the loss function of the model. In all these approaches, it is postulated that distributing feature weights will require the adversary to mimic a large number of features, in order to gain access to the system, thereby making it secure. This is especially true for the case of targeted evasion attacks, where the adversarial cost is determined by the number of features which need mimicking, to cause an evasion [51]. Robustness provides security against targeted evasion attacks, against feature deletion attacks and against random perturbation to existing attack samples [71, 72]. In [188], the idea of defensive distillation and gradient masking was proposed, to make deep neural networks robust against minor perturbations to attack samples. In [138], the idea of adversarial training is proposed for securing against large perturbation to attack samples, by preemptively training networks with adversarial samples, to ensure robustness at test time. The underlying notion in all these techniques is that the cost of changing samples by an adversary, given by the number of features to be mimicked, is increased. However, the effectiveness of these countermeasures against indiscriminate and black box attacks on classifiers, is not evaluated. An attacker free to move in the feature space is not concerned with the number of features which need modifying. As such, the idea of robustness and the security it promises needs scrutiny in such a setting.

The above mentioned techniques of *Complex learning* and that of *Obfuscation*, rely on a static assumption of the problem of security. It is expected that security is responsible only for delaying attacks, or for making it harder for the system to get attacked by an adversary. These countermeasures do not discuss the impact of an attack or possible recovery mechanisms, after a system is attacked. These works themselves demonstrate that increasing adversarial effort/budget will eventually lead to an attack. Since no system is

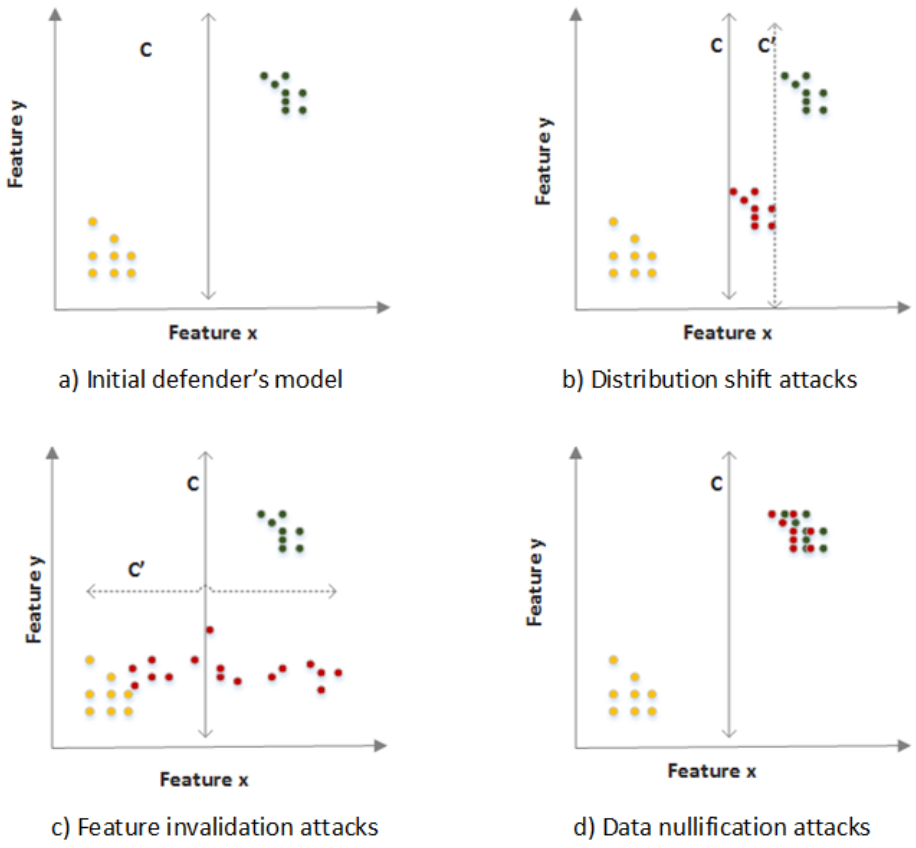


Figure 5.4: Illustration of exploratory attacks on defender's classifier C. *Green*- Legitimate training samples, *yellow*- Malicious training samples, *red*- Adversarial attack samples at test time.

truly secure, these countermeasures only discuss one aspect of the attack, which is the situation before the onset of attacks. It is important to consider the dynamics of the system, to truly understand its vulnerabilities and the effectiveness of proposed security measures. The dynamic aspects of security are illustrated in Figure 5.1, where static measures are shown to focus on preventing onset of attacks, while dynamic measures are seen to focus on detection and recovery from attacks.

Ideas and motivation regarding the dynamics of the machine learning security, were proposed in [114]. It was proposed that for a system to be of any practical use, it should evolve over time and engage with human experts beyond feature engineering and labeling. A new taxonomy of exploratory attacks was presented in [114], based on their effects on a dynamic data mining process. The attacks are illustrated in Figure 5.4, and are classified as:

a) Distribution Shift attacks, b) Feature Invalidation attacks and the c) Data Nullification attacks. Distribution shift attacks are a results of attacks where the samples move to a new region of space to avoid detection. These can be fixed by collecting the new attack samples and retraining the original model using the already defined set of features. These attacks are illustrated in Figure 5.4b), where the model C' represents the newly trained model, in response to attacks. Feature Invalidation attacks are a result of indiscriminate attacks on a subset of the features, in a denial of service (DOS) attack fashion, causing the feature to be unusable for future prediction tasks. This is illustrated in Figure 5.4c), where the attacker launches indiscriminate attack on feature x, making it unusable for the prediction task. However, these attacks can be fixed by retraining models on the other set of informative orthogonal features (here, feature y), collected from the data. The Data Nullification attacks (Figure 5.4d)) are the most severe category of attack, as they leave the training data completely useless, due to overlapping attack and legitimate samples. The classifier cannot be retrained using the existing set of features, and new analysis and data collection is needed to redesign the system, which requires tedious examination of samples, to detect attacks and generate new set of features. From a dynamic machine learning perspective, the task of the defender should be to ensure that such type of attacks are avoided, even though evasion might be made slightly easier. Static methods of security, as discussed earlier the obfuscation and the complex learning strategies, increase vulnerability to data nullification attacks, in their pursuit to make evasion harder.

Data nullification attacks are a result of an attacker gaining excessive information about the training data used by the defender. The adversary obtains this information via probing the black box model of the defender's model. In case a defender uses all the informative features from the training data into the deployed model, as is done in the case of complex learning and randomization approaches, the adversary can reverse engineer all this information via probing. An overly informative model leaks information to the adversary, making the defender vulnerable to adversaries equipped with increased information about the prediction space. This information leverage which the adversary

can obtain is especially harmful due to the following reasons: a) The attacker can launch data nullification attacks, which can make them indistinguishable from the benign samples, b) Detecting and retraining after such attacks is difficult and labor intensive, c) Data nullification can lead to loss of privacy, as the adversary can approximate the space of *Legitimate* data samples. The loss of privacy was discussed in [100], where experiments on ML-as-a-service providers of Amazon AWS[1] and the Google Cloud Platform[2], demonstrated that overfitting leads to data leakage. Overfitting, due to increased complexity of defender's model, leads the attacker to the training space of *Legitimate* class samples. This causes the models to leak sensitive information about the internal state of the training data. It was observed that models which sweep larger generalization areas are more resistant to data leakages. This was in direct contrast to the robustness strategies advocated by existing works on security of ML. Recent works in [40], advocate the use of complex hypothesis class in deploying neural networks and that of adversarial training for defense against large perturbations, and were also shown to be vulnerable to privacy losses in [47].

The idea of data leakage and data nullification attacks, present a new dimension in the evaluation of machine learning security, beyond the traditional established metrics of hardness of evasion [51]. These ideas are a result of a more widespread and practical understanding of the vulnerabilities of classifiers, resulting from black box and indiscriminate attacks at test time. These attacks are harder, if not impossible, to completely account for at the training time, and have to be dealt with in a reactive manner. Existing work on concept drift approaches this problem as a cyclic one, where data distribution changes are detected and handled over time [58]. However, concept drift research has hitherto taken a domain agnostic approach to dealing with distribution shifts. Adversarial characteristics of the drift and preemptive strategies to mitigate and manage drift, have not been analyzed or discussed before, to the best of our knowledge. The only mention we found was in [1, 114], where adversarial drift was understood to be a special type of drift, being a function of the deployed classifier itself. Since the attacks are targeted towards evading the deployed classi-

[1] `https://aws.amazon.com/machine-learning/`

[2] `cloud.google.com/machine-learning`

fier, the choice of the deployed classifier directs the gamut of possible attack on the system, to a large extent. We analyze this specific aspect of the attacks, and design a preemptive strategy which benefits the dynamic handling of the attacks at test time. By directing the adversary to regions of space of easy detect-ability and defense, the defender can control drift and stay ahead at every step. This chapter analyzes one such solution, based on the notion of feature importance hiding. This intuition goes against the established ideas of robustness, which claims that the defender should incorporate maximum information in the deployed model. The effects of the different classifier designs, on adversarial activity at text time, are analyzed and the *Predict-Detect* classifier framework is proposed as a viable design for usage in a *Dynamic Adversarial Environment*.

5.3 Adversarial Uncertainty - On the ability to effectively react to attacks in dynamic adversarial environments

Deployed classifiers are vulnerable to adversarial perturbations at test time, which cause the models to degrade over time. Operating in a dynamic environment requires a forward thinking and adversarial aware design for classifiers, beyond fitting the model to the training data. In such scenarios, it is necessary to make classifiers - a) harder to evade, b) easier to detect changes in the data distribution over time, and c) be able to retrain and recover from model degradation. While most works in the security of machine learning has concentrated on the evasion resistance (a) problem, there is little work in the areas of reacting to attacks (b and c). Although streaming data research concentrates on the ability to react to changes to the data distribution, they often take an adversarial agnostic view of the problem. This makes them vulnerable to adversarial activity, which is aimed towards evading the concept drift detection mechanism itself. Understanding the impact of attacks and defense in a cyclic environment, where a system is attacked and needs to recover from it periodically, requires a new viewpoint on the problem and the metrics used for evaluating the different design strategies. The metric of attack evasion rate [51] is sufficient for evaluating robustness of models to the onset of exploratory attacks, but it does not provide intuition

about the effectiveness of strategies which are designed for reacting to attacks in a dynamic environment. In this case, we need a metric which can convey information about the ability to react to attacks and continue operations, following the onset of attacks.

In this section, the idea of *Adversarial Uncertainty* is introduced, to understand the impact of different classifier design strategies on the ability to react to attacks, once they start to impact a system's performance. The motivation behind adversarial uncertainty is presented in Section 5.3.1. The impact of the defender's classifier design on adversarial uncertainty, is presented using an illustrative example of binary feature valued data in Section 5.3.2. A heuristic method to measure adversarial uncertainty is presented in Section 5.3.3, by means of introducing the *Adversarial Margin Density* (AMD) measure. The ability of adversaries to utilize the available probe-able information, from the defender's black box model, in order to launch high confidence attacks, is demonstrated in Section 5.3.4 and Section 5.3.4.1. This is done to analyze the impact of a determined adversary, and to be able to design secure classifiers to counter them.

5.3.1 Motivation

The defender extracts information about the predictive problem, by analyzing and learning from the training data. The adversary on the other hand, obtains its information by probing the deployed black box model of the defender. As such, there is a gap between the information held by the two players. This is not a significant concern when the adversary is aiming to only evade the deployed model, as complete understanding of the feature space is often not needed to evade most robust classifier models. However, this information deficit/gap is necessary to evaluate the impact an adversary can have on the reactive capabilities of a defender. Adversarial uncertainty refers to the uncertainty on the part of the adversary, due to the unavailability of the original training data.

To understand the impact of adversarial uncertainty, consider the following toy example: a 2-dimensional binary dataset, where the sample *L(X1=1, X2=1)* represents the *Legitimate* class training sample and *M(X1=0, X2=0)* represents the *Malicious* class sam-

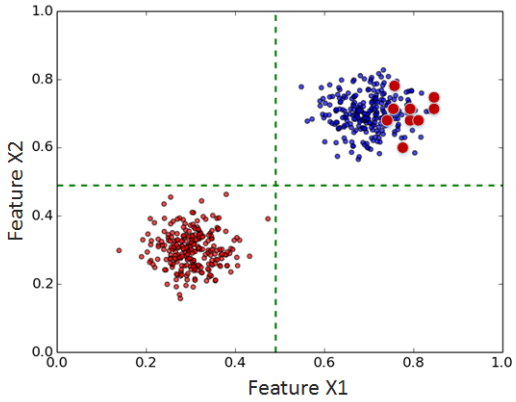


Figure 5.5: Illustration of data nullification attacks on the space of legitimate samples (blue).

ples. A model trained as $C : X1 \vee X2$ provides generalization capabilities, by allowing (0,1) and (1,0) to be considered as legitimate samples at test time. In this case, an adversary looking to evade C, can pick one of the following three attack samples at test time: (1,0), (0,1), (1,1). While any of these samples will lead to a successful evasion on the adversary's part, only one is truly devastating for a reactive system. The adversarial sample (1,1) will make the defender's model ineffective, as it completely mimics the training sample L. However, in this case, the probability of selecting this sample is 1/3, as an adversary is not certain about the exact impact of the features $X1$ and $X2$, on C. This uncertainty on the part of the adversary is referred to as adversarial uncertainty. In this example, adversarial uncertainty will enable the defender to recover from attacks 2/3 times, as the attack sample (1,0) can be thwarted by an updated model $C' : X2$, and the sample (0,1) can be thwarted by the model $C' : X1$.

The attack sample (1,1) is an example of a data nullification attack [114]. Data nullification leaves the defender unable to use the same training data, to continue operating in a dynamic environment. An illustration of data nullification attacks on numerical data spaces is shown in Figure 5.5. In this figure, the attacks samples generated, overlap with the original space of legitimate data samples. As such, the defender will be unable to train a high quality classification model, which can effectively separate the two class of samples,

while using the existing set of features. Recovering from data nullification attacks will require the defender to generate additional features and collect new training data, both of which are time taking and expensive operations. Also, such attacks will be harder to detect via unsupervised techniques (such as the margin density drift detection (MD3) methodology of Chapter 4), because the legitimate and the attack samples have a high degree of similarity and proximity, in the feature space. It is therefore required to ward of data nullification attacks, to be able to effectively detect and recover from attacks. Adversarial uncertainty provides a rough indication of the inability of the adversary, to deduce with confidence the exact impact of the various features on the prediction problem. A high adversarial uncertainty will lead to a lower probability of a data nullification attack.

From a streaming data perspective, the ability to detect attacks and being able to retrain the classifier, are important characteristics of the system. This is possible only if the original legitimate training data is not corrupted by an adversary at test time (i.e., data nullification attacks are prevented). Data nullification attacks are possible if an adversary is able to simultaneously and successfully reverse engineer the impact of the entire feature set, on the defender's classification model. This is a result of an adversary's confidence in the impact of the different features on the prediction task, which it obtains via exploration of the deployed black box model. A highly confident adversary can not only evade the deployed classifier, but can also avoid detection by unsupervised methodologies. Thus, it is essential to evaluate the impact of the various defender strategies, on their ability to ensure that they do not leak excessive information to an adversary (i.e., their ability to ensure high adversarial uncertainty).

5.3.2 Impact of classifier design on adversarial uncertainty

Adversarial activity and capabilities are a function of the deployed black box classifier, which is being evaded. The adversary's perception of the prediction space is directed by the feedback on probes submitted to the defender's model. As such, the defender's classifier design has a certain degree of control in defining the range of attacks, that can be launched

TABLE 5.1

Impact of classifier design on evasion probability and adversarial certainty.

Classifier Model (C)	Model representation	Evasion probability	Adversarial Certainty
Simple model (e.g., C4.5 Decision Tree)	$X_i = \quad 1$	$2^{N-1}/2^N = \quad 1/2$	$1/2^{N-1}$
Complex model - One class Legitimate	$\bigwedge_{i \in N} X_i = \quad 1$	$1/2^N$	1/1=1
Complex model - One class Malicious	$\bigvee_{i \in N} X_i = \quad 1$	$(2^N - 1)/2^N$	$1/(2^N - 1)$
Complex model - Feature bagged robust model (e.g., random subspace majority voted ensemble)	$\bigvee_{s \subset N} \bigwedge_{i \in s} X_i = \quad 1$	$\sum_{i=\frac{N}{2}+1}^{N} \binom{N}{i}/2^N = \quad 1/2^*$	$1/2^{N-1}$
Randomized model - Feature bagged robust model with majority voting and random selection	$\bigwedge_{i \in s} X_i = \quad 1; \quad s \subset N$	$1/2^{N/2}$	$1/2^{(N/2)}$

$^* \sum_{i=\frac{N}{2}+1}^{N} \binom{N}{i} = \quad \sum_{i=0}^{N} \binom{N}{i}/2, \quad if \quad N \quad is \quad even = 2^N/2 = 2^{N-1}$

against it. Classifier design strategies range from restrictive one class classifiers [189, 190], to robust majority voted classifier [83, 84], and randomization based feature bagged ensembles [91, 98]. From an adversarial perspective, the selection of classifier learning strategies can have a significant impact on adversarial uncertainty and on the hardness of evasion. The impact of popular classifier design strategies, based on research directions in security of machine learning, is illustrated here using an example of a N-dimensional binary feature space. We consider the training dataset to be comprised of one *Legitimate* class sample *L(1,1,...(N features),1)* and one *Malicious* class sample *M(0,0, ...(N features),0).* As such, the features 1 to N are all informative in discriminating between the two classes of samples. The effective usage of this orthogonal information, leads to the various design strategies, as shown in Table 5.1. Since this is meant to be an illustrative example, we consider the impact of these designs against a random probing based attack strategy, where the attacker tries different permutations of the N binary variables, to understand the behavior of the black box classifier.

The *Simple model* strategy of Table 5.1, represents a classifier design which emphasizes feature reduction in its learning phase. An example of such a classifier is a C4.5 Decision tree or a Linear SVM with L1-regularization penalty, which gravitate towards sim-

pler model representations. A representative model in this case, is given by $C : X_i = 1$, as the learned classification rule. The probability that a randomly generated N-dimensional probe sample will evade C is 0.5, as the prediction space is divided into two halves, on this feature X_i. The adversarial certainty (Table 5.1) refers to the confidence that an adversary has about the defender's training data, given that an attack sample is successful in evading C. In the case of the simple model, the adversary is fairly uncertain, as the training data sample L could be any one of the 2^{N-1} probe samples, which provide successful evasion.

The *Complex models* rely on aggregating feature information, to make the learned models more complex (i.e., have more coefficients and important features). The one class classifier strategy is a complex model and comes in two flavors: a restrictive boundary around the legitimate training data samples, and a restrictive boundary around the malicious training data samples. In the former case, the model is the hardest to evade, as an adversary will need to simultaneously evade all N features, to gain access to the system. This provides additional security from attack onset, by reducing evasion probability to 1 in 2^N. However, it also leads to an adversarial certainty of 100%, implying that any successful attack will lead to a data nullification and subsequent inability of the defender to recover from such attacks. Thus, as opposed to common belief in cybersecurity [190], a more restrictive classifiers could be a bad strategy, when considering a dynamic and adversarial environment. The one class model on the set of malicious training data, represents the other end of the spectrum, where a boundary is drawn around the malicious class samples. This model makes evasion easier, but ensures high adversarial uncertainty. This is also undesirable, as evasion can be performed by changing any of the N features. It could however benefit defenders facing multiple disjoint adversaries [3, 114].

A popular design strategy in adversarial domains, focuses on integrating multiple orthogonal feature space information to make a complex model, which is robust to changes in a subset of the features. A feature bagging ensemble, which uses majority voting on its component model's predictions, is a popular choice in this category. This model is resilient to attacks which affect only a few features at any given time. Most robust learning

strategies were evaluated against a targeted exploration attack, where the adversary starts with a predetermined attack sample and aims to minimally alter it, so as to evade the classifier C. By measuring adversarial cost in terms of the number of features which need evasion, the efficacy of feature bagged models was demonstrated [51], as robust models by design will require a majority of the features to be modified, for a successful evasion. This reasoning is not valid for the case of an indiscriminate exploratory attack, where an adversary is interested in generating any sample which evades C (as presented using the *Seed-Explore-Exploit* attack framework of Chapter 3), without any predefined set of attack samples. In such cases, an adversary is able to generate any attack sample at the same cost. In this setting, the impact of feature bagged ensemble is similar to that of a simple model, as seen from the evasion probability and adversarial certainty computations of Table 5.1.

An additional design strategy, relies on randomness to provide protection. By training models on multiple different subspace of the data and then randomly choosing one of the models to provide the prediction at any given time, these model aims to mislead attackers, by obscuring the feedback from the black box model C. Although this strategy has a higher perceived evasion resistance (Table 5.1), security through obscurity has shown to be ineffective when faced with indiscriminate probing based attacks [76].

5.3.3 Using Adversarial Margin Density (AMD) to approximate adversarial uncertainty

The use of adversarial certainty in Table 5.1, was illustrated by using a random probing attack on a binary feature space. Here, the idea of adversarial uncertainty is expanded and a methodology for its computation and usage is presented, for empirical evaluation.

Adversarial uncertainty is essentially the uncertainty on the adversary's part, due to the non availability of the original training dataset. For the task of classification, this uncertainty refers to the inability of the adversary to successfully reverse engineer the impact of all the important features, on the prediction task. The notion of partial evasion of

the feature space is employed, in defining the margin density drift detection algorithm in Chapter 4. The MD3 approach, defines regions of uncertainty (margin/blindspots) for a classifier and then uses it to detect concept drift in the data. The intuition behind this approach was that drifts seldom affect all informative features simultaneously, in real world applications. This same intuition can be used in defining adversarial uncertainty, as we are trying to measure the impact of an attack, based on their ability to impact the available features. An attacker that impacts only a minimal set of feature, so as to be sufficient to evade the defender's classifier C, will lead to a large margin density. This is due to the increased disagreement between orthogonal models, trained from the original training dataset. Conversely, an attacker with a low margin density is indicative of one who was successful in evading a large set of informative features. To this end, the notion of Adversarial Margin Density (AMD) is defined here, to capture adversarial uncertainty on the set of evaded features.

Definition 5.3.1. ***Adversarial Margin Density (AMD):*** *The expected number of successful attack samples (attack samples classified by the defender as Legitimate), which fall within the defender's margin (i.e., the region of critical uncertainty), defined by a robust classifier (one that distributes feature weights) on the training data.*

The definition of AMD highlights the following two concepts: a) The AMD is computed only on the attack samples which successfully evade the classifier C, and b) We measure only the region of critical uncertainty (margin/blindspots), predefined for a classifier trained on the original training data. The AMD measures the uncertainty by quantifying the attack samples which evade the classifier C, but do not have high certainty about the entire feature space. This is demonstrated in Figure 5.6, where the adversarial margin is given by the region of space where the defender C is evaded, but the feature $X2$ is still not successfully reverse engineered by the adversary. This causes the attacks to fall within blindspots of a robust classifier (i.e., one that distributes feature weights). For an attack to have a low uncertainty, it will have to avoid falling in the blindspots, which is possible only if an attacker can successfully evade a significant portion of the feature space (given

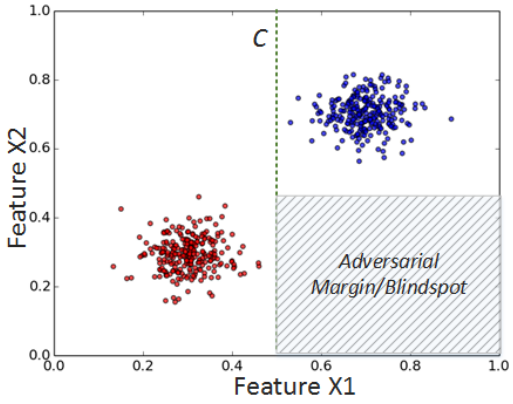


Figure 5.6: Illustration of adversarial margin. It is given by the region of space which leads to evasion of defender's classifier C, but does not lead to evasion of all the informative features.

by critical uncertainty), simultaneously.

The idea behind the AMD is presented in Definition 5.3.1. An implementation methodology which can be used to compute AMD is presented next. A random subspace ensemble is considered for defining the robust classifier, over the training dataset, as this classifier is general in its design and does not make assumptions about usage of any specific classifier type [191]. The AMD is computed using Equation 5.1.

$$AMD = \frac{\sum S_E(x)}{|x|}; \quad \forall x \in X_{Attack} : C(x) \quad is \quad Legitimate$$
$$where, \quad S_E(x) = \begin{cases} 1, & if \quad |p_E(y_{Legitimate}|x) - p_E(y_{Malicious}|x)| \leq \theta_{Margin} \\ 0, & otherwise \end{cases} \tag{5.1}$$

The AMD is measured only on the attack samples (X_{Attack}) which successfully evade the defender's classifier C, at test time. These set of samples are evaluated to determine if they fall within the region of critical uncertainty given by the parameter θ_{Margin}. By adjusting the parameter, the sensitivity of measurements and tolerance for adversarial activity, can be tuned. A value of 0.5 is typically considered effective for most scenarios. Samples falling in this region of high disagreement (given by parameter θ_{Margin}) are con-

sidered to be cases where there is critical uncertainty between the constituent models of the ensemble. Since the ensemble is trained as a feature bagged model, this disagreement is due to different feature value distributions, than the benign training samples, caused by adversarial uncertainty. The ensemble E is taken to be a random subspace ensemble. In the experimentation here, we consider a feature bagged ensemble of 50 base Linear-SVM models, each with 50% of the total features, randomly picked. In Equation 5.1, $p_E(y|x)$ is obtained via majority voting on the constituent models, and represents disagreement scores for each sample x. The AMD is always a ratio between 0 and 1, and a higher value indicates a high adversarial uncertainty. By extension, a higher AMD also indicates an increased ease of unsupervised detection and subsequent recovery from attacks.

5.3.4 Adversarial evasion using high confidence filter

In Chapter 3, the *Seed-Explore-Exploit (SEE)* framework was introduced, to simulate adversarial attacks on black box classifiers at test time. The framework was developed as a proof of concept, with adversarial goals being: the generation of samples which evade the defender's model and at the same time provide high variability, to make blacklisting based countermeasures ineffective. In this section, the SEE framework is extended further, to simulate sophisticated attackers, who are capable of using all the probe-able information to generate attacks of high adversarial certainty. Such adversaries can leverage the probed information, to launch attacks which can avoid falling in the margin/blindspots of the defender. Consequently, such an adversary can be harder to detect and stop. This extension of the SEE framework, will allow for testing against a sophisticated adversary, and will enable us to better understand the severity of exploratory attacks possible at test time. Since this extension maintains the original black box assumptions of the defender's model, it provides for analyzing the impact of severe exploratory attacks, which are possible to carry out against a classifier, by an adversary.

The proposed extension to the SEE framework is depicted in Figure 5.7. Specifically, the filter strategy is developed as an extension to the Anchor Points attacks (AP) of Sec-

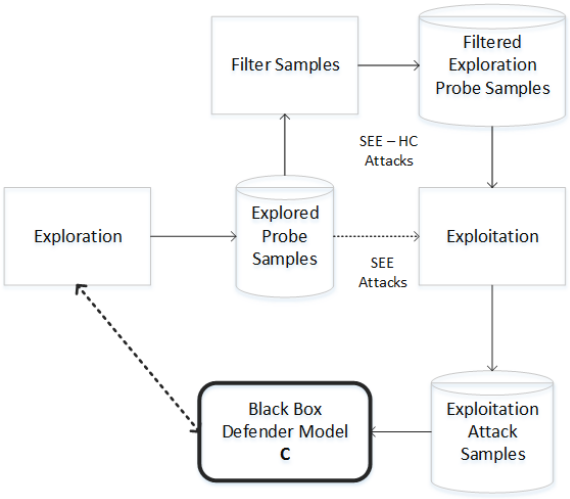


Figure 5.7: SEE based evasion attack framework, with a high confidence filter phase.

tion 3.3.2. The anchor points exploration samples obtained are first sent to a filter phase, where the low confidence samples are eliminated, before the exploitation phase starts. This filtering leads to a reduced size of exploration samples, for which the adversary has high confidence that they do not fall in the defender's margins. This approach is called the Anchor Points - High Confidence (AP-HC) strategy, and it is developed as a wrapper over the SEE framework, making it easy for extension to other data driven attack strategies as well. The AP-HC strategy will simulate adversaries who are capable of utilizing all the probe-able information, to launch attacks which generate low Adversarial Margin Density (AMD), on the defender's part.

In the filtering phase of the Figure 5.7, the adversary relies on stitching together information made available by the defender's black box, to launch attacks which evade a large number of features simultaneously. The adversary does so by integrating information learned in the exploration phase, about the impact of different subsets of features, on the prediction outcome. It then filters out samples which are good for evasion, but only result in evading a small subset of features. This will result in an attack exploitation, which would provide to high adversarial certainty. The idea is illustrated in Figure 5.8, on a 2D synthetic dataset. In this example, there are two sets of features $X1$ and $X2$, both of which are informative for the classification task, as $X1 > 0.5$ or $X2 > 0.5$ both lead to the

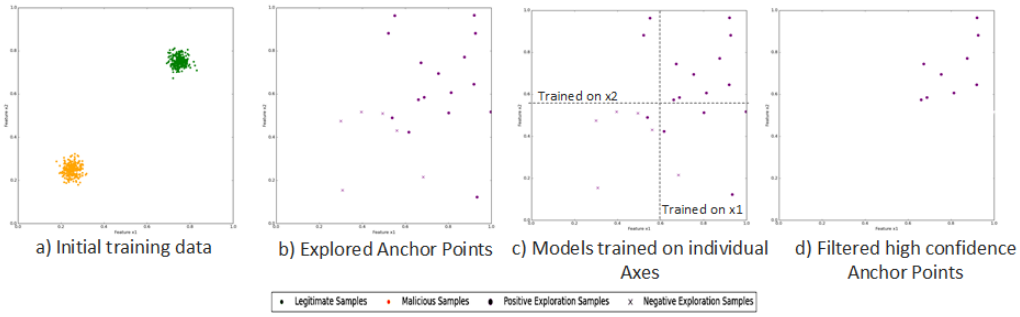


Figure 5.8: Reducing adversarial uncertainty via filtering, in anchor points attacks. After the exploration phase in b), the adversary trains models on individual feature subspaces (X1 and X2). It then aggregates this information to clean out samples in low confidence areas (C:X1≠ C:X2). The final set of filtered high confidence samples are shown in d).

high quality defender models. Consider the defender's model to be $'X1 > 0.5 \vee X2 > 0.5'$. As seen in Section 5.3.2, this leads to an adversarial uncertainty of 1/3, since the training data could be in any of the 3 quadrants where the samples are perceived as *Legitimate*. A naive adversary using the SEE framework, can launch an attack using samples where atleast one of the two feature $X1$ or $X2$ is greater than 0.5. However, an adversary using the high confidence filter attack will combine this orthogonal information and launch attack samples only if both conditions are satisfied (i.e., $X1 > 0.5$ and $X2 > 0.5$). In doing so, the adversary is learning the orthogonal information about the prediction landscape and aggregating this information to avoid detection by the defender. This strategy is illustrated in Figure 5.8, where the initial explored samples for the AP attacks (b), are filtered using the aforementioned information aggregation technique, to generate a high confidence set of exploration samples (c). The adversary does so by using the exploration samples in b), to train orthogonal models and admit only those samples which have high consensus (low uncertainty), based on the trained model. The filtered exploration samples are then used in the exploitation phase, and as can be seen in Figure 5.8, these attacks have an AMD of 0, as none of the attack samples fall inside the margin of the classifier.

In our analysis presented in Table 5.1, it was shown that Complex feature bagged ensembles models have a low adversarial certainty ($1/2^{N-1}$, for N dimensional binary feature space). This made it more secure in dynamic environments, when compared to the

restrictive one class classifier model, where the adversarial certainty was 1. However, similar to a one class classifier, the robust ensemble model also makes a majority of the information available, to be probed by a patient adversary, as demonstrated in Figure 5.8. This information is not directly available (as in the case of a one class classifier where evasion leads to an adversarial uncertainty of 0), and does not directly lead an adversary to the space of training data. However, the AP-HC filtering step can cause the adversary to stitch together information made available by such ensemble models, to launch a potent attack which could leave the defender helpless. A generic approach to extend the intuition of Figure 5.8, to high dimensional spaces, is presented in the following section.

5.3.4.1 The Anchor Points - High Confidence (AP-HC) approach

The AP-HC attack strategy is presented in Algorithm 5.1. The algorithm receives the exploration samples from the SEE framework, and then uses the filtering approach of Figure 5.8, to clear out samples of low confidence. The adversary does so by training a robust classifier, such as a random subspace ensemble, from the probed exploration samples. The trained model is then used to identify samples of low confidence, as these are the ones which have low consensus among the feature bagged ensemble's models. The resulting set of filtered exploration samples $D_{Explore-HC}$, from Algorithm 5.1, is then used in the exploitation phase of the SEE framework, to launch the attacks. This methodology is implemented completely on the adversary's side, while maintaining the same black box assumption about the defender, as presented by the SEE framework and the Anchor Points attacks in Section 3.3.1. As such, it serves as a methodology for thorough analysis of adversarial capabilities, to better design secure machine learning frameworks. The purpose of the AP-HC attack approach is to highlight the effects of making excessive information available to the adversary, and the resulting space of possible attacks which could be launched by it, using purely data driven methodologies.

The parameter $\theta_{Adversary\_confidence}$, controls the filtering operation in Algorithm 5.1. Since an adversary is interested only in high confidence attack samples, we consider a high

Algorithm 5.1: SEE based high confidence filter attacks (SEE-HC).

Input : Exploration Samples from SEE framework - $D_{Explore}$, Filter threshold $\theta_{Adversary\_confidence}$

Output: High confidence exploration samples $D_{Explore-HC}$

```
E ← Train random subspace samples from D_Explore
D_Explore-HC = ∅
for sample in D_Explore do
    if |p_E(y_Legitimate|sample) − p_E(y_Malicious|sample)| ≥ θ_Adversary_confidence
     then
        D_Explore-HC ∪ sample
return D_Explore-HC
```

confidence threshold of $\theta_{Adversary\_confidence}$=0.8. A higher confidence threshold will allow for more stringent filtering of samples. This heuristic filtering approach will be used to demonstrate the capabilities of an adversary to evade detection, by gaining more certainty about the location of the training data. This approach will be used to highlight innate vulnerabilities in seemingly secure designs, such as the robust feature bagged ensemble strategy, and will be used for thorough analysis of the impact of classifier design on adversarial capability.

5.4 Experimental evaluation and analysis

In this section, the impact of different classifier design strategies, on adversarial capabilities, is evaluated. Moving beyond accuracy, the effects of attacks from a dynamic-adversarial perspective, is analyzed. The idea of adversarial margin density is considered, to account for detect-ability and retrain-ability from attacks. The adversary is considered to be capable of using machine learning to meet its needs. In Section 5.4.1, the protocol and setup of experiments performed in this section, is presented. Effect of using a restrictive one class model for the defender is presented in Section 5.4.2. Impact of using a robust feature bagged ensemble and that of using randomization based models in presented in Section 5.4.3 and Section 5.4.4, respectively. Discussion and analysis is presented in Section 5.4.5.

5.4.1 Experimental setup and protocol

The basic *Seed-Explore-Exploit* framework of Chapter 3, is considered for generating attacks on the defender's black box model. In particular, the Anchor Points (AP) attack algorithm Section 3.3.2 is employed. An adversary starts with an exploration budget $B_{Explore}$=1000 probes, and then uses it to generate an attack of 2000 samples, via exploitation. The parameters of the AP attacks are taken without any modifications, to ensure consistent analysis and extension of the attack paradigms. The defender is considered to be a black box by the adversary, with no information about its internal working available. The only interaction the attacker has with the defender, is by means of submitting probe samples, and receiving tacit Accept/Reject feedback on them.

Adversarial margin density proposed in Section 5.3.3, is used for evaluating uncertainty of the attacks. A θ_{Margin}=0.5, for measuring the AMD (as motivated by MD3 in Chapter 4), is considered. A random subspace model with 50 Linear SVMs (regularization constant c=1, and 50% of the features in each model), is taken for the AMD computation. Also, the effects of filtering by an adversary is evaluated in this section, with a $\theta_{Adversary\_confidence}$=0.8. A random subspace model with 50 Linear SVMs (regularization constant c=10, and 50% of the features in each model), is chosen for the filtering task. A high regularization constant ensures that the models do not overfit to the exploration samples, as the adversary's goal is not to fit the model to the explored samples, but to learn from it about the region of high confidence. This also makes it more robust to black box feedback noise and stray probes.

Description of datasets used for evaluation is presented in Table 5.2. The synthetic datasets is a 10 dimensional dataset, with two classes. The *Legitimate* class is normally distributed with a μ=0.75 and σ=0.05, and the *Malicious* class is centered at μ=0.25 and σ=0.05, across all 10 dimensions. This datasets provides for controlled experimentation and analysis, by exhibiting 10 informative features. The CAPTCHA dataset is taken from [31], and it represents the task of classifying mouse movement data for humans and bots, for the task of behavioral authentication. The phishing dataset is taken from [145], and it represents

TABLE 5.2

Description of datasets used for experimentation.

Dataset	#Instances	#Features
Synthetic	500	10
CAPTCHA	1886	26
Phishing	11055	46
Digits08	1499	16
Digits17	1557	16

characteristics for malicious and benign web pages. The digits dataset [145] was taken to represent a standard classification task. The multidimensional dataset was reduced to a binary class problem with the class 1 and 7 taken for the Digits17 dataset, and the class 0 and 8 taken for the Digits08 dataset, respectively. In all datasets, the class 0 was considered to represent the *Legitimate* and 1 was taken as the *Malicious* class. For Digits17, the class 7 is considered to be *Legitimate*, and for the Digits08, class 0 is considered *Legitimate*. The data was normalized to the range of [0,1], using min-max normalization, and the features were reduced to a numeric/binary type. The records were shuffled, to eliminate stray effects of concept drifts within them. All experiments are repeated 30 times and average values are reported. The experiments are performed using python and the scikit-learn machine learning library [149].

5.4.2 Analyzing effects of using a restrictive one-class classifier for the defender's model

In adversarial domains, restricting the space of samples classified as *Legitimate*, is considered an effective defense strategy [92, 190]. By tightly restricting what data points are qualified as legitimate, the ability of random probing based attacks is significantly reduced. This is a consequence of the reduced probe-able feature space area, as shown in Figure 5.9, where a one class classifier is trained on the set of legitimate training data points. The feedback obtained from the defender's model, by probing the feature space is shown. The significantly smaller area of the blue feedback (Figure 5.9), is what makes

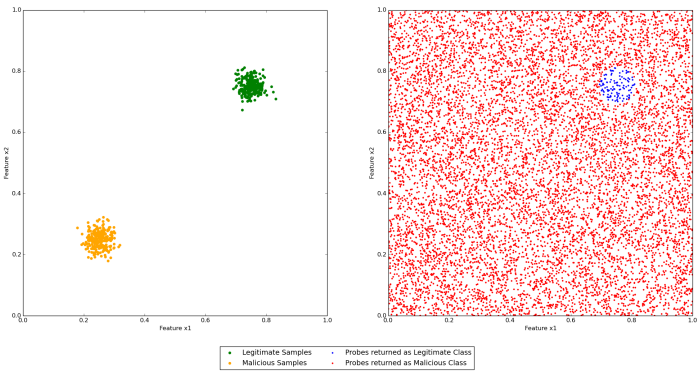


Figure 5.9: Illustration of prediction landscape of a one-class classifier. Smaller area of the legitimate samples indicate the resilience against probing based attacks.

one class classifiers harder to probe and reverse engineer. However, the one class classifier has limitation, which make them unsuitable for an adversarial domain. Firstly, a one class classifier sacrifices generalization ability and it is not always possible to train an accurate model over the available training data [92]. Secondly, as seen in our analysis in Table 5.1, a one class classifier could cause high adversarial certainty, making attack detection and recovery difficult. Most works on the security of classifiers advocate inclusion of all orthogonal information, to make the system more restrictive and therefore more secure [2, 40, 71, 72, 92, 190]. However, these works approach the security of the system from a static perspective. Evaluating a one class classifier provides valuable insights into the inefficacy of using restrictive models, in a dynamic adversarial environment. While recent works present various novel ideas for integrating feature information [4, 92], a one-class classifier serves as an ideal representative approach, in which all information across all features is used in securing the system.

In a dynamic environment, it is necessary to maintain adversarial uncertainty, so as to ensure that attacks can be recovered from. A one-class classifier, being overly restrictive, leads an attacker directly to the space of the legitimate training data. Although this classifier design makes the adversary expend greater effort to evade the system, once evaded the attacks are indistinguishable from the benign traffic entering the system. This is illustrated

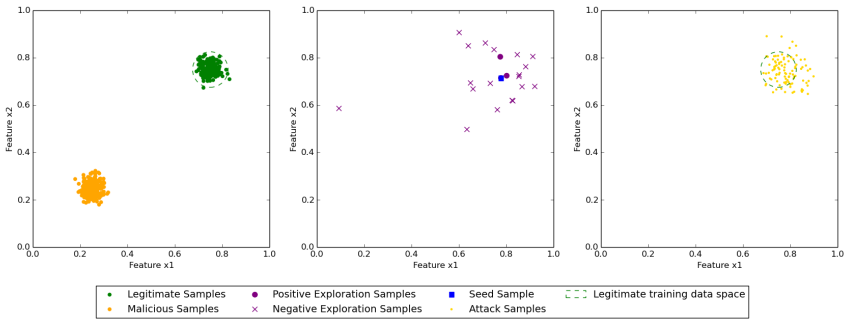
TABLE 5.3

Results of experimentation on one-class and two-class defender models. Training accuracy, Effective Attack Rate (EAR), Data Leakage (DL) and Adversarial Margin Density (AMD), are presented for comparison.

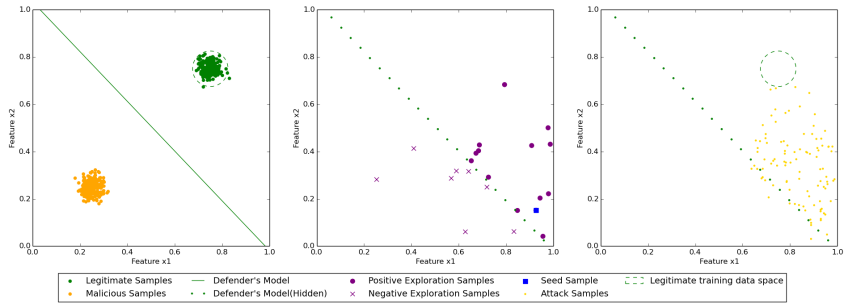
Dataset	Defender's Model	Training Accuracy	EAR	Data Leak	Adversarial Margin Density (AMD)
Synthetic (2 features)	Restrictive one class	97.4	0.56	0.67	0
	Robust two class	100	0.89	0	0.85
Synthetic (10 features)	Restrictive one class	97.6	0.04	0.67	0
	Robust two class	100	0.99	0.1	0.28
CAPTCHA	Restrictive one class	96.4	0.88	0.23	0
	Robust two class	100	0.99	0.01	0.14

in case of a 2D synthetic data in Figure 5.10. Here, the anchor points attack is used to generate attacks, on two different classifier designs: a) A one class classifier on the legitimate training data (SVM with parameters: ν=0.1, RBF kernel and γ=0.1), and b) A two class linear classifier model (Linear SVM with L2-regularization, c=1). The attack used 20 samples for exploration ($B_{Explore}$) and generated 40 attack samples, in each case. It is seen in Figure 5.10, that the attack leads to a large number of samples occupying the same region as that of the legitimate samples, in case of the restrictive one class defender model (Figure 5.10a)). This will cause problems in a dynamic environment, as retraining to thwart attacks is close to impossible in this case. Also, detecting such attacks is difficult, due to the increased similarity with benign input. In case of the two class model (Figure 5.10b)), a larger data space is perceived as legitimate, due to the generalization provided by these models, leading to attacks which are farther from the training data space. This illustrates the ability of classifier model designs, to influence the severity of attacks on the system, and to cause higher adversarial uncertainty.

Experimental evaluation of the synthetic 2D dataset, and two additional datasets from Table 5.2, is presented in Table 5.3. Here, the metric of *Effective Attack Rate* (EAR) is taken to measure the vulnerability of the the defender's model, to anchor points attacks. Additionally the following two metrics are introduced, to measure the effectiveness of attacks in a dynamic environment - *Data Leakage* and the *Adversarial Margin Density* (AMD).



(a) Use of a one class classifier by the defender causes reduced attack rate, but leads to increased training data corruption and leakage.



(b) Two class classifiers ensures less data leakage, but makes evasion easier.

Figure 5.10: Illustration of AP attacks on a synthetic 2D dataset, using a restrictive one class classifier and a generalized two class classifiers, for the defender's model.

Adversarial margin density was introduced in Section 5.3.3, as a measure for approximating adversarial uncertainty over the defender's training data. Data leakage is introduced here as an adhoc metric for evaluating the loss of private data, in a one class classification setting. Data leakage is measured by developing a one class classifier on the space of legitimate training samples, and then measuring the number of attack samples which are incorrectly classified by this classifier. Data leak is used to measure the proximity of the attack samples, to the original space of legitimate training samples, with a large value

indicating the ability of the adversary to closely mimic the legitimate samples. A one class SVM model (parameters: ν=0.1, RBF kernel, γ=0.1), is used to measure the data leakage metric. The metrics were computed for the case of the one class and the two class defender's model, as shown in Table 5.3.

The Effective Attack Rate (EAR), is significantly lower for the one class classifier (Δ=0.46 on average, from Table 5.3). This is a result of the stricter criterion for inclusion into the legitimate space, as imposed by the complex and restrictive classifier boundary. However, this comes at the cost of an increased possibility of data leakage and lower adversarial uncertainty. From Table 5.3, it can be seen that the data leak increases sharply for the one class classifier (Δ=0.49, on average), as the restrictive nature of the classifiers leads the adversary to the training data samples. This causes problems with retraining, loss of privacy, loss of clean training data, and issues with unsupervised attack detection. The data leak metric provides a heuristic way to measure the severity of data nullification attacks. However, the data leak metrics is difficult to compute for high dimensional datasets, due to the inability to train an accurate one class classifier, which is needed to measure the data leakage. For this reason the other datasets of Table 5.2, are not used for the analysis in this section. Also, these datasets provided low training data accuracy when using a one class classifier, making it unsuitable as a choice for the defender's model. Going forward, the adversarial margin density metric will be used, to indicate the strategic advantage of the defender over the adversary, in detecting attacks and relearning from them.

The adversarial margin density (AMD) is more general in its applicability, when compared to the data leak measure, and provides a way to indirectly measure adversarial uncertainty and the severity of attacks in dynamic environments. From Table 5.3, it is seen that the AMD is 0 for all 3 datasets, when using a one-class defender model. This is due to the increased confidence in the location of the training data, once the restrictive model is evaded. Although the one class classifier is secure, as it ensures a low EAR, it leaves the defender helpless once an attack starts. As such, designing for a dynamic environment requires forethought on the defender's part. The experimentation in this section was pre-

sented for illustrating the ill effects of using an overly restrictive model for the defender, and the need to reevaluate the notion of security which relies on generating complex learning models.

5.4.3 Analyzing effects of using a robust feature bagged ensemble for the defender's model

Robust models, which advocate complex models involving a large number of informative features, are considered to be effective in safeguarding against targeted exploratory attacks [2, 4, 40, 71–73, 87]. In these attacks, the adversary starts with a predefined set of attack samples, and intends to minimally modify it, so as to evade the defender. Robust models require that a majority of the feature values be mimicked by the adversary, increasing the cost and effort needed to carry out attacks. However, in the case of an indiscriminate exploratory attacks, this same line of reasoning is not valid, as an attacker is interested in any sample that causes evasion. In these attacks, the ease of evasion is given by the effective space of prediction, which is recognized as *Legitimate* by the defender. This is because an adversary launching indiscriminate attacks, does so by probing the black box model to find samples which would be classified as legitimate. A robust model, such as a Linear SVM with L2-regularization, incorporates information conveyed by a majority of the features, from the training dataset, into the learned model. It does so to make a model robust to stray changes, and also to generate wide generalization margins, for better test time predictive performance. In an adversarial environment, the effective area of legitimate samples conveyed by a robust model, is similar to that of a simple model (which aims at reducing the number of features in the model), as shown in Figure 5.11. In Figure 5.11, a non robust linear SVM (with L1-regularization) is considered alongside a robust linear SVM (with L2-regularization). The effective area of legitimate samples (blue) is the same in both cases. This makes the two strategies equivalent in terms of securing against indiscriminate probing based evasion attacks.

The equivalence of robust and non robust models, to indiscriminate probing attacks,

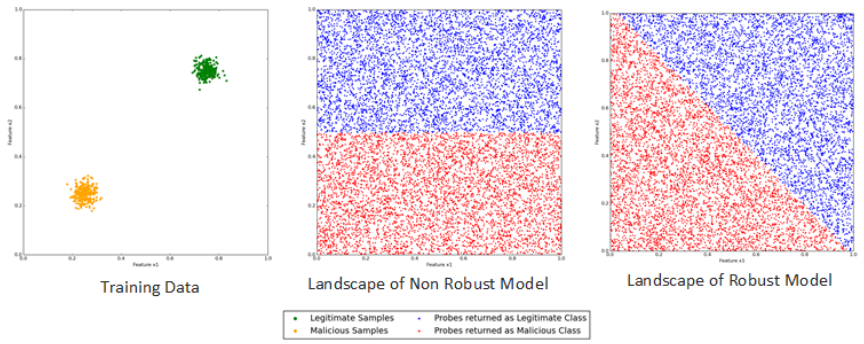


Figure 5.11: Illustration of prediction landscape using simple and robust models, on 2D synthetic data. *Left*- Initial training data, of the defender. *Center*- L1-regularized linear SVM model for the defender (Non robust). *Right*- L2-regularized linear SVM model for the defender (robust).

is further analyzed by evaluating on 5 datasets in Table 5.4. The anchor points attacks (AP), are performed on two sets of classifiers: a) A robust classifier - Random subspace ensemble with 50 linear SVMs (L1-regularized, each with 50% of the features randomly selected), and b) A non robust classifier- Single linear SVM (with L1-regularization). The effect of these strategies on the adversarial outcome is presented in Table 5.4. The Effective Attack Rate (EAR) is seen to be similar for the two cases (Δ=0.01, on average). This demonstrates the equivalence in effects of the two strategies, when it comes to securing against probing based attacks. The similarity in the diversity metrics of the attacks, measured using diversity metrics of Section 3.4.1.2 (standard deviation (σ), K-nearest neighbor distance(KNN) and minimum spanning tree distance (MST)), further indicate that the attacks operate in the same extent of space for the two models.

Based on the analysis on a binary feature space in Table 5.1, it was deduced that simple models behave similar to robust models, when attacks are of an indiscriminate nature, as both result in the same adversarial certainty of $1/2^{N-1}$. From the same analysis, it was deduced that robust models are better than one-class classifiers, in maintaining adversarial uncertainty. The intuition behind this was that the increased generalization of robust models will create uncertainty regarding the exact location of the training data, and the impact of various feature subspaces on the prediction task. However, this intuition relied on the assumption of a naive adversary, whose primary aim is to evade the system only.

TABLE 5.4

Results of AP attacks on robust and non-robust defender models.

Dataset	Metric	Non Robust Model	Robust Model
Synthetic	Training Accuracy	100	100
	EAR	0.99	0.98
	Diversity (MST, KNN, σ)	(0.35, 0.30, 0.21)	(0.35, 0.29, 0.21)
CAPTCHA	Training Accuracy	100	100
	EAR	0.98	0.99
	Diversity (MST, KNN, σ)	(0.74,0.64,0.21)	(0.80, 0.689, 0.22)
Phishing	Training Accuracy	94.1	92.6
	EAR	0.99	0.98
	Diversity (MST, KNN, σ)	(1.05,0.91,0.22)	(1.08,0.932,0.22)
Digits08	Training Accuracy	97.7	97.1
	EAR	0.97	0.94
	Diversity (MST, KNN, σ)	(0.50,0.44,0.23)	(0.456,0.40, 0.24)
Digits17	Training Accuracy	99.8	99.6
	EAR	0.98	0.97
	Diversity (MST, KNN, σ)	(0.52,0.45,0.22)	(0.5,0.44,0.22)

The proposed high confidence filtering attack strategy (AP-HC) of Section 5.3.4, provides a way to simulate a sophisticated adversary, who is capable of utilizing all the probe-able information to launch attacks of high certainty. The result of using this attack strategy on the robust classifier model is presented in Table 5.5. It is seen that the AMD for the robust classifier significantly reduces, when faced with the high confidence attack. In case of the CAPTCHA dataset and the Synthetic dataset, the adversarial activity will go totally unnoticed, while in case of other datasets, a significant drop in the AMD is observed (Δ=0.38, on average over all datasets). An important observation comes from the fact that the filtering operation was performed completely on the adversary's side, without any help or information from the defender's model. The adversary starts off with the goal of admitting only the most confident exploration samples, and in doing so, it makes it difficult for the defender to detect or stop it. This result indicates that robust classifiers and the

TABLE 5.5

Results of Anchor Points (AP) attacks and Anchor Points – High Confidence (AP-HC) attacks, on the Effective Attack Rate (EAR) and the Adversarial Margin Density(AMD).

Dataset	Training Accuracy	EAR		AMD	
		AP Attacks	AP-HC Attacks	AP Attacks	AP-HC Attacks
Synthetic	100	0.98	0.999	0.28	0.01
CAPTCHA	100	0.996	0.999	0.19	0.002
Phishing	93.1	0.978	0.999	0.62	0.19
Digits08	97.1	0.928	0.999	0.73	0.22
Digits17	99.5	0.965	0.999	0.66	0.16

margin density approach to attack detection, are both themselves susceptible to adversarial activity and need to be made secure, to be effective in a dynamic adversarial environment.

The effectiveness of the high confidence attacks, is due to the availability of all information, to be probed by a sophisticated adversary. The majority voting scheme of the robust ensemble, is reverse engineered by stitching together orthogonal subsets of feature information, to generate attacks which closely mimic the legitimate samples. As such, the robust ensemble methodologies behave similar to one-class classifiers, by being vulnerable to low adversarial uncertainty. Both design approaches rely on incorporating maximum extracted information from the training data, thereby conveying excessive information to an adversary and equipping it with a thorough understanding of the impact of different features to the classification task.

5.4.4 Analyzing effects of using randomization in the defender's model

Robustness via complex learning methodologies aims at increasing the adversary's effort, by making it reverse engineer a large set of features, as seen in the case of one-class classifiers and the feature bagged ensemble. The other advocated approach to dealing with adversarial activity relies on obfuscation via randomization. By randomizing the feedback presented to the attacker, these methodologies aim to mislead adversarial learning from probes, leading to less effective attacks. Randomness can be introduced into multiple clas-

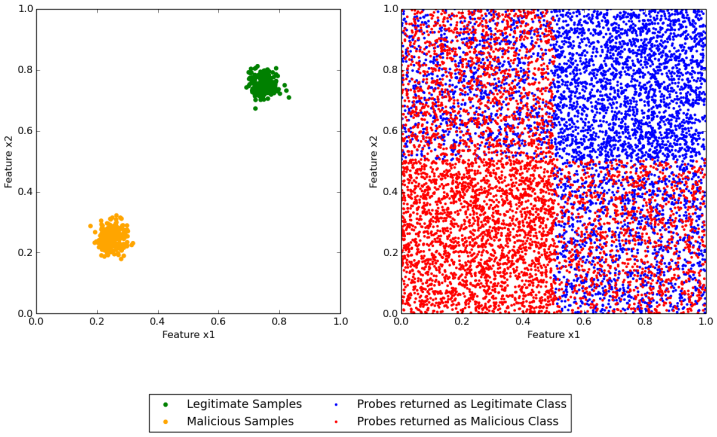


Figure 5.12: Prediction landscape for the randomized feature bagging ensemble. Blindspots are perceived to be obscured, while high confidence spaces remain consistent across repeated probing.

sifier systems, particularly the ones using feature bagging. This is done by training multiple models on subsets of features and then using any one of these models to provide prediction at any given time [4, 72, 75]. The idea relies on confusing the adversary by constantly presenting it with a moving target. In this section, the impact of randomization is analyzed, when used with a feature bagged ensemble, against indiscriminate evasion attacks.

Randomization is introduced into the defender's model, by extending the random subspace ensemble of Section 5.4.3, to generate feedback based on the posterior probability for a given sample X. The use of random subspace ensemble allows for the randomness be caused due to disagreement between orthogonal feature information. Given a sample X, the defender computes the confidence on it, based on majority voting of its component models. This confidence is then used as a probability of prediction, to generate the class label for X. As an example, consider a sample X for which the classifier C predicts with 0.8 probability to be in the *Legitimate* class. A standard threshold of 0.5, based on majority voting, will cause the feedback on X to be of class *Legitimate*. To introduce randomness, the defender's model will instead sample number in the range of [0,1] and return feedback *Legitimate*, only if the random number is >0.8. As such, randomness is introduced into

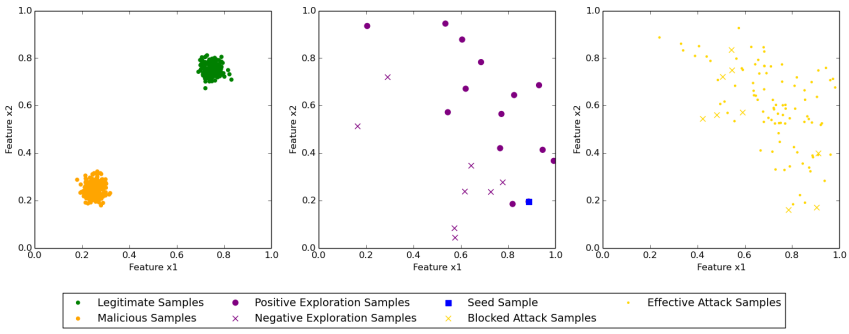


Figure 5.13: Anchor Points attacks against a randomized defender classifier.

the classification scheme, while still maintaining the essential predictive properties of the classifier. Here, the adversary may not be aware of the internal scoring or the randomness of the black box model, as it still only experiences the defender as a black box model providing *Legitimate/Malicious* feedback on its submitted probe X. The perceived space of the adversary is shown in Figure 5.12. The regions of uncertainty is heavily influenced by randomness, due to the disagreement between models trained on the two features.

The effects of randomness is demonstrated over a synthetic 2D dataset in Figure 5.13, where the anchor points (AP) attacks is used. The misleading feedback from the defender, causes the exploration phase to be corrupted, due to the naive assumption on the adversary's part about the veracity of the defender's feedback. This is presented in Table 5.6 for the 5 datasets. An average drop of 22.9% in the Effective Attack Rate (EAR), is seen. This comes at the cost of reduced accuracy for the defender (Δ=1.65, on average), because of randomized response returned by the defender even for a few of the legitimate samples. This is unavoidable, as attack samples are no different than the regular traffic faced by the defender, since both use the same input channels to access the system.

The analysis of a naive adversary assumes that the defender always returns the correct feedback on the submitted probes. This was seen to result in the attacker being mislead, as seen for the EAR of the naive attacker in Table 5.6. However, an adversary can

TABLE 5.6

Results of randomization against a naive adversary and an adversary capable of filtering low confidence probes.

Dataset	Metric	Randomized Model Naive Adversary	Randomized Model- Filtered Confident Samples
Synthetic	Training Accuracy	100	100
	EAR	0.85	0.97
	Diversity (MST, KNN, σ)	(0.35, 0.93, 0.2)	(0.31, 0.27, 0.19)
CAPTCHA	Training Accuracy	99.9	99.9
	EAR	0.9	0.97
	Diversity (MST, KNN, σ)	(0.79, 0.68, 0.21)	(0.71, 0.62, 0.21)
Phishing	Training Accuracy	89.6	89.6
	EAR	0.74	0.89
	Diversity (MST, KNN, σ)	(1.05,0.91,0.22)	(0.78,0.69,0.25)
Digits08	Training Accuracy	94.6	94.8
	EAR	0.624	0.89
	Diversity (MST, KNN, σ)	(0.46,0.40, 0.25)	(0.35, 0.32,0.27)
Digits17	Training Accuracy	96.9	97.1
	EAR	0.67	0.87
	Diversity (MST, KNN, σ)	(0.51,0.44,0.23)	(0.37,0.33,0.25)

become aware of the randomness, by submitting the same probe multiple times and observing different responses on it. Such an adversary can account for randomness in designing its attacks. A simple heuristic strategy is simulated here, to understand the behavior of such an adversary. In the exploration phase, the adversary makes repeated submission on every exploration point, to understand the defender's confidence in the sample. A sample which is closer to the training data, will generate feedback with more consistency, while samples falling in blindspots will be more random in their feedback (Figure 5.11). Using this intuition, the the anchor points (AP) attack strategy is modified, to account for smart adversaries capable of dealing with randomness. In the exploration phase, the adversary submits each sample $N_{Retries}$ (taken as 5 in experiments here) times, and accepts the probe

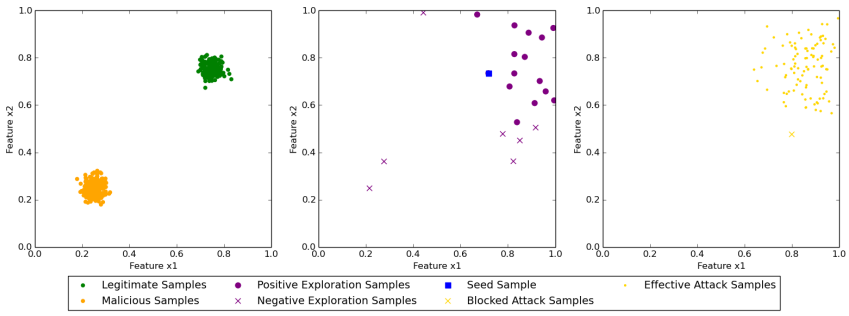


Figure 5.14: Anchor Points attacks with confidence filtering against a randomized classifier. Repeated probing of exploration samples is used to weed out samples with inconsistent feedback.

to belong to the *Legitimate* class, only if it returns the same feedback for all $N_{Retries}$ times. All other samples are assumed to belong to the $Malicious$ class. By cleaning samples of low confidence, the blindspot exploration samples are removed, making the exploitation phase more potent, as demonstrated in Figure 5.14.

Applying the filtering step allows for simulating an adversary capable of dealing with randomness. The results of such an adversary is shown in Table 5.6, where an increase in 16.7% in the EAR is seen, for such attackers. After this filtering, the attacks are similar to that on a robust model, with only a 5.1% difference on average, as seen in Figure 5.15. This demonstrates the ineffectiveness of the randomization approaches in providing security, against probing based attacks. There is increased onus on an adversary to use more probes to validate the exploration samples, but if a possible adversary makes this additional investment, incentivized by a high EAR, the randomization approach fails. The reduced diversity in case of the confident samples, in Table 5.6, is because the space of the actual confident legitimate samples, is much smaller than the one perceived by the naive adversary.

The inability of Restrictive models, Complex learning models and Randomization based models, to continue providing security in a dynamic adversarial environment, was highlighted in these sections. These approaches are advocated by works in the area of

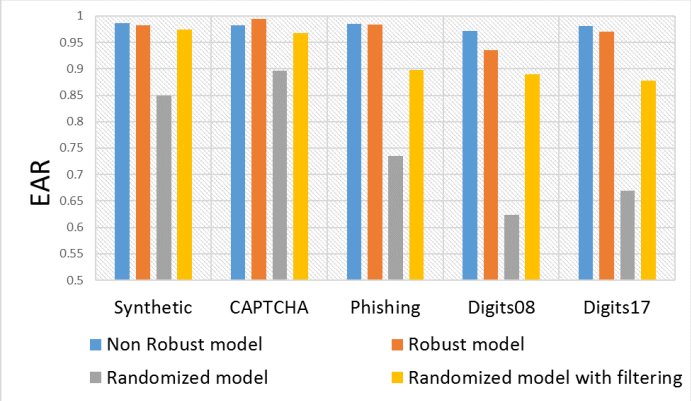


Figure 5.15: Comparison of Effective Attack Rate (EAR) for Non robust models, Robust models, Randomization based models, and Randomization models with adversaries capable of filtering low confidence samples.

machine learning security, as effective ways to ensure security against targeted exploratory attacks. However, when faced with an adversary capable of launching indiscriminate evasion attacks, these methodologies provide no better security than a simple two class classifier. Also, the overly restrictive or informative models were seen to leak training data information to the attacker, causing them to launch attacks, which are difficult to attack and recover from. Additionally, randomness as a strategy was seen to be ineffective, based on a simple repeated probing based modification by the adversary. Randomness is detect-able and manageable, on the adversary's end. It makes getting the information difficult, but like complex learning models they also reveal excessive information about the importance of features to the prediction task.

5.4.5 Why informative models are ill suited for adversarial environments?

The evaluation of classifier design from the perspective of adversarial uncertainty provides new insights into their vulnerabilities, when applied in a dynamic adversarial environment. The model design strategies of one-class classifiers, robust feature bagging models, randomized feature bagging models and even the dynamic margin density based drift detection technique (MD3), are all susceptible to attacks which can leave detection and retraining, intractable. All these methodologies try to include the maximum information from the training data, into the learned model, in order to maintain an information

advantage over the adversary. This is seen to be the general trend in the adversarial machine learning research community [40, 66, 72], where including more information into the models is believed to make it more secure. However, incorporating more information into the learned model means that the adversary will be able to reverse engineer and learn more of this information from the deployed model, via probing. In case of a one-class classifier, which advocates using maximal information to develop a tight boundary around the training data, it was seen that an attacker is led straight to the space of the legitimate data (Section 5.4.2). In case of Robustness and Randomization, information available is made harder to mimic/extract. In robustness, more features need to be mimicked to gain access, while randomization aims to mislead the adversary's learning. It is seen that both fail against a determined adversary, who given time/resources can learn and leverage all the available information.

Robust models are also seen to fail at detection, when faced with an adversary who is capable of generating high confidence attacks. These also stems from the increased availability of probe-able information, to be used by an adversary. These defense strategies relies of unrealistic assumption on the adversary's part, to ensure safety. It is generally assumed that if an attack is too expensive (i.e., requires many probes or the modification of many features), the adversary will give up. While this assumption is the basis of security against targeted evasion attacks, it does not hold in case of indiscriminate exploratory attacks. This section aimed to highlight some of the issues with incorporating excessive information into the defender's model. In an adversarial domain, the attack is a function of the deployed model. A highly informative model will lead to a highly confident attack. It is necessary to reevaluate this central idea of overly informative defender models, and analyze the effects of causing an information gap between the defender and the attacker, for improving security.

5.5 Hidden classifier design for dynamic adversarial environments - the *Predict - Detect* classifier framework

In this section, the idea of hiding feature importance from being probed and learned by an adversary, as a classifier design strategy for dynamic adversarial environments, is proposed. By limiting the amount of probe-able information in the defender's model, the ability to ensure adversarial uncertainty is emphasized. In Section 5.5.1, the motivation for the strategy of hiding feature importance, is presented. Section 5.5.2 develops this idea into the *Predict - Detect* classifier design. Potential benefits of this design are presented in Section 5.5.3.

5.5.1 Motivation - The effect of hiding feature importance

The approaches of *Complex learning* and *Randomization*, rely on including maximum information from the training dataset, into the learned model. As such, they are vulnerable to information leakage via adversarial probing and reverse engineering. Experimentation in Section 5.4, demonstrated that a classifier which exposes excessive information about the feature importance is susceptible to high confidence attacks. The increased confidence in attacks leads to issues with adversarial detect-ability, data leakage and retrain-ability. As such, the impact and ability to hide the importance of some of the features, to ensure that the model is shielded from total reverse engineering, is evaluated here.

The idea behind feature importance hiding relies on eliminating a few of the important features from the classification training phase, so as to intentionally misrepresent their importance to an external adversary. By eliminating these features from the classification model, the adversary is made to believe that they are not important for the prediction task. No amount of probing will help the adversary to ascertain the importance of the hidden features, as their importance and informativeness is shielded from the classification task. Although, this does not provide any direct benefits against the onset of evasion attacks, they help in maintaining adversarial uncertainty. An attacker will not be able to completely reverse engineer the importance of all features, no matter the resources/time used by it, as

they are not available in the black box model. This idea of feature importance hiding, is illustrated in Figure 5.16. The prediction landscapes of the following classifiers are depicted: a) A restrictive classifier aggregating all feature information ($X1 \wedge X2$), b) A randomization based classifier which picks its output based on either feature's prediction ($X1 \vee X2$), and c) The hidden feature classifier where feature $X2$'s influence is hidden. The choice of these designs affects the adversarial uncertainty, at test time. Based on experimentation in Section 5.4.2 and Section 5.4.4, we see that restrictive models (a)) and randomization models (b)) are ineffective, as an intelligent adversary can reverse engineer them to generate high confidence attacks. The robust model of Figure 5.16 a) reduces the effective space of legitimate samples, but in doing so it leaks information about the importance of features $X1$ and $X2$. The randomized model of Figure 5.16 b), aims to mislead the adversarial probing, but is vulnerable to repeated probing attacks, as demonstrated in Section 5.4.4. In case of the hidden feature importance design of Figure 5.16 c), an adversary sees a misrepresented view of the prediction landscape. The adversary is led to believe that only feature $X1$ is important to the defender's model. Since $C2$ is kept hidden, the attacker has information about only half the feature space and no amount of probing will help it understand the importance of feature $X2$ to the prediction problem. Attacks generated at test time, will fall under the blue region to make them effective, and when they do so, they have 50% chance of falling under the blindspot $B2$. $B2$ serves as a honeypot in the learned model, which helps capture attacks. An important distinction between the robustness and randomization approaches, in comparison to the hidden classifier approach, is that in the former case the defender expects the adversary to be dissuaded by the increased cost of evasion (in terms of probing budget), while in the latter case, the defender is making it unfeasible for an attacker to probe and obtain information necessary to generate a high confidence attack.

It should be noted that, the idea of hiding feature importance is not in direct violation of the Kerckhoffs's security principle [95, 96], which states that security should not rely solely on the unavailability of information, on the part of the adversary. This is because the defender is not using hidden features, but only misleading the adversary into believing

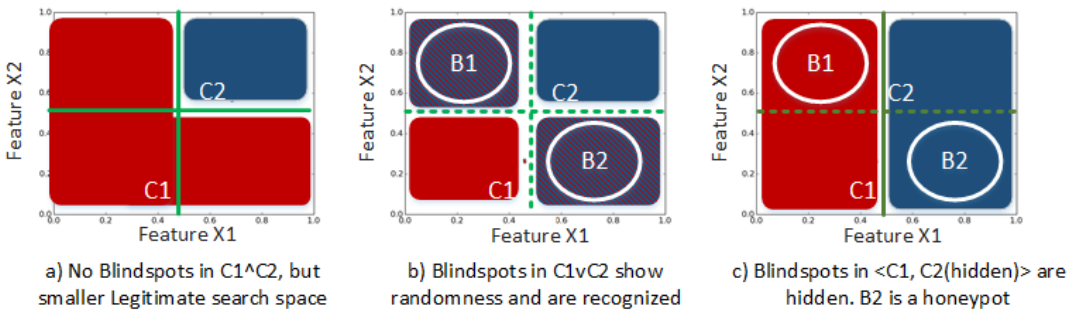


Figure 5.16: Prediction landscapes as perceived by probing on the defender models. *a)*: Defender model is given as $C1 \wedge C2$. *b)*: Defender model given by randomly selecting $C1 \vee C2$, to perform prediction. *c)*: Defender model is given by C1 (trained on feature X1), while C2 is kept hidden to detect adversarial activity.

that a few of the features are not important to the prediction task. As in Figure 5.16, the attacker is aware of features $X1$ and $X2$, but infers that only $X1$ is important to the classification task. All features are included in the classification model, but by intentionally hiding the importance of a subset of the features, the adversary is misled into generating attacks of partial confidence only (partially mimics the training legitimate data).

In order to demonstrate the effect of hiding feature importance, experimentation is presented on a classifier model with 50% of the features intentionally eliminated from the classification process, in Table 5.7. A random subspace model is chosen as the defender's model (50 Linear SVM, 50% of features per model). A random subset of 50% of the features are considered to be eliminated from the classification process. This is done by eliminating the features from the training dataset, and then training the defender's model on the reduced set of features. Any incoming probe is first truncated to the reduced feature set and then evaluated on the model. This makes the feature reduction strategy opaque to the external users. The results of the Anchor Points - High Attacks (AP - HC) attacks on this classifier is presented in Table 5.7. The Effective Attack Rate (EAR) and the Adversarial Margin Density (AMD), are evaluated on this hidden feature classifier design and also a baseline classifier which uses all features in its model, as presented in Figure 5.16.

It can be seen that the training accuracy is only marginally affected (<5% difference at max), by the reduction of features from the models. This is a result of the presence of

TABLE 5.7

Training Accuracy, Effective attack rate (EAR) and Adversarial Margin Density (AMD) under AP-HC attacks, for defender using all features and one which uses only half of the features.

Dataset	Training Accuracy		EAR		AMD	
	All Features	Hidden Features	All Features	Hidden Features	All Features	Hidden Features
Synthetic	100	100	1	0.99	0.01	0.172
CAPTCHA	100	100	0.99	0.99	0.002	0.075
Phishing	93.1	89.4	0.99	0.99	0.19	0.562
Digits08	97.1	95.1	0.99	0.99	0.22	0.648
Digits17	99.5	94.9	0.98	0.99	0.16	0.517

orthogonal information in the features of the training data. In case of a robust ensemble design, which incorporates all information in the model, the AMD is seen to drop as the attacks leverage the orthogonal information from the probes, to avoid regions of low certainty. However, for the hidden features approach, a significantly higher AMD value is seen in all cases (0.28, on average). This is because no amount of probing and cleansing will help the adversary in determining that the hidden features are important to the classification, and what exact values of the features they need to mimic. The EAR of all attacks is comparable to that of the robust classifier (Table 5.7). However, the increased AMD ensures that the defender has an upper hand in the attack-defense cycle, as it can detect attacks and keep the training data clean for future retraining. This makes hiding of feature importance an effective strategy in providing reactive security to adversarial drift. The notion of hiding feature importance suggests that not all information available at the training phase should be used at the same time, to ensure leverage over the adversary. An improved mechanism to employ this intuition in the defender's classifier design will be discussed next, as the coupled *Predict - Detect* classifier strategy.

5.5.2 The *Predict - Detect* classifier design

The *Predict - Detect* classifier framework is proposed here, to leverage the benefits of feature importance hiding, as presented in Section 5.5.1. The proposed design is depicted in Figure 5.17. In this framework, the available set of features are divided into two subsets.

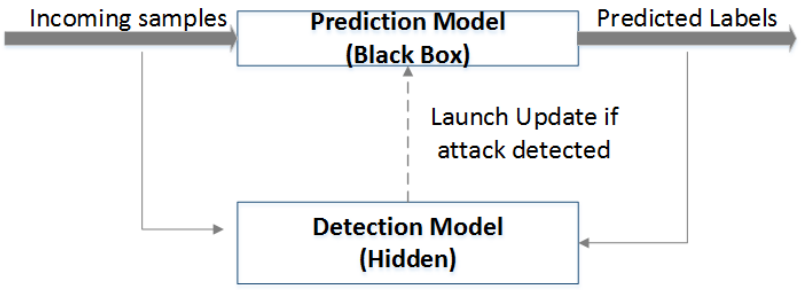


Figure 5.17: The Predict-Detect classifier design.

The first subset of features is used to form the defender's black box model, to perform prediction on the input samples. This classifier is called the *Prediction* model, as its primary purpose is to perform prediction on the input samples, submitted by either the benign users or the adversary. Since this forms the black box model, it is vulnerable to probing based attacks by an adversary. The *Prediction* model is expected to get attacked at some point, after deployment, due to the nature of the adversarial environment. The second subset of features is used to train a classifier, called the *Detection* model. This classifier is not used for any of the prediction task and as such is shielded from external adversarial probes. This model represents information known by the defender, based on the original training data, but not accessible via probing by an adversary. The purpose of the *Detection* model is to indicate adversarial activity, based on disagreement with the *Prediction* model. Since an adversary launches an attack based on information learned from the black box, an attack is characterized by evasion of the *Prediction* model, but only partial evasion of the *Detection* model. It should be noted that this division of features is opaque to the adversary, who still submits probes on the entire feature set. The division of features is done internally by the framework. Thus, no additional information is leaked to an adversary. Also, this framework does not advocate feature hiding, but instead relies on misrepresentation of the importance of the features, to the classification task.

This design has several advantages. Firstly, the separation of features between the hidden and the prediction model ensures that the hidden features are not reverse engineered or mimicked, thereby ensuring that total corruption of the training data features is

prevented. Secondly, the hidden features ensures that adversarial uncertainty is preserved, because of the inability of attacker to avoid the margin of detection. Lastly, the hidden features model provides a set of untarnished features for retraining and deploying in the event of an attack.

The Predict-Detect classifier is illustrated on a 2D dataset in Figure 5.18, where the defenders model provides the prediction, as the black box, and the hidden model is known by the defender only. Under such a setting, the adversary is made to believe that only Feature $X2$ is important to the classification (by reverse engineering using probes) and that feature $X1$ has no significance. This is made possible, as the defender indeed does not use the feature $X1$ in its black box. No amount of probing can help the adversary to determine any information about $X1$'s importance to the classifier. Consequently, the attacks are generated by the adversary and this leads to a large number of samples being captured at the *Detection Region*. This detection region is the classifier's adversarial margin/blindspot, and it is the region where the *Prediction* and the *Detection* models disagree. A large number of samples in this area indicates an attack, as an adversary exhibits a lack of knowledge imposed by such a classification design. The *Detection Region* serves as a honeypot to detect adversarial activity, and as such can be used in dynamic environments to detect and relearn from attacks. This preemptive classifier design ensures adversarial uncertainty, and benefits future unlabeled drift detection. It is an ideal example of the *Dynamic Adversarial Mining* principle, where defenders prepare not only for the attacks, but also for recovery and continuity of operation.

The effects of the coupled classifier approach is presented in Table 5.8, where a random subspace ensemble (50 linear SVM, 50% of features in each model), is used to train the prediction and the detection models. The feature set is divided such that 50% of the features are chosen to be part of the *Prediction* model and the rest are chosen to be a part of the *Detection* model, at random. It is seen that the training accuracy of the *Prediction* model is not significantly affected (<6%, at max) due to the loss of 50% of the predictive features. This is a result of orthogonal informative in the training dataset.

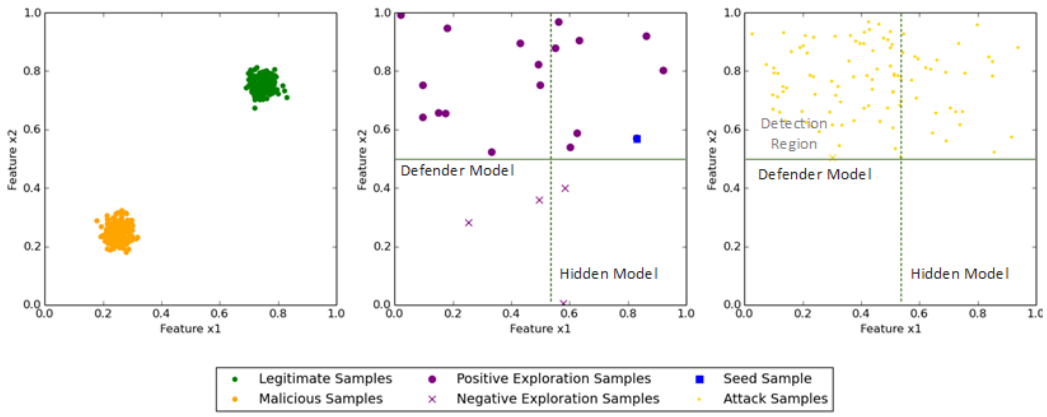


Figure 5.18: Illustration of the *Predict-Detect* classifier on 2D synthetic data. Samples are captured by the ensemble blindspot, shown as the Detection Region.

There is no significant improvement in the evasion rate either, indicating the equivalence of the two approaches in a static environment. However, in a dynamic environment, the classifier using all features, does not lead to significant adversarial uncertainty. This is a result of the adversary utilizing the probed information, to generate high confidence attacks. However, the disagreement computed by the *Predict - Detect* classifier shows a strong indication of attacks, as attack samples are captured by the feature honeypot created by the classifier design. The large number of samples being detected by this setup (<50% in 3 cases), indicates that the setup will ensure that adversarial samples will be easier to recognize, isolate and thwart. Also, when compared to the AMD computation of the hidden classifier design of Section 5.5.1, the *Predict - Detect* classifier's disagreement computation provides a higher indication of adversarial activity, as shown in Figure 5.19. This figure demonstrates the increased information captured in the disagreement between the two models, and emphasizes the advantages of using the *Predict-Detect* design as an implementation strategy of hidden feature importance paradigm

5.5.3 Benefits of the *Predict - Detect* classifier design

Using a hidden model allows for several advantages in a dynamic adversarial environment. As seen in the previous section, the hidden model allows for better detect-ability of attacks, with increased number of samples falling within the adversarial margin. Addi-

TABLE 5.8

Training Accuracy, Effective attack rate (EAR), Adversarial Margin Density (AMD) and Disagreement metrics under AP-HC attacks, for defender using the *Predict - Detect* Classifier and one that uses all features.

Dataset	Training Accuracy		EAR		AMD - All Features	Disagreement-*Predict-Detect* Classifier
	All Features	Predict-Detect Classifier	All Features	Predict-Detect Classifier		
Synthetic	100	100	1	0.99	0.01	0.172
CAPTCHA	100	99.3	0.99	0.99	0.002	0.075
Phishing	93.1	90.5	0.99	0.99	0.19	0.562
Digits08	97.1	93.4	0.99	0.99	0.22	0.648
Digits17	99.5	94.2	0.98	0.99	0.16	0.517

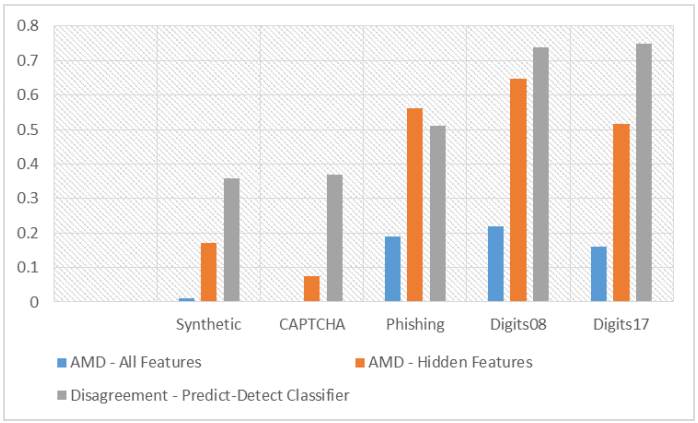


Figure 5.19: Comparison of AMD values for defender using all features, defender with 50% of features hidden from prediction task, and disagreement values for the *Predict-Detect* classifier.

tionally, the lack of complete information by an adversary ensures that only partial reverse engineering is possible, and that the training data is not completely corrupted by overlapping attack samples. By preventing training data nullification, this design ensures that data privacy is ensured, and that the same set of features can be used for retraining after attacks occur. This can be seen from the Table 5.9, which demonstrates the effect of the AP-HC attacks on the *Predict-Detect* classifier. While the *Prediction model* is evaded due to the attacks, the *Detection* model is still relatively unaffected. This demonstrates the inability of the adversary to launch a high confidence attack, which evades a majority of the informative features of the data, simultaneously. The detection model features, not evaded by the adversary in this cycle, can now be used to train additional models for protection against

TABLE 5.9

Effective Attack Rate (EAR) of the *Prediction* and the *Detection* models, after AP-HC attack.

Dataset	EAR	
	Prediction Model	Detection Model
Synthetic	0.99	0.67
CAPTCHA	0.99	0.65
Phishing	0.99	0.47
Digits08	0.99	0.32
Digits17	0.99	0.32

the current attack cycle. Also, the already trained *Detection model*, can be immediately deployed to serve as a temporary replacement model for the vulnerable black box model, while additional labeled information is gathered to retrain the classifier.

The disagreement between the hidden model and the prediction model can also be used for active learning, to selectively label only a few samples to determine the new attack characteristics. Since a majority of the samples falling in the adversarial margin are a result of an adversary, labeling them will ensure high informativeness in semi supervised settings. This is a more efficient strategy than random labeling of samples. In an adversarial domain this is especially important, as attacks are generally a minority class, and identifying and labeling them can be costly and inefficient. By naturally defining regions where adversarial activity is expected, the *Predict-Detect* classifier allows for effective allocation of labeling budget for detecting and verifying attacks.

The essence of the *Predict-Detect* approach is to have an information leverage over the adversary at all times. This leads to misleading of the adversary, who even though aware of the presence of the features in the data, is unable to probe and ascertain their importance. Even if an adversary is aware of the hidden model strategy, it cannot probe the model to try and mimic these features, as no feedback is presented on those set of features, by the prediction black box classifier model. The generic design of the framework leaves it open for extension and usage with other security mechanisms. The robustness and

randomization strategies can be applied to the features of the prediction model, to increase the effort required by an adversary, to evade it. As long as a set of informative features are hidden from the black box model, the attacks can be detected and recovered from.

5.6 Chapter summary

In this chapter, the effect of different classifier design strategies, on the ability to provide long term security benefits, is analyzed. In particular, we evaluate the shortcomings of existing classifier security designs of Restrictive one-class classifiers, Complex learning methodologies and Randomized classifiers. All these strategies fail to account for dynamic aspects of the system, as they lead to limited defender options, post attacks. The reason for their vulnerability, is the fundamental idea behind integrating more information into the learned model. Although this strategy works in the case of targeted exploratory attacks, they do not work against indiscriminate probing based attacks, which are common on black box classifier models. We propose a classifier design counter-intuitive to the ideas of robustness in existing works, and demonstrate that using a reduced feature space is more beneficial to maintaining adversarial uncertainty over the training data. This uncertainty on the adversary's part, is essential to allow for easy detection and retraining from attacks. The proposed *Predict-Detect* classifier design is able to detect adversarial activity and allows for recovery, even when faced with an intelligent and determined adversary. It does so by misrepresenting feature importance, to prevent incessant probing from learning the complete internal state of the system. The following claims were evaluated in this chapter:

- ***Claim a:*** *Restrictive one-class classifiers, Complex Learning and Randomization, are not effective as long term security measures.*

  The notion of adversarial uncertainty demonstrates that the above mentioned classifiers lead to attacks of high confidence, which makes their dynamic handling difficult. They cause problems of undetect-ability and can cause data nullification in the extreme case. The developed *Anchor Points High Confidence (AP-HC)* filter attacks,

shows that adversary can generate high confidence attacks against robust classifiers, without any additional information about the black box system.

- ***Claim b:*** *Unsupervised Drift Detection (particularly MD3), can be actively evaded by an adversary at test time.*

  Unsupervised margin based drift detection relies on the ability to capture changes which affect only a few important features of the data space (Chapter 4). However, the AP-HC attacks demonstrates that, in the presence of multiple orthogonal information, the adversary can leverage all of this information, by probing the system and training robust classifiers from the probes. By filtering out low confidence exploration samples, the adversary can cause exploitation which falls outside the critical margins of the defender, leading to the reduction of the margin density based detection capabilities. As such, an adversarial agnostic view of the problem can lead to evasion of the drift detection capabilities of the defender.

- ***Claim c:*** *Using a reduced feature set in the defender's model can improve the detectability of attacks and prevents information leakage to the adversary.*

  The proposed *Predict - Detect* classifier, was shown to provide defense against the AP-HC attacks, as they prevent the reduction in adversarial uncertainty. When tested on 5 datasets, the adversarial margin density was found to be significantly higher for the proposed model (Δ=0.43, on average, when compared to robust model using all features). The Effective Attack Rate (EAR) of the Detection model was also found to be 50.6% lower than the Prediction model, indicating adversarial uncertainty over the hidden features and, by extension, the ability of the defender to train future classifier models, using the new attack data.

The *Predict - Detect* classifier was shown to provide benefits of unsupervised adversarial detection, data nullification prevention, and was also shown to be capable of providing benefits for retraining. We will use this motivation and analysis, in developing a streaming algorithm capable of dealing with adversarial drift. By extending the idea of adversarial

awareness, to margin based drift detection of Chapter 4, we can develop a model capable of reacting to adversarial drift, in an efficient and timely manner, while still retaining the labeling efficiency and reliability, of the drift detection mechanism. Also, experimental evaluation of the active learning benefits of the *Predict-Detect* classifier, when faced with a minority attack class, needs to be evaluated.

CHAPTER 6

HANDLING ADVERSARIAL CONCEPT DRIFT - The *PREDICT - DETECT* STREAMING CLASSIFICATION FRAMEWORK

In this chapter, we develop an adversary-aware drift detection and handling system, for streaming data environments. The SEE framework of Chapter 3 is extend for simulating data streams with adversarial drifts. The need for an adversarial aware design is emphasized, by analyzing the margin density drift detection (MD3) algorithm of Chapter 4, on adversarial drifting data streams. The idea of feature information hiding from Chapter 5, in conjunction with the reliability of the MD3 drift detection methodology, is used to develop the proposed streaming data framework. The proposed framework demonstrates the advantages of an adversarial aware drift handling system, setting it apart from traditional concept drift handling methodologies.

6.1 Adversarial concept drift

Data in dynamic real world environments is characterized by non-stationarity. The changes in the distribution of the data, called concept drift, can cause the learned model to drop in predictive performance, over time. It is therefore essential to detect and handle drifts swiftly, to continue using the predictive capabilities of the model. Adversarial drift is a special kind of concept drift, where the changes in the data distribution is targeted towards changing the characteristics of one class of samples (i.e., the *Malicious* class). The adversary starts by learning the behavior of the defender's classifier model, using crafted probes, and then exploits this information to generate attack samples, to evade classification. These attacks leads to a change in the distribution of the data and also leads to a drop in the prediction capabilities of the defender's model. From the perspective of streaming data

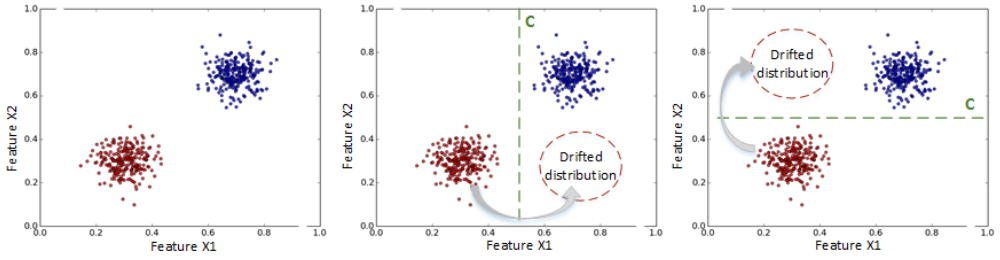


Figure 6.1: Illustration of Adversarial Drift, as a function of the defender's classifier model C.

mining, we refer to such changes in the data distribution at test time, as *Adversarial Drifts*. The main characteristics of adversarial drift which distinguishes it from traditional concept drift are: a) The drift is a result of changes to the malicious class samples only, b) The drift is a function of the deployed classifier model, as the adversary learns and gains information about it, before trying to evade it, and c) The drift is always targeted towards subverting the deployed classifier (i.e., it is relevant only if it leads to a drop in the performance of the deployed model). The dependent nature of adversarial drift is shown in Figure 6.1, where the deployed classifier C, dictates the possibility of adversarial drifts in the data space. The figures b) and c) demonstrate adversarial drifts, which are caused by an attacker trying to subvert C. The two scenarios are a result of the different defender models, which the adversary is trying to learn and circumvent. The nature of the drift is dependent on the choice for C, and as such the defender has a certain degree of control over the possible space of drifts, at test time.

The detection of drifts is often carried out by supervised approaches [63], which continuously monitor the predictive performance of the stream of data, flowing into the system. However, this is not a practical solution, as human expertise in the form of labeled data, is often expensive and time taking to obtain. There have been proposed unsupervised drift detection methodologies [102, 156–158, 170, 174], which directly monitor the feature distribution of the data, to indicate drifts. These approaches suffer from excessive pessimism, as they cannot differentiate between drifts which affect the classifier's performance and those

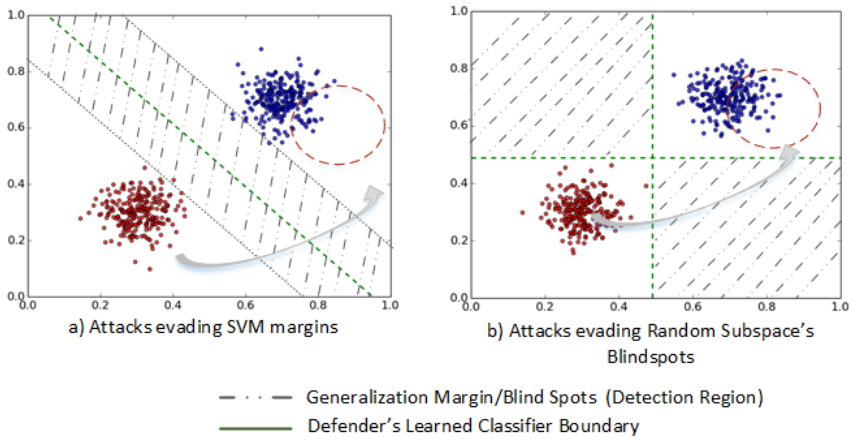


Figure 6.2: Drifts in adversarial environments will avoid low confidence regions, to avoid detection.

which do not. This leads to false alarms and the unreliability of unsupervised drift detection methodologies. The Margin Density Drift Detection (MD3) approach of Chapter 4, provides reliability in the unsupervised detection of drift, by explicitly tracking the distribution of the samples in the uncertain regions of a robust classifier. MD3 tracks the density of the samples which are found to be uncertain by the deployed model, to determine if a significant drift has occurred. MD3 was shown to be a domain independent, model independent and reliable indicator of drift. MD3's reliability in unsupervised drift detection, stems from its inclusion of the deployed classifier, in the drift detection process. However, the domain agnostic nature of the MD3 approach causes it to disregard the specific nature and characteristics of the drifts. In adversarial domains, the drift is characterized by an attacker continuously trying to hide its trail, by learning about the behavior of the detection system first. As such, the MD3 can itself be vulnerable to adversarial evasion at test time.

The MD3 was designed as an adversarial agnostic approach, where it considers drift to be independent of the deployed classifier. In an adversarial domain, the relation between the type of drift and the choice of the classifier C, is strongly coupled. An adversary aware of the presence of a drift detection scheme, can design attacks of high confidence (as shown in Section 5.3.4), to avoid detection. This is illustrated in Figure 6.2, where attacks intentionally avoid low confidence areas of the feature space. The MD3 relies on

tracking samples in the SVM margins (Figure 6.2a)), or in the blindspots of feature bagged ensembles (Figure 6.2b)). An adversary realizing this, can launch high confidence attacks, which are designed to fall outside the uncertain regions of space. This is possible even when the feedback received from the black box classifier (C), is a binary *Accept/Reject* feedback. By ignoring the adversarial nature of the drift, the MD3 is left vulnerable to such drifts in the data distribution.

An adversarial aware unsupervised drift detection approach, will take preemptive steps at the design of the classifier model, to ensure that future detection and retraining is made easier. In this chapter, one such framework is proposed: the *Predict-Detect* framework. This framework misrepresents feature importance information to an adversary, to mislead adversaries into getting detected at test time. The ability to use attack foresight and integrate it into a preemptive design, provides long term benefits for reactive security. In this chapter, the design of the *Predict-Detect* framework, for dealing with adversarial concept drift in streaming data, is presented. Specifically, the following claims are evaluated in this chapter: a) Adversarial agnostic drift detection is vulnerable to evasion, b) The design of the classification model at training time, can provide benefits for handling test time drifts, and c) Hiding feature importance, can provide long term security benefits in terms of detection and recovering from attacks.

The rest of the chapter is organized as follows: Section 6.2 presents the proposed *Predict-Detect* classifier framework, for handling adversarial drifts. Section 6.3 presents a framework for simulating adversarial drift on black box classifiers, from real world datasets. Experimental analysis of the framework is presented in Section 6.4. Extending the framework, to active learning over imbalanced drifting streams is presented in Section 6.5. Section 6.6 presents additional suggestions and guidelines for system designers, to use and extend the proposed *Predict-Detect* framework. Summary of important results and takeaways is presented in Section 6.7.

6.2 Proposed *Predict-Detect* framework for classification in *Dynamic-Adversarial* environments

The proposed *Predict-Detect* framework is developed as a streaming incremental framework for detecting and handling adversarial drifts. The overview of the framework is presented in Section 6.2.1. Detailed design of the framework and its major components, is presented in Section 6.2.2.

6.2.1 The streaming *Predict-Detect* classification framework

The nature of adversarial drifts, makes it dependent on the characteristics of the deployed classifier model, which it is trying to evade. As such, preemptive measures taken during the training of a classifier model can benefit dynamic test time detection and handling of such drifts. The *Predict-Detect* framework uses this intuition to develop classifier models which are able to detect adversarial activity at test time; reliably and with limited labeled data.

The overview of the *Predict-Detect* streaming framework is presented in Figure 6.3. The input stream of unlabeled data samples X is processed, leading to the output predicted label stream Y. In this process, the framework has an unsupervised drift indication component, which processes X to detect if there is any significant drifts which could cause the predictive performance to drop. Upon signaling a drift, the framework request additional labels (N_{Labels}), from an external oracle, often at a price (financial and time resources). These labeled samples are used for confirmation of the earlier signaled drift (by the unsupervised component). If a significant drop in predictive performance (accuracy/f-measure) is seen over the labeled samples, the framework confirms the occurrence of an adversarial drift. In the event of a confirmed drift, the labeled samples are used to retrain the predictive model, to ensure the continued efficacy of predictive performance of the system. Since the labeled samples are requested only when a drift is first suspected, using unlabeled data, this framework prevents wastage of labeling effort, which results from continuous monitoring of the stream. In the event of infrequent drifts and long periods of stationarity, the framework

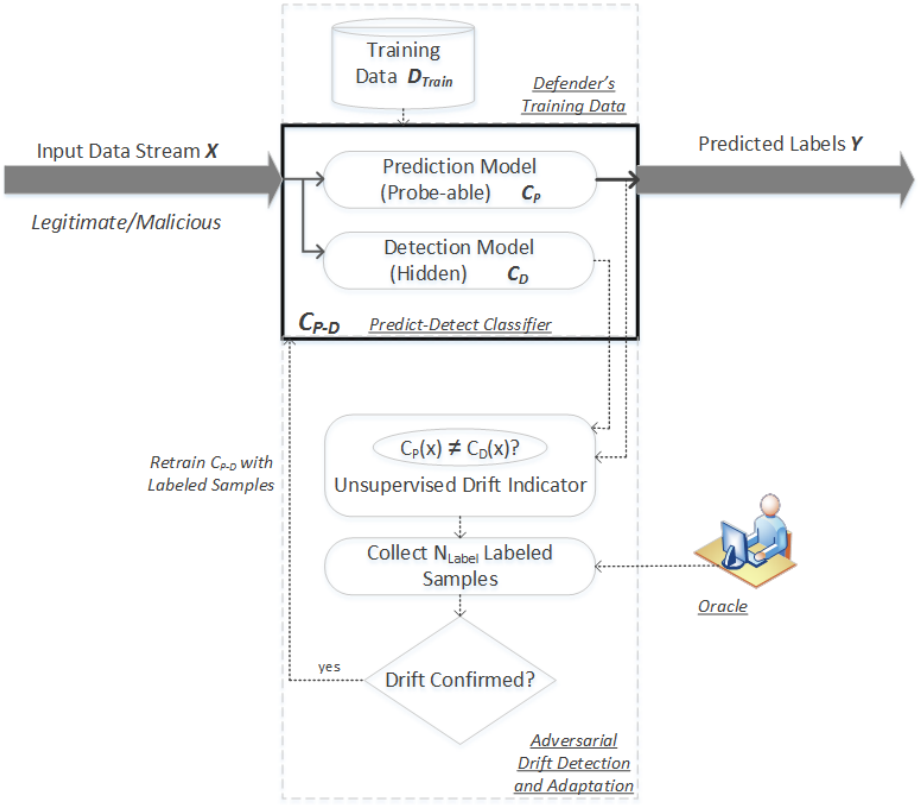


Figure 6.3: Overview of the *Predict-Detect* framework.

will not indicate drifts, and thereby no labeling will be wasted.

The core design aspect of the framework, which makes it effective in an adversarial environment, is the development of the *Predict-Detect* classifier model, as seen in Figure 6.3 as the C_{P-D} model. The design is motivated by our analysis in Chapter 5, where it was seen that overly informative models result in attacks which are harder to detect and recover from. As such, the design of the *Predict-Detect* classifier was proposed, as a novel mechanism to hide feature importance information from the adversary. The classifier relies on a coupled model strategy, where two orthogonal models are trained from the training data D_{Train}. The first model is called the *Prediction* model (C_P), and it is trained on one disjoint subset of features of the original training dataset. The other subset of features, not used in (C_P), is used to train the *Detection* model (C_D). The *Prediction* model is the defacto model of the framework, used for classifying the input unlabeled stream X, to produce output labels Y. Since this is the prediction model facing the input stream of data, this model is susceptible

to adversarial activity, at test time. This forward facing model is expected to be attacked, by an adversary using probing based exploratory attacks and observing the feedback provided by the framework. As such, the model's performance is expected to drop over time. The *Detection* model is shielded from the probes of the adversary, as it is kept hidden and away from the prediction process. As seen in Figure 6.3, the feedback of the model C_D is not presented to the outside world. This model is used for the detection of adversarial activity. An adversary, using probes to understand the behavior of the black box model C_P, will fail to successfully reverse engineer C_D. As such, an increase in the disagreement between the predictions of the two models, will be indicative of adversarial activity. The adversary is oblivious to the internal split of the two classifiers, as the framework provides a unified black box representation to the outside world. Input data stream X is split between the two models, vertically based on the features used to train the respective model, and the prediction by C_P, is presented as the output of the framework. No amount of probing will enable the adversary to understand the significance of features held by C_D, as they are not a part of the prediction process of the framework. The misrepresentation of these feature importance, in an adversarial domain, is the core of what makes the framework effective.

6.2.2 Design and major components of the *Predict-Detect* framework

The design of various components of the framework is presented in this section. The development of the *Predict-Detect* classifier model (C_{P-D}), from the training data D_{Train}, is presented in Section 6.2.2.1. Section 6.2.2.2 presents the development of the unsupervised drift detection algorithm, which uses the disagreement between the two models, as an indication for adversarial drift. Design strategies for the regeneration of the C_{P-D} models, after a drift is confirmed, is presented in Section 6.2.2.3.

6.2.2.1 Generating the *Prediction* and the *Detection* models from the training data

The training dataset containing F features, is divided into two subsets (vertically based on features), to train the *Prediction* and the *Detection* models. Experimentation in Chapter 5, has shown that such splits are feasible without significant compromise on the predictive performance, in cybersecurity domains. This is due to the presence of multi-modal and orthogonal informative features, in high dimensional datasets. We intend to define a division of the feature space, such that each of the trained models has high predictive performance. Such divisions are naturally defined when the training dataset is aggregated from multiple sources, with clearly defined boundaries. An example of such a system is a multi-modal biometric system [127], which uses both face recognition and fingerprint scanners, to provide the final authentication. In such a system, it is easy to split the features into two disjoint subsets, one for the face recognition and the other for fingerprint data.

In the absence of domain specific knowledge of the features, random partitioning is usually resorted to. However, this is not optimal, as a majority of the informative features can end up clumped together in the same partition. We propose a feature ranking based approach, which considers dividing the features uniformly based on their importance to the classification task. We use feature ranking to rank the initial set of F features, based on the training data. We then distribute the features in a round robin fashion, to form the two subsets: F_P, used to train the *Prediction* model, and F_D, for training the *Detection* model. While several feature ranking approaches are available, the F-value from ANOVA is considered here for experimentation purposes, as the methodology for measuring feature importance[1], without loss of generality.

The splitting of features and generation of orthogonal trained models, is done according to Algorithm 6.1. The set of F features $(1..k)$, is divided to form multiple disjoint

[1]Using scikit-learn's [149] *sklearn.feature_selection.SelectKBest* function to score all features using *sklearn.feature_selection.f_classif*.

Algorithm 6.1: Generating multiple models from training data, by splitting features.

Input : Training data (D_{Train}) with features $F(1..k)$, Number of splits N (For the C_{P-D} classifier, N=2.)
Output: Trained models $M_{Trained}$

```
1  F_Splits = [[]]
2  M_Trained = []
3  F_sort_importance = RankByFeatureImportance(F)
4  for i = 1 .. N do
5      F_Splits[i%N].add(F_sort_importance[i])
6                                   ▷ Round robin split of ranked features
7  for i = 1 .. N do
8      D_Train_split = D_Train
9      for feature in F do
10         if feature not in F_Splits[i] then
11             Blank out D_Train_split, by replacing with default value
12                                   ▷ 'blanking-out' non associated features
13     M_Trained[i] ← Train model on D_Train_split
14 return M_Trained
```

subsets, based on their informativeness to the prediction task, obtained from the training data. For the *Predict-Detect* classifier, two subsets are needed: F_P for the *Prediction* model (C_P), and F_D for the *Detection* model (C_D) ($F = F_P \cup F_D$). As such, each model has an associated subset of features, which it deems important. The original training data is modified, such that only the associated features are used in training a model. This is done by the *'blank-out'* process, given in Line 10. This process takes the original training dataset, and replaces all non model associated features with a predefined default value. An example of this process is depicted in Figure 6.4. The original set of 5 features is divided into 2 disjoint subsets. The *Prediction* model is associated with Features 1, 3 and 5 ($F_P = \{1, 3, 5\}$). As such these features are retained in the training data and the remaining are filled with default values, for all data samples. This process nullifies the discriminatory information of the Features 2 and 4, thereby barring them from being included in the *Prediction* model. Similarly, for the *Detection* model, Features 2 and 4 are included in the model ($F_D = \{2, 4\}$), while Features 1,3 and 5 are blanked out. Once the two models are trained on the modified training data, they can each receive unlabeled stream samples of $|F|$ features. The ability

Training Data Features 1-5

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
2.2	300	'chevrolet'	'red'	2017
3.2	200	'ford'	'blue'	2013
4.2	250	'ford'	'green'	2012
1.2	350	'bmw'	'black'	2005

Prediction model- Selected Features 1,3,5

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
2.2	0	'chevrolet'	''	2017
3.2	0	'ford'	''	2013
4.2	0	'ford'	''	2012
1.2	0	'bmw'	''	2005

Detection model- Selected Features 2,4

Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
0.0	300	''	'red'	0
0.0	200	''	'blue'	0
0.0	250	''	'green'	0
0.0	350	''	'black'	0

Figure 6.4: Illustration of feature splitting between the *Prediction* and the *Detection* model. The blanked out features of each model are highlighted.

to disregard features is built into the training of the two models. The advantage of this *'blank-out'* process, as opposed to splitting each incoming sample into two subsets based on features, is that now each of these models operate on the entire feature space F and can receive the same input sample X. The training of the models has equipped it with the ability to assign importance to their required feature set only. The Algorithm 6.1 is general in its presentation, as it allows the training of N disjoint models. For the *Predict-Detect* classifier, we take N=2, and use one of the trained models for prediction, while the other for detection.

In the absence of domain specific knowledge and correlation information, we can ensure that important features are evenly distributed among the two sets of models, using Algorithm 6.1. In real world systems, where correlation between features can cause attributes to change in tandem, more sophisticated feature splitting techniques can be employed. The feature subset ensemble techniques of [192], combined with the cluster based feature splitting of [193], could be used to form multiple classifiers with uncorrelated features within them. For the effective usage of the *Predict-Detect* framework, we only need to ensure that the generated splits of features are disjoint and result in good predictive

performance for each of the models.

6.2.2.2 Detecting adversarial drift from unlabeled data

The proposed framework relies on detection of drifts from the unlabeled data stream, to save expenditure of labeling budget on validation of the prediction model. This is made possible by the setup of the *Prediction* and the *Detection* models. An increased disagreement in the predictions of the two models, on new incoming samples, is suspicious and indicates a possible drift. The tracking of this disagreement, over time, in a streaming environment is used for adversarial drift detection. This is presented in Algorithm 6.2, which is motivated in setup by the margin density drift detection algorithm (MD3) of Chapter 4. The MD3 algorithm tracks the number of samples falling in a robust classifier's margin, in a streaming environment, to indicate the possibility of a drift. The *Predict-Detect* provides an adversarial aware drift detection mechanism over unsupervised streams, by causing attacks to be detected, based on disagreement with the hidden classifier *Detection* model.

The unsupervised drift detection mechanism is presented in Algorithm 6.2. The prediction and the detection models (C_P and C_D), generated from the initial training data, are used for the detection of drifts from the unlabeled data stream X. Also, the training data is used to learn the expected disagreement and acceptable deviation (PD_{Ref} and σ_{Ref}), along with the expected prediction performance (measured in accuracy for balanced streams), given by Acc_{Ref} and σ_{Acc}. This information is learned via 10-fold cross validation, and is used to characterize the normal behavior of the stream. This is done by dividing the training data into 10 bands, and generating the C_P and C_D models on 9 bands at a time, and then computing the disagreement of the 10^{th} band. This value is obtained 10 times, and the reference distribution is learned from it. This serves as a basis for establishing normal expected behavior, from which drift can be detected based on deviation. Sensitivity is controlled using the parameter θ, which provides a user friendly way to specify acceptable deviation of the stream, in terms of reference characteristics learned from the initial training data. As such, the framework allows specifying data independent parameters, making it

Algorithm 6.2: Unsupervised drift detection in the *Predict-Detect* framework.

Input : Unlabeled stream X, Predict - Detect models C_P, C_D, Reference distribution (PD_{Ref}, σ_{Ref}, Acc_{Ref}, σ_{Acc}). *Parameters:* Sensitivity θ, Stream progression $\lambda = (N-1)/N$(where N is the chunk size), N_{train}(= N by default), $N_{unlabeled}$(= N by default)

Output: Predicted label stream Y

1 $PD_0 = PD_{Ref}$

2 currently_ drifting = False

3 **for** *t= 1,2,3,...:* **do**

4 Compute disagreement score - $Dis(x = X_t) = \begin{cases} 1, & if \quad C_P(x) != C_D(x) \\ 0, & otherwise \end{cases}$

5 Update $PD_t = \lambda * PD_{t-1} + (1-\lambda) * Dis(X_t)$

6 **if** $|PD_t - PD_{Ref}| > \theta * \sigma_{Ref}$ *and not currently_drifting* **then**

7 currently_drifting = True ▷ **Drift Suspected**

8 Collected_unlabeled_samples=0

9 $D_{Unlabeled} = \emptyset$

10 $D_{Labeled} = \emptyset$

11 **if** *currently_ drifting and Collected_unlabeled_samples* $< N_{unlabeled}$ **then**

12 $D_{Unlabeled} \cup X_t$ ▷ **Collect samples to be labeled**

13 Collected_unlabeled_samples ++

14 **else if** *currently_drifting* **then**

15 ▷ Enough samples to make decision

16 $D_{Labeled}$ ← Label samples from $D_{Unlabeled}$, using *Oracle*, upto N_{train}

17 ▷ **Active learning** can be used in case $|D_{Unlabeled}| > N_{train}$

18 **if** $(Acc_{Ref} - Acc_{D_{Labeled}}) > \theta * \sigma_{Acc}$ **then**

19 Retrain C_P, C_D with $D_{Labeled}$

20 ▷ **Drift Confirmed**

21 Update Reference distribution (MD_{Ref}, σ_{Ref}, Acc_{Ref}, σ_{Ref})

22 currently_ drifting = False

23 **return** $C_P(x)$

intuitive to the end user.

For each incoming samples x, the disagreement in predictions between the C_P and the C_D, is computed as the signal Dis. This disagreement is aggregated over time, to form the PD_t metric, as computed in Line 5. The computation uses a time decaying incremental tracking of the metric PD, dictated by the chunk size N. A sudden increase in the disagreement metric PD, is indicative of an adversarial drift. This indication is controlled by the sensitivity parameter θ, specified by the user based on its tolerance for deviation. The initial indication of drift, given by Line 6, is the unsupervised drift indicator

component of the framework. It goads the system administrator to examine the data for potential attacks. Attacks are confirmed by collecting $N_{Unlabeled}$ samples, and labeling them to form the labeled dataset ($D_{Labeled}$). This $D_{Labeled}$ set of labeled samples is used to confirm drifts, and to retrain the models of the framework in case a drift is confirmed. A drift is confirmed if the predictive performance (measured as accuracy here) on the labeled samples is seen to fall significantly. This fall is measured in terms of deviation from the reference accuracy (Acc_{Ref}), which was obtained from the training data.

Once a drift is confirmed by testing the predictive performance on the obtained labeled data, the models (*Prediction* and *Detection*) need to be retrained, to represent the new distribution of the stream. Relearning is performed using the obtained labeled samples $D_{Labeled}$. Additional samples may be requested by an application, by querying the Oracle. Querying the Oracle is expensive, as it requires time/effort to obtain expert feedback on the unlabeled samples. In Algorithm 6.2, the labeling of samples is seen to follow a naive strategy, where the N subsequent samples after a drift indication are requested to be labeled by the Oracle, to form the labeled set $D_{Labeled}$. However, this can be extended to integrate active learning methodologies into the framework. In case of $N_{Unlabeled} > N_{train}$, active learning methodologies can be employed to effectively choose the $D_{Labeled}$ set of samples. The exact mechanism for retraining and labeling of collected samples is not discussed in Algorithm 6.2. This has been done intentionally, to allow for a generic presentation the algorithm, which allows for ease of extension with other active learning strategies and retraining policies.

The entire drift handling process is kept internal to the black box system, and the end-user/adversary is agnostic to the adaptive mechanism of the framework. The end-user/adversary is provided prediction on the input samples x, using the prediction model $C_P(x)$. Since additional labels are only requested when an attack is suspected, the framework works in an unsupervised manner for the majority of the stream, without the need for constant labeled validation. Only when drifts are suspected and retraining may be needed, are labeled samples requested. This makes the framework attractive for usage in dynamic

adversarial drifting environments, where labeling is expensive and time taking.

6.2.2.3 Retraining the *Predict-Detect* models - Drift Recovery

In order to recover from the effects of an adversarial drift, the classifier models of the framework need to be retrained, using the obtained labeled data. The following strategies and their impact on the dynamic learning process are discussed, as potential retraining options.

- *Using the existing feature split (**PD-NoShuffle**)*: In this strategy, the existing feature splits of the *Prediction* and the *Detection* models, are retained. The models are trained on the new labeled data, based on the same set of features already assigned to them. This strategy has the advantage to keep features in the *Detection* model hidden from an adversary, throughout the progression of the stream. However, this strategy does not account for the changes in the feature ranking, and can lead to poorly trained classifier models, which are a result of changes in the importance of features over time, following drifts in data.

- *Re-splitting features (**PD-Shuffle**)*: Here, the framework uses the labeled data to re-generate feature splits based on Algorithm 6.1. The newly split models are then used for deploying the *Prediction* and the *Detection* models. This strategy assumes every drift recovery to be an independent start of a new attack-defense cycle. This is especially true when a system is faced with multiple independent adversaries, over time. This strategy is assumed to provide better trained models than the *PD-NoShuffle* , as it has the opportunity to re-calibrate feature ranking and generate new splits of the features, after every attack cycle.

While both strategies have their shortcomings and advantages, a defender can also resort to a combined approach while applying this framework. First the defender can check to see if it is able to use the existing feature splits to retrain the model and receive sufficient predictive performance, using cross validation on the training dataset. If the features are

rendered unusable due to a sophisticated attack, the features can be re-split. Delaying the shuffling of features is advantageous, as shuffling transfers information about the hidden features onto the prediction forefront. An adversary could use this information, over time, to generate a high confidence attack, by aggregating information from multiple cycles of attacks. While we have not found any documentation for such a category of attacks, it is a possibility and as such we should avoid shuffling of features whenever possible. In either case, the adversary is devoid of complete information, which makes it unable to launch a high confidence attack or a data nullification attack [114]. This ensures that adversarial activity will be detected and that retraining will be possible, for continued operation in streaming domains. By constantly gaming the adversary and responding quickly to attacks, the attacks are rendered expensive and futile, making this reactive system an attractive defense strategy for securing against adversarial data drifts.

6.3 Generating adversarial concept drift on real world datasets - Extending the *Seed-Explore-Exploit* framework to streaming domain (*SEE-Stream*)

Concept drift refers to the change in the distribution of data over time. Adversarial concept drift is a special type of concept drift, as the data distribution changes are introduced by an adversary aiming to subvert the performance of the deployed classifier. As such, these distribution changes are dependent on the classifier trained and deployed by the defender. An adversary begins the attack cycle, by probing the deployed black box model of the defender, and then uses this information to generate samples which evade detection. This characteristic of adversarial drift makes it a special category of concept drifts, which needs to be analyzed and dealt with differently, than regular concept drift. Adversarial drift is a function of the deployed classifier and as such is directed by the design of the deployed classifier. It is not possible to evaluate these drifts on existing datasets (which are popularly used for concept drift research), as these drifts are specific to the type of classifier models, which they are trying to evade. We present here, a strategy to simulate adversarial drifts using real world datasets. We do this by extending the *Seed-Explore-Exploit* (SEE) attack

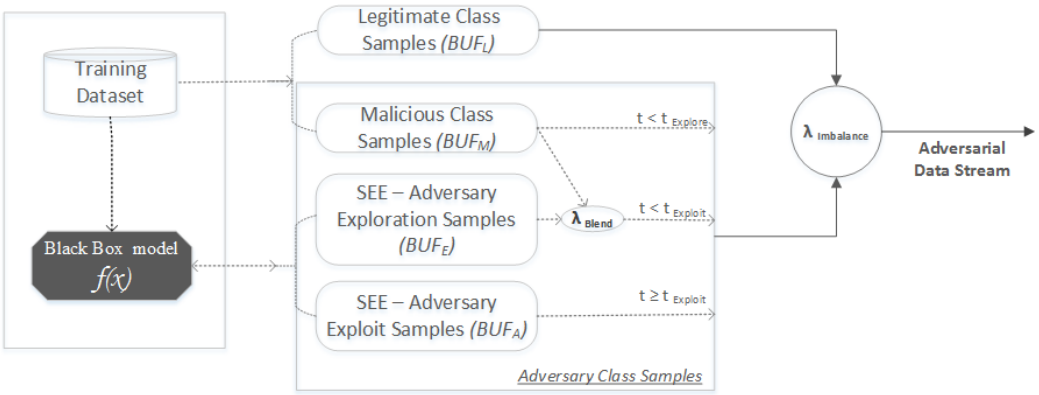


Figure 6.5: The SEE-Stream framework for simulating adversarial concept drift.

framework of Chapter 3. The SEE framework was developed for simulating data driven exploratory attacks, on black box classifier models. The framework was developed for static evaluation, as the attack samples were only evaluated for the purposes of demonstrating the possibility of classifier evasion, by an adversary. We extend the SEE framework, to be used in a streaming environment, under the *SEE-Stream* framework, presented here.

The *SEE-Stream* framework is presented in Figure 6.5. The framework provides a wrapper over the data driven attack framework of SEE. As such, it operates under the same assumptions of a black box defenders model, where the only access to the system's model is via submitting probes and observing the feedback on them. The *SEE-Stream* framework receives samples from the attack simulation on the defender's black box, and it converts these samples into a stream of data for adversarial analysis. The initial training data, from the real world dataset, is split into two parts: the *Legitimate* class samples and the *Malicious* class samples. These splits are used to form the initial distribution of the stream, before the drift starts. The training dataset is also used to train the defender's model ($f(x)$). The defender's model is then attacked using the SEE framework, and the resulting exploration and exploitation samples are stored in the corresponding buffers (BUF_E and BUF_A, respectively), as shown in Figure 6.5. These are results obtained from static evaluation of the data, and we will use this data to generate a stream of adversarial samples.

Data stream samples are generated by random sampling from the 4 different data buffers of Figure 6.5. At any given time, the legitimate class samples are obtained from

sampling the buffer BUF_L, and the adversary's samples are sampled from one of BUF_M, BUF_E or BUF_A. The parameter $\lambda_{Imbalance}$ is used to control the amount of imbalance between the legitimate and adversarial class samples, in the stream. Since we assume that the legitimate class does not drift over the course of the stream, we draw the regular class samples by always sampling from the initial pool of legitimate class data (BUF_L), obtained from the training dataset. However, for the adversarial class samples, the samples are drawn from the three different buffers, over the course of the stream. Initially, till time $t_{Explore}$, samples are drawn from the original pool of the malicious class training samples (BUF_M). This is the time period where an adversary has not yet started the attack process. After $t_{Explore}$, we draw samples from both the Malicious class training samples (BUF_M) and the pool of exploration samples (BUF_E). The rate at which the samples are drawn from each of these buffers is controlled by the parameter λ_{Blend}, called the blending rate. By blending exploration samples with the original set of malicious samples, an adversary avoids detection in the exploration phase. Consequently, it takes a long time for an adversary to obtain the required number of exploration samples, but as is assumed, the adversary wants to avoid detection at this stage. After the adversary obtains enough information about the black box model (upto the exploration budget $B_{Explore}$), it starts the exploitation phase, which is the attack payload for this adversarial cycle. This is done after time $t_{Exploit}$, by sampling from the pool of exploitation samples (BUF_A). The time values $t_{Exploit}$ and $t_{Explore}$, enable the user to set up the profile of the stream, that they want to simulate. It provides greater flexibility about length of stream and location of the adversarial drift, for more fine-tuned analysis. It should be noted that this proposed framework serves only the purpose of simulating an adversarial drifting stream, as all the exploration and exploitation samples are generated statically using the SEE framework. Nevertheless, this provides a simple and effective wrapper, to convert existing datasets to data streams exhibiting adversarial concept drift. After the completion of an attack cycle, the same framework can be used to generate additional attacks, by regenerating the data buffers with samples from a new launch of the SEE framework, on the retrained defender's model.

6.3.1 Simulating adversarial concept drift on the *phishing* dataset

In this section, we demonstrate the use of the *SEE-Stream* framework, for simulating adversarial concept drift on the *phishing* dataset [145]. The dataset is pre-processed by normalizing it to the range of [0,1]. The defender uses a random subspace ensemble (50 Linear SVM models, 50% features per model), to train a model on the dataset. Anchor points attacks of the SEE framework (Section 3.3.2), is used to generate adversarial samples on the defender's model.

The simulation of adversarial drift on the *phishing* dataset, is shown in Figure 6.6. The stream demonstrates an exploration phase till $t_{Explore}$=10,000 samples, and the attack phase starts at $t_{Exploit}$=30,000 samples. The data is taken to be balanced ($\lambda_{Imbalance} = 0.5$) and the blend rate is taken to be $\lambda_{Blend} = 0.05$ (i.e., 5% of the sample in the exploration phase are drawn from the exploration buffer). From the figure, the effects of adversarial drift can be seen starting at t=30,000, as the accuracy starts to rapidly drop. This is a result of the adversarial manipulation of samples, to evade the deployed black box model. In this sense, this generated adversarial drift is different from a traditional concept drift, as this drift is dependent on the type of model deployed originally. In order for a model to be usable, it is necessary to detect and fix the effects of this adversarial drift. The detection of drifts, the learning of additional new information, and the retraining of the defender's model, are the main aspects of consideration in designing reactive security measures for adversarial drift. The drop in accuracy at the exploration phase is minimal, and can go undetected, as is intended by the adversary. In case of an overly sensitive detection system, the adversary over time can learn to use a lower blend ratio, to cover its tracks.

The effect of using different blend rates λ_{Blend} is shown in Figure 6.7. In the simulation of Figure 6.7, the exploration continues till 1000 samples are obtained at the specified blending rate. Smaller blend ratios, lead to longer exploration time periods, causing the attack phase to get delayed. However, a higher ratio causes a drastic fall in the accuracy and causes the defender to become cautious and thwart the attacks even before the start. In case of the blend ratio of 0.05, it is seen that it takes close to 40,000 samples in the

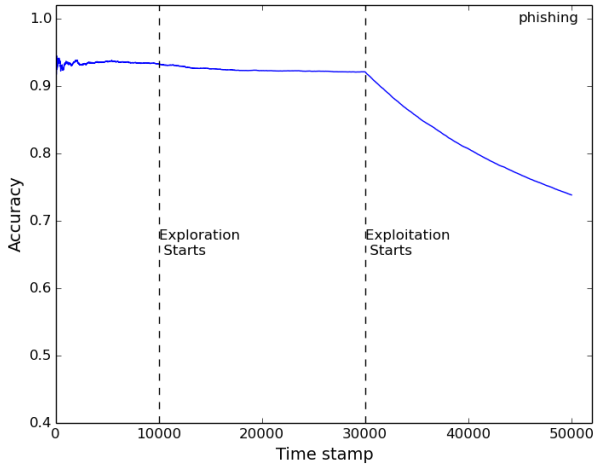


Figure 6.6: Simulation of adversarial concept drift using the *phishing* dataset.

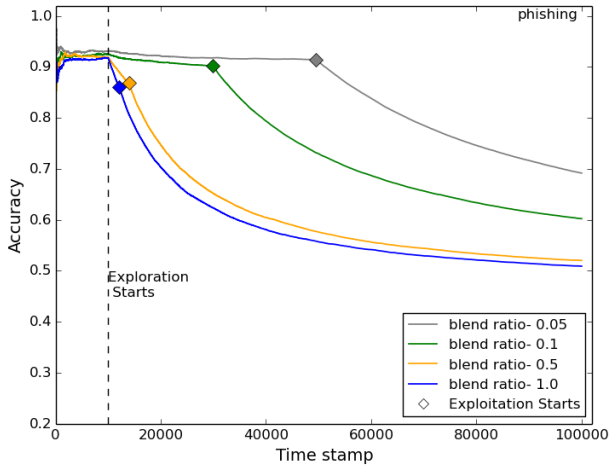


Figure 6.7: Effect of varying blend ratio on the detect-ability of the exploration phase. Lower blend ratios cause attacker reconnaissance to go unnoticed, while delaying onset of attacks.

exploration phase to reach 1000 explored samples. However, this ensures that the drop in accuracy ($< 3\%$), is small enough to go unnoticed. As such, the introduction of the λ_{Blend} parameter allows for more realistic simulation of an adversary's behavior, for better analysis of adversarial concept drift streams.

Using the proposed *SEE-Stream* framework, we can simulate the SEE framework to work in a temporal environment, and can adjust the characteristics of the stream to analyze particular aspects of our security measures. The stream generated by SEE-Stream, repre-

sents the essential characteristics of the original SEE framework, while making it amenable for experimentation in a streaming environment.

6.4 Experimental evaluation of the *Predict-Detect* framework on adversarial drifting streams

In this section we presents experimental evaluation of the *Predict-Detect* classifier framework, on streaming data exhibiting adversarial drift. Adversarial drift is simulated on real world datasets, using the *SEE-Stream* framework. Section 6.4.1 presents experimental setup and methods used for comparing the efficacy of the proposed framework. Experimental analysis on 4 datasets, in the presence of a single adversarial drift, is presented in Section 6.4.2. Experimental analysis on streams exhibiting multiple subsequent drifts is presented in Section 6.4.3.

6.4.1 Experimental methods and setup

6.4.1.1 Methodologies used for comparative analysis

The effects of the adversarial drift, on the classification performance over time, is demonstrated by experimentally comparing the following drift handling methods.

1. *Static Baseline Model (**NoChange**):* This methodology is overly optimistic, as it assumes that the data will never drift, and that the initial trained model is sufficient to retain performance over time. This is an unrealistic assumption, but this serves as a lower baseline for evaluating other drift handling strategies. Any proposed methodology should be atleast as good as this strategy, to be of any real use.

2. *Fully Labeled Accuracy Tracking (**AccTr**):* This serves as an upper baseline for our evaluations, as it represents an optimal case, where all the data is labeled and the labels are immediately available after the prediction is made on an input sample. This model tracks the classifier's predictive performance (e.g., Accuracy), to signal drift. An unsupervised technique is considered effective, if it provides performance

close to the AccTr approach, while reducing labeling requirements. The accuracy is tracked incrementally by using the EWMA [165] formulation of change tracking, using the same stream updating equation as Algorithm 6.2, but using the prediction error, instead of the disagreement score.

3. *Margin Density Drift Detection (**MD3-RS**)*: The MD3 methodology was proposed in Chapter 4, as an alternate to traditional unsupervised drift detectors. MD3 was shown to be more reliable than traditional distribution tracking methodologies, as it tacitly involves the classifier's notion of uncertainty into drift detection. The MD3-RS methodology uses a random subspace ensemble for the detection purposes. A sudden increase in the number of samples in the ensemble's margin (i.e., region of disagreement), is considered to be indicative of a drift. Comparing with MD3 provides us with insights into the need for adversarial awareness in drift detection. In our experiments, we consider a random subspace ensemble of 50 linear SVMs, with 50% of the features in each base model. The threshold for the certainty margin is taken to be as 0.5.

4. *The Predict-Detect framework without feature shuffling for retraining (**PD-NoShuffle**):* This is the proposed *Predict-Detect* classifier framework. With 50% of the features belonging to the *Prediction model* and the other 50% belonging to the hidden *Detection model*. The drift is detected based on tracking the disagreement between the two models. Upon drift confirmation, the individual models are retrained without regenerating the feature splits. This methodology evaluates the ability and effects of continuing to use the same set of initially split features, so as to ensure feature importance hiding, for a longer period of time.

5. *The Predict-Detect framework with feature shuffling for retraining (**PD-Shuffle**):* This is the proposed framework similar to PD-NoShuffle, except for the fact that retraining involves reshuffling all features and then regenerating the two separate prediction and detection models. This model evaluates the impact of ignoring temporal

information gained by an adversary, and focuses on impact against multiple independent adversaries over time.

The *Prediction* and the *Detection* model in the proposed methodology, are comprised of a random subspace ensemble of 50 linear SVMs (L1 penalty, regularization constant c=1), each with 50% of the features allocated to them (randomly chosen). We use the same ensemble framework for the MD3 classification model, as well as for the prediction model for the AccTr and the NoChange model, to ensure consistency in evaluating the methods.

6.4.1.2 Description of datasets and experimental setup

The datasets of Table 6.1, are used for the generation of the adversarial drifts. All data was normalized to the range of [0,1], and data was converted to numeric/binary features type only. The synthetic datasets is a 10 dimensional dataset, with two classes. The *Legitimate* class is normally distributed with a μ-0.75 and σ=0.05, and the *Malicious* class is centered at μ-0.25 and σ=0.05, across all 10 dimensions. The CAPTCHA dataset is taken from [31], and it represents a problem concerned with the classification of mouse movement data for humans and bots, for the task of behavioral authentication. The phishing dataset is taken from [145], and it represents classification between malicious and benign web pages. The digits dataset [145] was taken to represent a standard classification task. Only classes 0 and 8 were considered, so as to convert it to a binary classification task. In all datasets, the class 0 was considered to be the *Legitimate* class. After the dataset is prepared, it is used in the *SEE-Stream* framework, to generate the adversarial stream. The data stream is considered balanced with a $\lambda_{Imbalance}$=0.5 (unless otherwise specified), and the blending ratio λ_{Blend} is taken as 0.05, to avoid triggering drifts in the exploration phase. Detection uses a threshold of θ=3. The parameters of labeling and retraining N_{train}, and the profile of the stream are discussed at the beginning of every following subsections.

The generation of adversarial drift is based on real world datasets, which are used in the *SEE-Stream* framework to simulate a streaming environment. The attacks considered are the Anchor Points (AP) attacks of Section 3.3.2 and the Anchor Points - High Confidence

TABLE 6.1

Description of datasets used for adversarial drift evaluation.

Dataset	#Instances	#Attributes
Synthetic	500	10
CAPTCHA	1886	26
Phishing	11055	46
Digits08	1499	16

(AP-HC) attacks of Section 5.3.4.1. Anchor points attacks is based on radial search in the exploration phase, to find regions of the data space which is recognized as *Legitimate* by the defender's model. The AP-HC attacks are an extension of the anchor points attacks, where an attacker filters out low confidence exploration samples before starting the exploitation attack phase. The high confidence attacks are representative of a sophisticated adversary, which uses all available probe-able information to launch an attack, which avoids falling in the regions of uncertainty of the defender's classifier. The experiments in this section, will evaluate the effect of these attacks in a streaming environment, by encapsulating them in the SEE-Stream framework. The exploration budget ($B_{Explore}$) for the attacks is taken as 1000 samples and the attack exploitation payload is taken to be 2000. For the High Confidence attacks (AP-HC), the generated exploration samples are filtered to remove the low confidence samples. This filtering is done by using a random subspace model (50 Linear SVM, 50% of features in each model), with a high regularization constant c=10 (to ensure robustness against stray probes). Samples with confidence less than the confidence threshold of $\theta_{Adversary\_confidence}$=0.8 are eliminated, before exploitation starts. This is done based on recommendation from Section 5.3.4.1, which shows that drift detection can be vulnerable to such filtering. We use the same parameter configurations as in the SEE framework, to extend its evaluation to a streaming domain, without changing its core behavior.

In all experiments in this section, averages are reported over 10 runs of the experiments. The experiments are performed using Python and the scikit-learn machine learning library [149]. Experimental analysis with a single simulated adversarial drift in the stream is presented in Section 6.4.2 and experiments with multiple consecutive adversarial drifts is

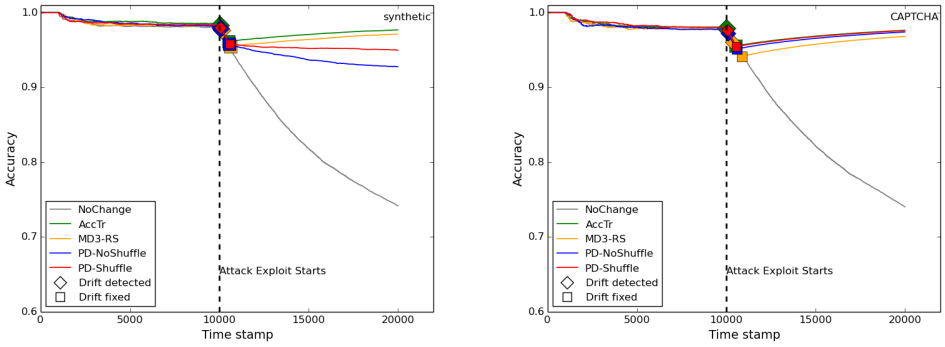
presented in Section 6.4.3.

6.4.2 Analysis on data streams with a single simulated adversarial drift

In this section, we evaluate the performance of the different drift handling methodologies, on a data stream with a single adversarial drift. Drift is simulated using the *SEE-Stream* framework, with a stream length of 20,000 samples and a $t_{Explore}$=1000 and $t_{Exploit}$=10,000. The chunk size is taken to be N=500, and retraining/confirmation is performed by considering $N_{train} = N$ additional labeled samples, after drift indication. We will first evaluate the impact of the Anchor Points (AP) attack strategy in streaming domain, and then move on to the Anchor Points High Confidence (AP-HC) attacks, which represents an adversary determined to avoid detection.

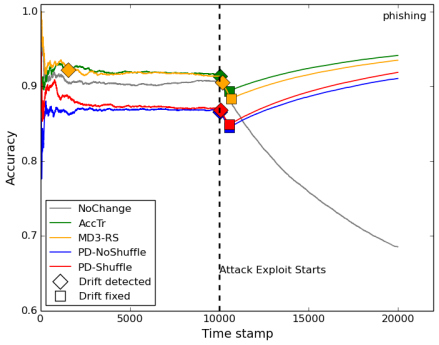
The results of the Anchor Points (AP) attacks are shown in Figure 6.8. It can be seen that the AP attacks when used in the *SEE-Stream* simulator, leads to a concept drift, which causes the performance to drop after $t_{Exploit}$=10,000. This is visible from the sudden drop in accuracy in the performance of the NoChange methodology, which assumes that the data will be static throughout. The drop in accuracy during the exploration phase (t=1,000-10,000) is minimal, in accordance with out simulation goals, as an adversary tries to hide its tracks while performing reconnaissance of the system. The need for drift detection and retraining is highlighted by the decreasing performance of the NoChange model, which rapidly becomes unusable after the drift. The fully labeled AccTr approach is able to effectively and quickly detect changes in the stream, allowing it to fix itself and maintain high accuracy, in all cases. This however comes at an unrealistic assumption of all labels being available immediately. Nevertheless, these two methodologies provide the upper and the lower baseline, for our comparative analysis.

The MD3 approach is able to detect the Anchor Points attacks, by recognizing drift in the margin density, and as such is able to maintain accuracy over the course of the stream. The drift detected and the period when retraining starts is highlighted in Figure 6.8. In all the 4 streams, the MD3 approach is seen to detect drifts swiftly after they start, and

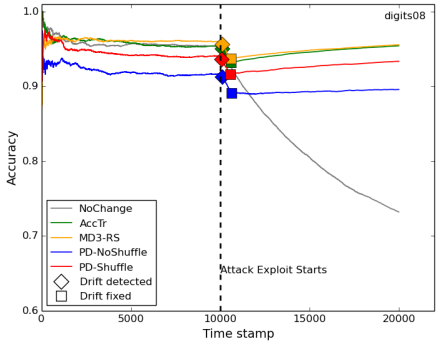


(a) synthetic

(b) CAPTCHA



(c) phishing



(d) digits08

Figure 6.8: Accuracy over time for streams with single adversarial drift, based on Anchor Points (AP) attacks.

as such maintains accuracy close to the fully labeled AccTr approaches. This is done at a low labeling rate of 2.5% on average, as seen in Table 6.2. The performance of the MD3 approach is better than the *Predict-Detect*(PD) approaches, as seen by the higher accuracy (2.25% on average), in Table 6.2. This is because the MD3 approach uses all its available feature set, to train models and provide prediction at every step. The PD-Shuffle outperforms the PD-NoShuffle ($\Delta = 1.15\%$ on average), as the shuffling allows for re-calibrating the feature importance of the models in the two ensembles. However using either method, we obtain accuracy within 5% of the fully labeled AccTr approach. The PD models, like the MD3 model, uses labels only when the drifts are first signaled by the unsupervised tracking mechanism. As seen in Table 6.2, all three unsupervised drift detectors use only 2.5% labeling, as opposed to 100% in the AccTr approach, to provide

TABLE 6.2

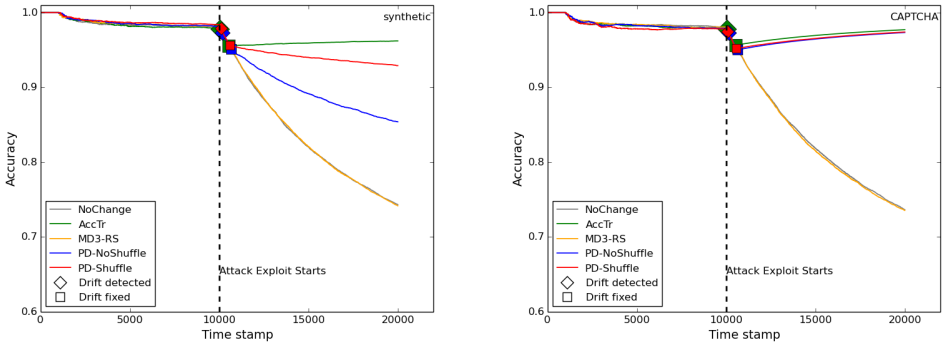
Accuracy and Labeling% of the NoChange, AccTr, MD3, PD-NoShuffle and PD-Shuffle methodologies, over streams with a single adversarial drift.

Dataset	Methodology	Anchor Points Attack		Anchor Points - High Confidence Attack	
		Accuracy	Labeling%	Accuracy	Labeling%
synthetic	NoChange	74.7	0	74.2	0
	AccTr	97.4	100	95.7	100
	MD3	96.9	2.5	74.2	0
	PD-NoShuffle	92.5	2.5	85.4	5
	PD-Shuffle	94.2	2.5	93.7	2.5
CAPTCHA	NoChange	74.1	0	74.3	0
	AccTr	97.6	100	97.7	100
	MD3	97.2	2.5	73.9	0
	PD-NoShuffle	97.4	2.5	97.4	2.5
	PD-Shuffle	97.2	2.5	96.5	2.5
phishing	NoChange	68.9	0	68.9	0
	AccTr	93.9	100	93.8	100
	MD3	92.5	2.5	68.5	0
	PD-NoShuffle	90.1	2.5	90.1	4.18
	PD-Shuffle	91.2	2.5	91.3	2.5
digits08	NoChange	73.5	0	71.9	0
	AccTr	94.9	100	95.6	100
	MD3	95.5	2.5	71.7	0
	PD-NoShuffle	90.8	2.5	91.8	2.5
	PD-Shuffle	92.8	2.5	92.9	2.5

sufficiently high performance, over time.

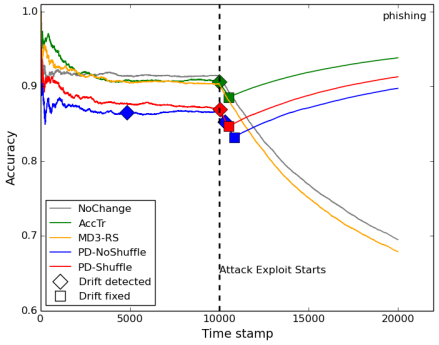
The results of applying the Anchor Points - High Confidence (AP-HC) attacks, for generating adversarial drift is presented in Figure 6.9. This attack is based on a sophisticated adversary, who uses all available probe-able information to launch an attack oh high confidence (one that evades a majority of feature vectors simultaneously) (Section 5.3.4.1). Robust classifiers which integrate many of the informative features into the learned model, are susceptible to these high confidence attacks, which can make adversarial detection and recovery difficult.

The results of Figure 6.9 demonstrate that the MD3 approach fails to detect these attacks, as the accuracy of the MD3 approach is seen to be no better than the NoChange approach. Both approaches fail at detecting any drift. The high confidence attack samples,

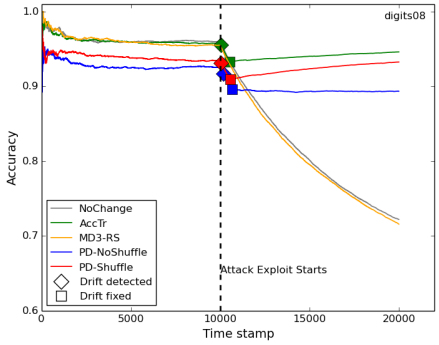


(a) synthetic

(b) CAPTCHA



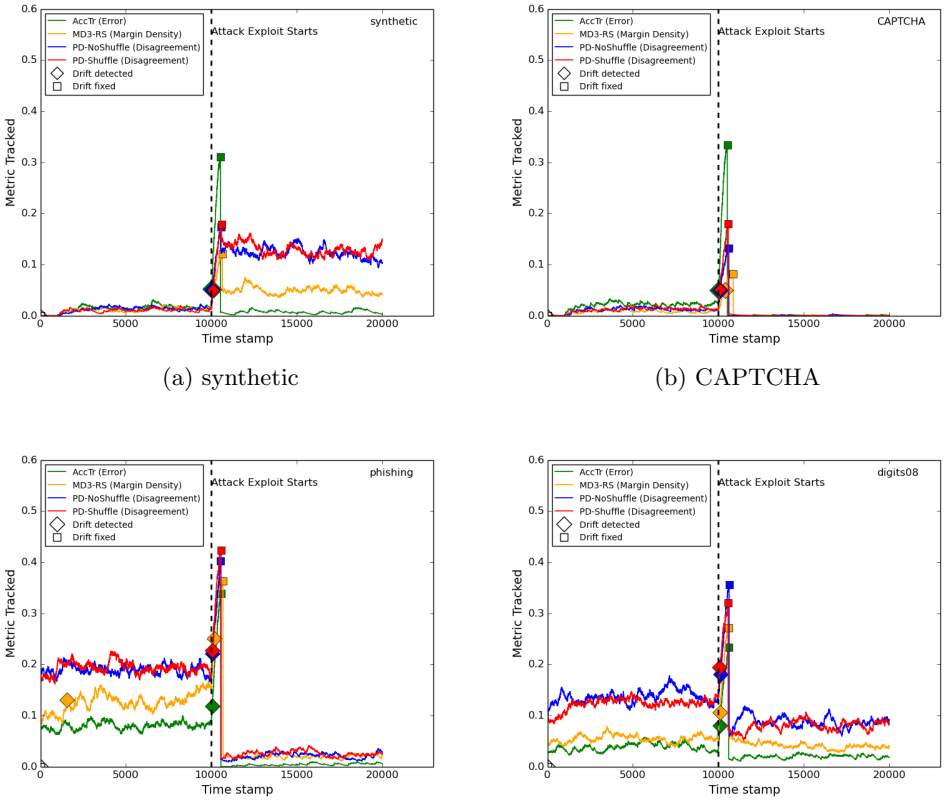
(c) phishing



(d) digits08

Figure 6.9: Accuracy over time for streams with single adversarial drift, based on Anchor Points-High Confidence (AP-HC) attacks.

evade several of the features simultaneously, causing the margin density based detection, to be circumvented. Margin density relies on having a few informative features drifting, while the other remain static (Section 4.3.1). This condition is intentionally violated by an adversary seeking to go undetected for a long time (Figure 6.2). An adversary begins by probing the black box model, in its attack exploration phase, and then weeding out low confidence samples from the exploration set. The exploitation phase will then generate samples which simultaneously evade a majority of the discriminatory features, obtained from the training data by the deployed classifier. This causes the samples to fall outside the regions of uncertainty (margins), of the defender, leading to failed unsupervised attack detection. The PD approach was developed to address this issue, by using an adversarial aware design. It does so by hiding feature importance, thereby shielding them from probing



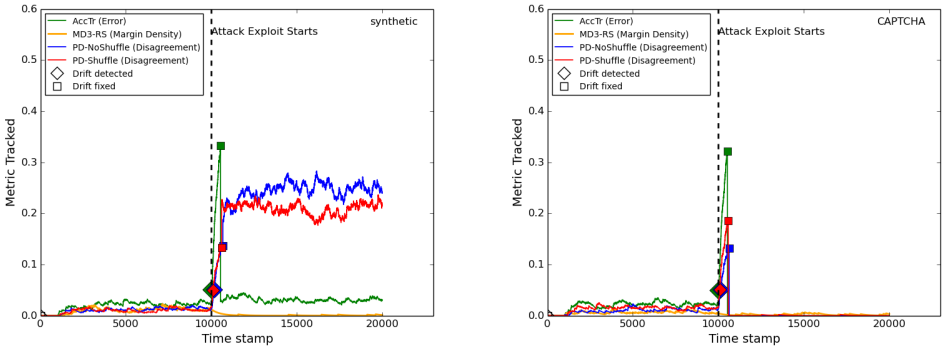
(a) synthetic (b) CAPTCHA

(c) phishing (d) digits08

Figure 6.10: Metrics being tracked by the different drift handling techniques, for the AP attacks. AccTr tracks error rate based on fully labeled data. MD3 tracks margin density from unlabeled data. The PD-Shuffle and the PD-NoShuffle, track the disagreement between the prediction model and the hidden detection model.

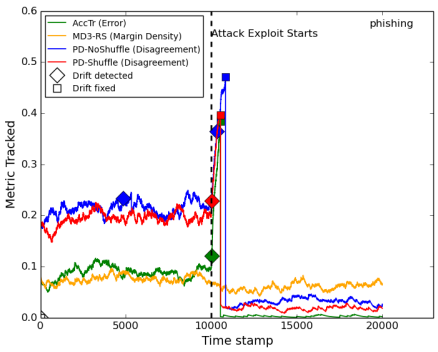
based attacks. This ensures that the adversary will have a misguided notion of confidence, as it will not be able to obtain information about the exact influence of a subset of feature, no matter how high its exploration probing budget gets. By tracking the successful reverse engineering and evasion of a subset of exposed features, with an increased uncertainty over the other set of features, the PD methodology aims to detect adversarial activity. This detection will allow for an unsupervised early indicator of attacks, enabling the defender to take effective countermeasures, suited to their application.

The *Predict-Detect* framework is able to detect these drifts caused by the high confidence attacks, shown by high performance after attacks, in Figure 6.9. The drift detection is prompt, and close to the AccTr approaches ($\Delta = 3.3\%$, on average). The difference in

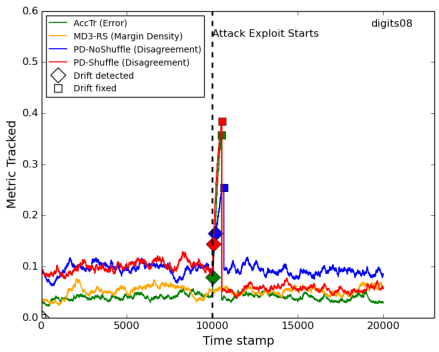


(a) synthetic

(b) CAPTCHA



(c) phishing



(d) digits08

Figure 6.11: Metrics being tracked by the different drift handling techniques, for the AP-HC attacks. AccTr tracks error rate based on fully labeled data. MD3 tracks margin density from unlabeled data. The PD-Shuffle and the PD-NoShuffle, track the disagreement between the prediction model and the hidden detection model.

performance, between the two PD approaches, is seen in case of the final accuracy, due to the different retraining approaches used. The PD-Shuffle outperforms PD-NoShuffle, due to reassessment and re-splitting of feature importance, possible in the retraining phase. However, the PD-NoShuffle approach provides security against possible temporal attacks, in which an adversary might be able to collect information about different features over time, to cause a more potent data nullification attack. It does so with a compromise in the accuracy (<2.5% on average), compared to the PD-Shuffle methodology, which is a reasonable trade off. In either case, the PD methodologies is able to detect drifts with <5% labeling, and is able to maintain classifier performance over time, as seen in Table 6.2. The higher labeling budget for the PD-NoShuffle case, is due to the increase in false alarms,

which leads to requesting of labeled data, which are eventually discarded, as they indicate no significant drop in the accuracy.

The progression of the metrics tracked by the different drift detectors is shown in Figure 6.10, for the Anchor Points (AP) attacks, and Figure 6.11, for the Anchor Points - High Confidence (AP-HC) attacks. AccTr tracks the error rate (or Accuracy), MD3 tracks the margin density, PD track the disagreement between the Prediction and the Detection models. It can be seen that all 4 metrics depict a significant jump at the attack exploitation phase, and remain relatively stable before and after the drift, in case of the AP attacks in Figure 6.10. This indicates a high signal-to-noise ratio for these information metrics, and their effectiveness in detecting drifts, which are caused by anchor points attacks. The effect of the metrics tracked over time for the AP-HC attacks is shown in Figure 6.11, where it is seen that the margin density metric is unable to detect any changes in the face of a high confidence attack, leading to its inefficacy. The adversarial aware PD approaches, are able to detect attacks similar to the AccTr approach. This demonstrates the importance of accounting for an adversarial aware design in the training phase of a classifier, and the effectiveness of simple solutions implemented in the design of a classifier, leading to long term security benefits. The preemptive strategy of hiding feature importance, leads to benefits in terms of better attack detection, lower dependence on labeled data and effective responsiveness to attacks, for higher availability and security, of the machine learning based system.

6.4.2.1 Effects of varying the number of hidden features

In the experiments so far, the available set of features are considered to be split equally split between the *Prediction* and the *Detection* model, based on the feature importance based splitting methodology of Section 6.2.2.1. This was motivated by the intuition to equally consider prediction on incoming samples and detection of adversarial drifts, as important tasks, over the course of the stream. The main reason for the effectiveness of the *Predict-Detect* design, is its ability to hide the feature importance of some of the features,

TABLE 6.3

Effects of varying the percentage of important hidden features, on the accuracy over the adversarial data stream.

Dataset / % of hidden features →	PD-NoShuffle			PD-Shuffle		
	10%	25%	50%	10%	25%	50%
synthetic	0.96	0.91	0.88	0.96	0.95	0.95
CAPTCHA	0.98	0.97	0.97	0.98	0.97	0.97
phishing	0.92	0.92	0.91	0.93	0.92	0.91
digits08	0.96	0.95	0.94	0.96	0.95	0.94

TABLE 6.4

Effects of varying the percentage of important hidden features, on the number of drifts signaled.

Dataset / % of hidden features →	PD-NoShuffle			PD-Shuffle		
	10%	25%	50%	10%	25%	50%
synthetic	1	1	1	1	1	1
CAPTCHA	1	1	1	1	1	1
phishing	2	1	1	2	1	1
digits08	3	1	1	3	1	1

from being probed and reverse engineered by an adversary. As such, it should be sufficient to hide fewer important features, in case they are sufficient to represent a high accuracy orthogonal representation of the training data. Generally, it is expected to have more features in the *Prediction* model, as this will enable better predictive performance for normal functioning of the ML based service, and at the same time delay the onset of attacks. Here, we evaluate the effects of reducing the number of features included in the hidden detection classifier, and its impact on the detection and prediction capabilities of the framework.

Table 6.3 and Table 6.4, present the effects of reducing the number of hidden features from 50% to 10% and 25%. Reducing features in the hidden model is done by blanking out additional features, in accordance to the methodology of Section 6.2.2.1. The features are still distributed round robin, based on their informativeness to the classification task. From Table 6.3, it can be seen that the accuracy of the stream is not significantly affected by reduction in the number of the hidden features, for both the PD-Shuffle and the PD-

NoShuffle scenarios. The number of drifts detected are also seen to be similar, as seen in Table 6.4. Additional false alarms are seen in case of 10% hidden features. However, the effects of these false alarms are minimal, compared to the savings in the labeling obtained by using this unsupervised drift detector. As a guideline, 25%-50% is suggested for the number of important features in the hidden model.

An important consideration for the applicability of the *Predict-Detect* framework, is the presence of multiple orthogonal features in the training dataset, which can result in disjoint classifiers each with high accuracy of prediction. In such a scenario, the prediction and the detection models form a self monitoring scheme, for detecting adversarial activity. So long as the two models can be trained to be disjoint (feature wise) and have high prediction performance, the number of hidden features, the type of the individual models and the training mechanism used, does not have a significant impact on the effectiveness of the framework.

6.4.3 Analysis on data streams with multiple subsequent adversarial drifts

In this section, we evaluate the performance of the different drift handling techniques in a streaming environment with multiple subsequent adversarial drifts. This stream represents a scenario where the system gets attacked, the defender adapts and then the adversary relaunches an attack on the newly deployed system. This represents a real world scenario, where protecting against the onset of attacks is not sufficient. Instead, the ability to detect attacks, fix them and continue to provide services, is paramount. This emphasizes a reactive approach to security, with an understanding of the adversarial nature of the problem, leading to long term benefits.

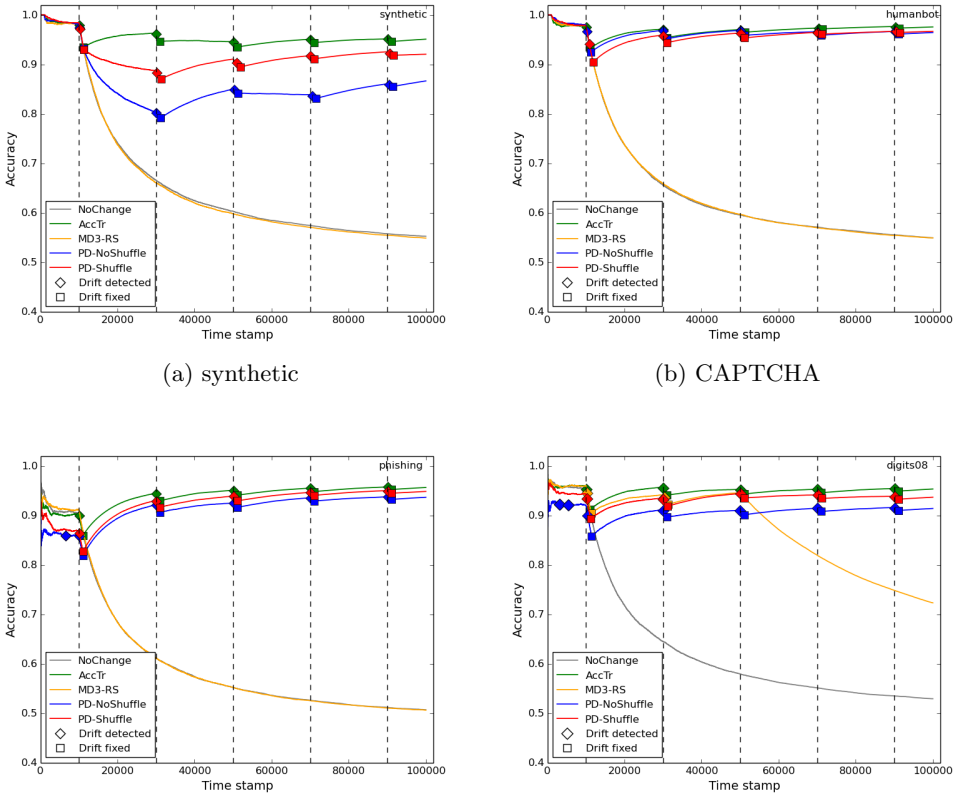
Multiple adversarial drifts are simulated using the *SEE-Stream* framework. The Anchor Points - High Confidence (AP-HC) attacks is used, as it was seen to be more dangerous to the integrity of a reactive system, as shown in Section 6.4.2. We generate 5 subsequent adversarial drifts, to simulate 5 attack-defense cycles, each of 20,000 samples. Exploration occurs from $(1000 + 20000 * i)$ to $(10000 + 20000 * i)$ samples, where $i = 0, 1, , 4$

TABLE 6.5

Results of drift handling, on multiple adversarial drift data streams.

Dataset	Methodology	Accuracy	Labeling%	Drifts Detected
synthetic	NoChange	55.10	0	0
	AccTr	94.80	100	5
	MD3	54.90	0	0
	PD-NoShuffle	87.68	5	5
	PD-Shuffle	90.54	5	5
CAPTCHA	NoChange	54.78	0	0
	AccTr	97.57	100	4
	MD3	54.88	0	0
	PD-NoShuffle	96.48	6	6
	PD-Shuffle	96.77	5	5
phishing	NoChange	50.70	0	0
	AccTr	95.66	100	5
	MD3	62.41	2	2
	PD-NoShuffle	93.08	6	6
	PD-Shuffle	94.65	5	5
digits08	NoChange	52.89	0	0
	AccTr	95.03	100	5
	MD3	72.99	2	2
	PD-NoShuffle	91.49	7	7
	PD-Shuffle	93.53	5	5

represents the cycle number. In each cycle, the adversary learns about the defender, using probes, and then launches a high confidence evasion attack on the defender's classifier. The defender is responsible for detecting the onset of the attacks, and retraining in the event of a confirmed attack. After 10,000 samples (of the previous attack exploitation step), the adversary is perceptive of the inefficacy of its attacks, due to the updated defender's model, and as such it relaunches its attack on the defender. This launch starts the new adversarial cycle, by probing the updated defender model, and launching evasion attacks against it. We do not emphasize details about how an adversary detects that the defender has retrained itself. Instead, it is assumed that an adversary gets perceptive of this change, as it receives *Accept/Reject* feedback on its submitted probe samples. The focus of experimentation is on the defender's ability to detect and retrain, in the face of multiple subsequent adversarial drifts.

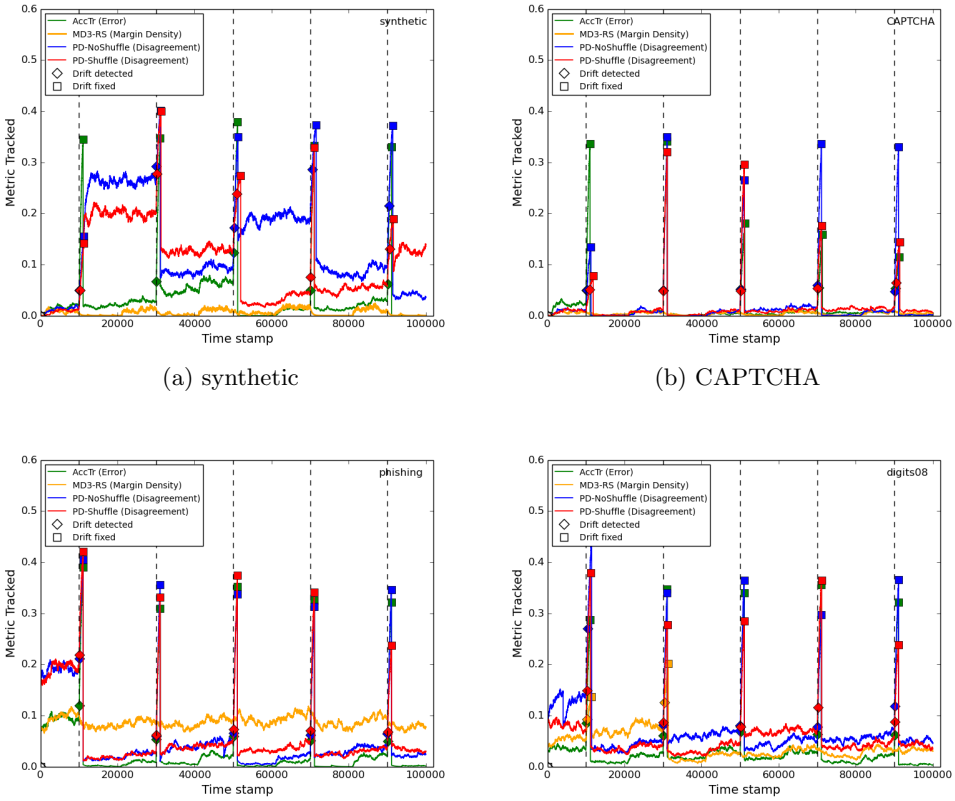


(a) synthetic (b) CAPTCHA

(c) phishing (d) digits08

Figure 6.12: Accuracy over time for streams with multiple subsequent adversarial drift, based on Anchor Points - High Confidence attacks.

The results over 5 cycles of the attack-defense cycle are presented in Table 6.5, and the accuracy over the stream is depicted in Figure 6.12. We consider a chunk size of N=1000, for all streams here, due to the increased size of the stream and the need for smoother analysis of the drift detection behavior. The progression of the streams in Figure 6.12, indicates that the drift detection behavior of the methodologies is consistent with our observation for the single drift case. The MD3 approach fails to effectively detect and adapt to drifts, this is seen by the reduced accuracy of this approach when compared to the fully labeled drift detector AccTr (34.5% lower accuracy on average). This is a result of the inefficacy of the MD3 approach, to track high confidence attacks. The MD3 detection methodology is evaded at test time, by an adversary using the AP-HC attacks. This results in the MD3 approach being similar in performance to the NoChange methodology (Δ=7.9%).



(a) synthetic (b) CAPTCHA

(c) phishing (d) digits08

Figure 6.13: Metrics being tracked by the different drift handling techniques, for the streams with multiple subsequent adversarial drifts. AccTr tracks error rate based on fully labeled data. MD3 tracks margin density from unlabeled data. The PD-Shuffle and the PD-NoShuffle track the disagreement between the prediction model and the hidden detection model.

The PD methods provide performance similar to the fully labeled AccTr approach (Δ=1.9 for PD-Shuffle and Δ=3.6 for PD-NoShuffle). This is achieved at a low labeling rate of 6% for the PD-NoShuffle and 5% for the PD-Shuffle. This is because the *Predict-Detect* framework provides a natural, adversarial-aware, unsupervised mechanism for detecting drifts. This minimum required labeling is needed, for drift confirmation and retraining. The drift detection metrics are depicted in Figure 6.13. It is seen that the disagreement score for the PD serves as an effective surrogate to the fully labeled AccTr approach, as both the signals changes concurrently and with minimal lag, in the event of an attack. The high peak at the exploitation stage of attacks; indicates the high signal to noise ratio of

the disagreement tracking signal and its effectiveness in detecting attacks. This makes the detectors resilient to stray changes, reserving labeling budget and model retraining only for cases where the adversarial attack can lead to serious degradation in the performance. The margin density signal is seen to be ineffective for detecting attacks, as it misses a majority of the drift scenarios in these experiments. This further highlights the need for an adversarial aware design, when implementing streaming data algorithms for security applications.

The PD-Shuffle performs marginally better than the PD-NoShuffle methodology, with an average improvement of 2% in the accuracy of the stream. However, the PD-NoShuffle provides better protection by hiding features importance from an adversary, over time. The drift detection and retraining lag is similar for the PD and the AccTr cases, indicating the effectiveness of the detection capabilities of the proposed framework. This makes the *Predict-Detect* framework effective for usage in an unsupervised, adversarial-aware, streaming environment.

6.4.4 Discussion

The results of experimentation on single and multiple adversarial drift streams, are summarized in Table 6.6. The average number of drifts detected by each of the drift detection methodologies, is presented. The NoChange methodology, owing to its static assumption over the data stream, does not detect any drifts in any of the streams. The fully labeled drift detector, the AccTr, detects drifts in all cases. The same is observed for the *Predict-Detect* classifier. This indicates the efficacy of the proposed framework, to be used as a surrogate for fully labeled drift detector. The proposed methodology is able to detect drifts from unlabeled data, and as such is practical for usage in a streaming environment, where labeling is time consuming and expensive.

The margin density drift detector (MD3), is able to detect drifts in case of the Anchor Points attack (AP), but fails to detect the required drifts in case of a high confidence attack (AP-HC). This is because of the adversarial agnostic design of the MD3 framework. An adversary is able to probe the deployed classifier and is then able to aggregate all the obtained

TABLE 6.6

Summary of adversarial drift detection results. *Average number of drifts detected/Total simulated drifts*, are presented for each methodology.

Methodology	Single adversarial drift		Multiple adversarial drifts
	AP attacks	AP-HC attacks	AP-HC attacks (5 attacks)
NoChange	0/1	0/1	0/5
AccTr	1/1	1/1	4.75/5
MD3	1/1	0/1	1/5
PD-NoShuffle	1/1	1/1	6/5 (1 False alarm on average)
PD-Shuffle	1/1	1/1	5/5

information, to launch high confidence attacks which fall outside the margin/blindspot of the defender's classifier model. Thus, an adversary can be sophisticated to avoid detection by the MD3 approach. The *Predict-Detect* classifier, relies on misrepresenting feature importance to the adversary at test time. As such, even if an adversary is sophisticated and uses a large number of probes, it will not be able to reverse engineer the importance of the hidden *Detection* model, to the classification task. The preemptive design of the framework makes it better prepared for dealing with adversarial activity, by enabling reliable unsupervised detection, and also preventing the onset of high confidence attacks, by a sophisticated adversary.

6.5 Disagreement based active learning on imbalanced adversarial data streams

The analysis thus far, has focused on the ability of the *Predict-Detect* framework to detect drifts from unlabeled data. After a drift is indicated by the framework, labeling additional samples for drift confirmation and for retraining of models, is considered to follow naive strategies. More specifically, 100% of samples from a chunk of $N_{Unlabeled}$ subsequent samples are requested to be labeled, and these samples are then used to confirm drifts and retrain the model. Although this approach is effective for a balanced stream, with near equal probability of occurrence of the *Legitimate* and the *Malicious* class samples, it is not an efficient strategy for imbalanced data streams. This is especially pertinent in adversarial environments, as the adversarial attacks are expected to be a smaller minority

class, in comparison the majority of benign traffic entering the system. To improve the labeling process, we analyze the effect of selecting N_{train} samples ($N_{train} < N_{Unlabeled}$) to label, for drift confirmation and retraining, after the framework indicates a drift using unlabeled data. In our proposed active learning strategy, we use the motivation that samples falling in the disagreement region of the two classifiers have higher probability of being of the adversarial class. Section 6.5.1 presents the proposed active learning methodology, based on the disagreement information between the prediction and the detection models. Experimental analysis is presented in Section 6.5.2.

6.5.1 Active learning using disagreement information - *Disagreement Sampling*

Adversarial drifts are detected by the *Predict-Detect* framework, by tracking the disagreement between the two component models, over time. The efficacy of the detection mechanism relies on the inability of an adversary to successfully reverse engineer all of the feature information, due to the hidden inaccessible detection model. This causes adversarial samples to fall in the disagreement regions of the two classifier models. We extend this motivation to the domain of active learning, for selecting samples for labeling, after a drift is indicated.

The active learning based on disagreement is performed on the collected $N_{Unlabeled}$ samples, after the drift is indicated by the PD model. The proposed active learning process is shown in Figure 6.14. The initial set of collected $N_{Unlabeled}$ unlabeled samples, after drift indication, are processed and N_{train} samples are selected ($N_{train} <= N_{Unlabeled}$), to be labeled by the Oracle. The active learning methodology aims to embody maximum informativeness into the selected N_{train} samples, about the new drifted stream distribution. A large number of adversarial class samples are aimed to be selected, as this is the drifting class and is often the minority in most real world applications. A good number of adversarial class samples selected, will lead to more balanced datasets, for better drift confirmation and retraining of models.

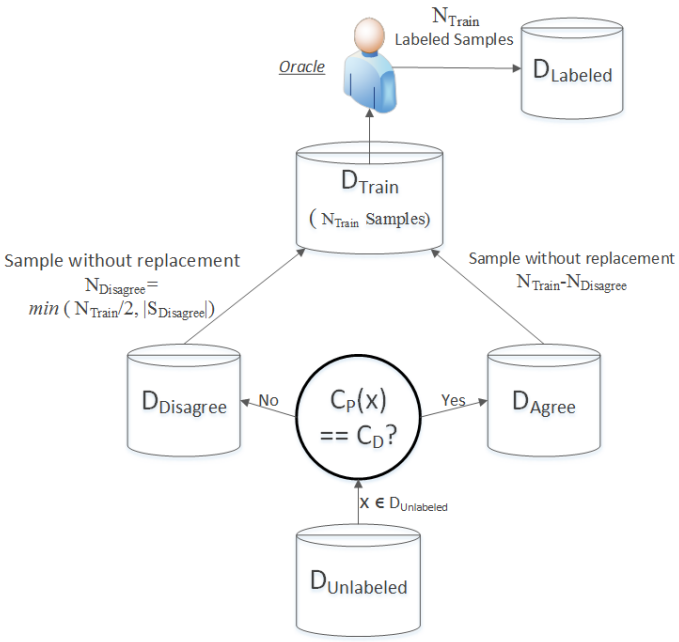


Figure 6.14: Disagreement based active learning methodology.

The active learning methodology is presented in Figure 6.14. The set of unlabeled data samples $N_{Unlabeled}$, collected after drift is indicated, is split into two pools: D_{Agree} and $D_{Disagree}$, based on the predicted class labels by the *Prediction* model (C_P) and the *Detection* model (C_D). Samples to be labeled are first obtained by random sampling on the pool of disagreeing samples $D_{Disagree}$, until the samples in this pool are exhausted or we reach 50% of our labeling budget (N_{train}). This ensures that we allocate more priority to the disagreement region, where we have a higher chance of obtaining the samples of the adversary class. The remaining samples are obtained by random selection on the samples which fall outside the disagreement region (i.e., by sampling from D_{Agree}). Since the data is predominantly assumed to be of the *Legitimate* class, this sampling will allow us to receive balanced samples for better retraining on the attack samples, while providing secondary validation on the static nature of the legitimate class samples. The final set of N_{train} samples are sent to the Oracle for labeling. By integrating the disagreement information into the labeling process, this methodology aims to achieve a more representative set of labeled samples, for better detection and retraining over imbalanced adversarial streams.

6.5.2 Experimental analysis on imbalanced data streams with limited labeling

Identifying and labeling samples of the adversarial class, in a streaming environment, is an important concern especially when the data stream is imbalanced. This is because, in a streaming data with drift in the minority class, the changes are harder to track and detect in the data space [112]. In an adversarial environment, the attack class is often the minority class [3], as opposed to the vast majority of legitimate samples being submitted to the system. As such, the adversarial class's impact could be shadowed by the legitimate class, causing the attacks to go unnoticed. We present experiments in this section, to provide an initial motivation into the usage of the disagreement samples, for better labeling and retraining of the models.

To demonstrate the effects of the proposed disagreement based active learning methodology, we first perform experiments on the synthetic dataset, using the same parameters and stream profile as in Section 6.4.2 (20000 samples, exploration starts at 1000 and exploitation starts at 10000), and by using the AP-HC attacks. The detection model is taken to be the PD model with shuffling of features for retraining (PD-Shuffle). The focus of this analysis is on the efficacy of the active learning approach based on the disagreement information, on the retrain-ability and overall quality, following the drift detection. Since the data stream is considered to be imbalanced, we update the Algorithm 6.2, to use f-measure for the confirmation of the drift, instead of accuracy. The size of the $N_{Unlabeled}$ samples after drift detection is taken to be 500, and the detection threshold is kept at θ =1.5, to account for imbalance in the data stream. The simulation of adversarial drifts in imbalanced streams is possible in the *SEE-Stream* framework, by usage of the $\lambda_{Imbalance}$ parameter. The effect of varying the stream imbalance rate $\lambda_{Imbalance}$ and the labeling rate, is analyzed. Labeling rate is the fraction of samples in the set of $N_{Unlabeled}$ samples (buffered after drift indication), which will be labeled to form the set of N_{train} samples, which are ultimately used for drift confirmation and retraining.

The effect of varying the labeling rate and the imbalance in the data stream, is shown in Table 6.8. The f-measure of the final stream and the percentage of malicious samples, in

TABLE 6.7

Effects of *Random Sampling*, on the classifier performance and the balance of the obtained re-training set, on the synthetic dataset. Italicized values indicate critical regions of evaluation imbalanced stream with low labeling ratio.

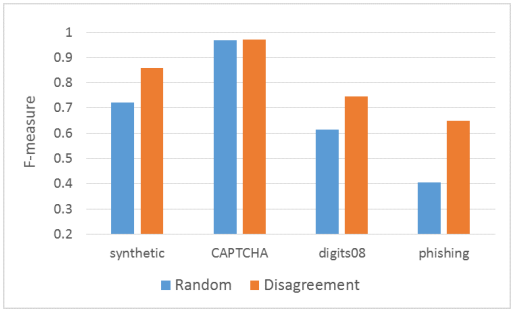
Labeling→ / Imbalance↓	100%		10%		5%	
	f-measure	Malicious%	f-measure	Malicious%	f-measure	Malicious%
0.5	0.93	51.20%	0.92	52%	0.89	36%
0.1	0.91	11.40%	*0.72*	*10%*	*0.65*	*8%*
0.05	0.89	6.20%	*0.78*	*4%*	*0.66*	*4%*

TABLE 6.8

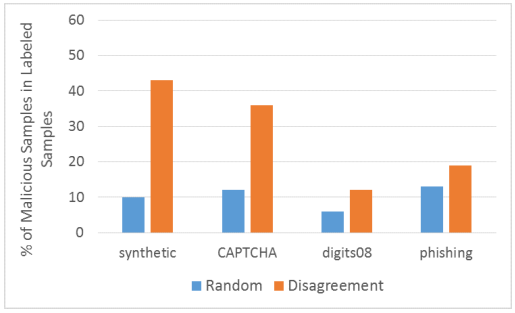
Effects of *Disagreement Sampling*, on the classifier performance and the balance of the obtained re-training set, on the synthetic dataset. Italicized values indicate critical regions of evaluation imbalanced stream with low labeling ratio.

Labeling→ / Imbalance↓	100%		10%		5%	
	f-measure	Malicious%	f-measure	Malicious%	f-measure	Malicious%
0.5	0.92	51.60%	0.89	56%	0.91	61%
0.1	0.90	9.00%	*0.86*	*43%*	*0.86*	*48%*
0.05	0.94	5.20%	*0.88*	*33%*	*0.84*	*48%*

the labeled N_{train} samples, is shown in the table. High values on both metrics are desirable. As a baseline, the result of using random sampling (without replacement) is also presented in Table 6.7. From Table 6.7 and Table 6.8, we can observe that for imbalanced stream with limited labeling, the disagreement based active learning leads to a more balanced dataset for retraining. This results in a better f-measure for the stream, as more malicious samples are detected and retrained on. This is more pertinent in the case of reduced labeling (shown by emphasized values in both tables). Even for a 95% imbalanced stream, and a budget of only 5% of the $N_{Unlabeled}$ samples (=25 samples in this case), a f-measure of 0.84 is seen across the stream for the *Disagreement Sampling*. This is beneficial, as the inability to detect the adversarial samples and train from them effectively, leads the random sampling approach to fall drastically in performance after attacks (with f-score of 0.66). The proposed PD model can detect drifts from imbalanced stream, and innately provides an active learning mechanism to further improve on labeling and retraining. This makes it attractive for usage in imbalanced streaming environments, where labeling is time consuming and expensive.



(a) Effect of sampling, on f-measure of stream.



(b) Effect of sampling, on the percentage of malicious samples obtained for retraining.

Figure 6.15: : Effect of *Random Sampling* and *Disagreement Sampling*, at an imbalance rate of $\lambda_{Imbalance}$=0.1 and a labeling rate of 10% (i.e., N_{train}=0.1 * $N_{Unlabeled}$).

The results of the proposed active learning methodology, on imbalanced streams with $\lambda_{Imbalance}$ of 0.1 and at a labeling budget of 10% on the set of unlabeled training data, is shown in Figure 6.15. Across the 4 datasets, it can be seen that the active learning method outperforms the random labeling strategy, by providing better f-measure and a more balanced labeled training dataset. While several active learning algorithms have been proposed in literature for streaming data [105], the focus here is on the innate availability of information, due to the disagreement between the two model of the defender. This information is leveraged for drift detection from unlabeled data, and also for the task of active learning. The hidden classifier serves as a honeypot to capture adversarial samples, for our task of active learning and retraining. This design embodies the principle of dynamic adversarial learning, where forethought and adversarial awareness, leads to advantages in the detection and the retraining process.

6.6 Towards a generalized and extensible *Predict-Detect* classifier framework

The developed *Predict-Detect* framework, is used to demonstrate the importance of feature importance hiding, in a streaming adversarial environment. The information leverage on the defender's part, allows for advantages in reactive security, mainly for better attack detect-ability and recovery. In this section, we provide additional ideas, which motivate the extensibility and customize-ability of the framework, to suit different appli-

TABLE 6.9

Results of splitting datasets, vertically, into orthogonal feature subsets. Number of splits is given as K. Splits possible within 5% and 10% accuracy loss, are italicized.

Dataset	CAPTCHA	Phishing	Spam	Spamassassin	Nsl-Kdd
#Instances	1885	11055	6213	9324	37041
#Attributes	26	46	499	499	122
Monolithic Model Accuracy% (K=1)	99.9	96.3	96.2	97.5	98.1
Min Accuracy% at K = 2	99.8	92.1	95.9	96.7	97.1
Min Accuracy% at K = 3	99.6	88.2	95.4	96.2	92.1
Max Partitions within 5% accuracy drop (K@5%)	*5*	*2*	*6*	*7*	*2*
Model performance at K@5% (Mean Accuracy, Min Accuracy)	(0.99, 0.99)	(0.92, 0.92)	(0.94, 0.92)	(0.94, 0.93)	(0.97, 0.97)
Max Partitions within 10% accuracy drop (K@10%)	*9*	*3*	*10*	*10*	*4*
Model performance at,K@10% (Mean Accuracy, Min Accuracy)	(0.99, 0.95)	(0.9, 0.88)	(0.90, 0.89)	(0.92, 0.91)	(0.94, 0.93)

cation needs. The ideas presented, can be used for designing and deploying classifiers in *Dynamic - Adversarial* domains. Section 6.6.1 presents motivation for developing a generalized *Predict-Detect* framework. An ensemble based multiple classifier system formulation of the framework is presented in Section 6.6.2.

6.6.1 Motivation - Orthogonal informative feature subsets

Most cybersecurity datasets tend to be high dimensional, and also tend to include multiple orthogonal subsets of informative features. This is demonstrated in the experimental results of Table 6.9, where experiments are performed on 5 datasets from the cybersecurity domain, namely: CAPTCHA [31], phishing [145], spam, spamassassin [183,184] and nsl-kdd [185] dataset. The datasets were split vertically (based on features, with each feature containing all instances of the dataset), based on the feature ranking criterion of Algorithm 6.1. Each model in the table is a Linear SVM. Accuracy was computed using 10-fold cross validation over the entire dataset and is reported in Table 6.9.

In Table 6.9, effects of splitting a dataset, based on features, into K orthogonal subsets, is demonstrated. For K=1, there is no split in the feature space and the learned

model is trained on the entire original dataset. For $K > 1$, the minimum accuracy of the K trained models is reported. Additionally, the number of splits- K possible at a 5% and 10% loss in the minimum accuracy, is also reported. From the results, it can be seen that it is reasonable to divide the dataset's features into $K > 3$ sets of independent disjoint subsets, such that the minimum accuracy over any feature subset is within an acceptable range (<10%), when compared to a monolithic classifier trained on the entire set of features. For the high dimensional datasets of spam and spamassassin, it is seen that 10 partitions are possible within 10% accuracy of the monolithic classifier.

We use minimum accuracy of the models for evaluation, as we are interested in feature set partitions, such that all splits have a high information content. Consequently, we want to train multiple orthogonal models of high quality, from the original training dataset. The extensibility of the *Predict-Detect* framework, is motivated by the presence of this orthogonal information, and the ability to train models on disjoint feature subsets. This will allow the development of the framework as a modular multiple classifier system (MCS), which effectively leverages the orthogonal trained models.

6.6.2 Using the *Predict-Detect* framework as a multiple classifier system (MCS)

The design of a classifier system, relies on effectively utilizing the set of orthogonal information, to meet the system's prediction and security needs. In *Complex learning* based design [66, 72, 73, 75], the orthogonal information is utilized by training multiple classifiers, and then using a consensus based mechanism to determine if a given sample should be admitted to the system as a legitimate sample. This in turn leads to an increased effort on the adversary's part, to modify its attack payload, by mimicking multiple features of the legitimate class. In *Randomization* based techniques [74, 76, 91, 98], a set of models are randomly chosen to provide the prediction at any given time. This serves to confuse the adversary about the true state of the classifier's learned boundary, making reverse engineering difficult. Based on analysis in Section 5.4, we observe that Randomization and Complex learning are ineffective in providing long term security, as they tend to cause

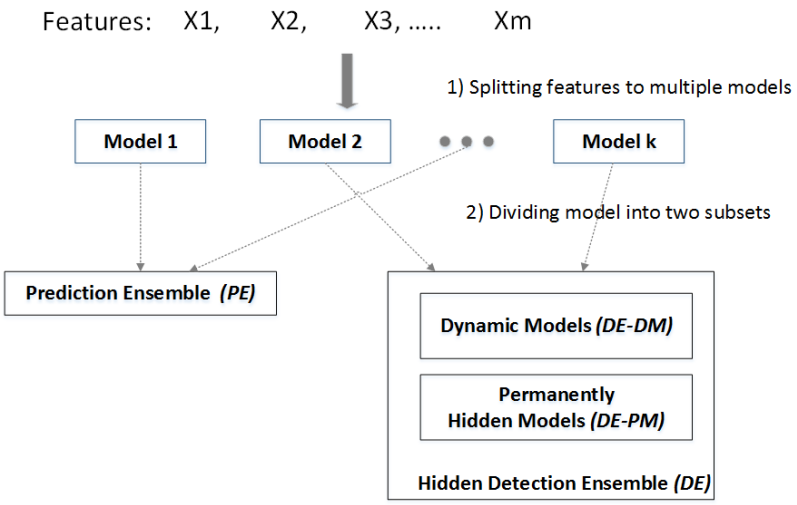


Figure 6.16: Generalized design of the Predict-Detect Ensemble framework, as a multiple classifier system.

attacks of high confidence, making their detection and recovery difficult. The proposed *Predict-Detect* classifier was shown to be more advantageous to a reactive system, as it misrepresents feature importance and as such is able to cause adversaries to have high attack uncertainty. We extend the proposed *Predict-Detect* design to a multiple classifier framework, which provides for flexibility and modular extensibility for better long term security and usage.

The generalized multiple classifier system (MCS) view of the *Predict-Detect* framework, is presented in Figure 6.16. The initial set of m features is split into K subsets, based on the feature splitting methodology of Algorithm 6.1. The value of K is chosen based on acceptable drop in prediction accuracy, by the system's designer. A larger K value provides for greater flexibility in the design of the framework. It is possible to have high K values for most high dimensional cybersecurity datasets, at a reasonable compromise in accuracy, as seen in Table 6.9 (average K=7 for $<$10% drop in accuracy). The K subsets can be obtained based on: a) Domain specific information, such as in the case of multi-modal biometric systems, b) Correlation based extraction of features, such as described in [192, 193], or by c) Feature ranking based division as described in Algorithm 6.1, in the absence of any specific information about the domain. The obtained set of K feature subsets can then

be used to generate K models M_1 to M_K, which represent the trained model pool for the training dataset. The models can be homogeneous or heterogeneous, so long as they provide a good fit to the obtained set of features, as determined by cross validation on the training data. These pool of classifiers form the basis of the multiple classifier system, from which different policies can be implemented by selectively aggregating model results.

To integrated the benefits of the *Predict-Detect* framework, into a MCS formulation, the obtained pool of K models are split between the *Prediction Ensemble* (PE) and *the Detection Ensemble* (DE). It is not necessary for this split to be uniform and it is suggested to have a higher number of classifier in *PE*, to ensure delay in the onset of attacks and high prediction performance, before attacks. The set of models in the *Prediction Ensemble* are at the adversarial forefront, susceptible to probing based attacks. As such these can be made more secure using the static security mechanisms of Complex learning and Randomization, as proposed in literature. A set of models in *PE* can be randomly selected, and their results aggregated in a conjunctive formulation, to form the final prediction on a given input samples. This will ensure a majority voted based approach in conjunction to randomness, to secure against evasion and reverse engineering attacks. The modular setup of the framework in Figure 6.16, allows for extensibility of the proposed approach to include existing mature work in the area of evasion security of classifiers [40,66], by allowing for their usage in the development of the *Prediction Ensemble*. The reactive component of the attack-defense cycle is handled by the design of the hidden *Detection Ensemble* (DE). By hiding the set of model from the *Prediction Ensemble*, they are shielded from probing based attack and mimicking. This is developed in line with the *Predict-Detect* framework, and as such it aims to provides the same benefits of detect-ability and retrain-ability in *Dynamic-Adversarial* environments. The models in the *Detection Ensemble*, are used to perform detection based on their disagreement with the *Prediction Ensemble* (PE). For the purposes of the detection, each model in the detection ensemble is considered as an independent expert with an orthogonal view of the data. As such, disagreement between any models of the DE, with the overall prediction of the PE, is indicative of drift. The presence

of multiple orthogonal models, also allows for fine grained active learning, where we can prioritize choosing of samples, which are disagree with a majority of the models in the DE.

Once a drift is confirmed and a new training dataset is obtained, the set of models need to be retrained. It was suggested in Section 6.2.2.3, that retraining be done on the entire set of features and that new splits of the models be used to develop the *Predict-Detect* models. However, this design can be susceptible to possible temporal attacks, where an adversary can learn from multiple attack campaigns, to eventually reverse engineer all features of the training data, leading to data nullification attacks over time. To safeguard against this possibility of attacks, we propose the division of the detection ensemble's models into the following two sub sets: a) *The Dynamic Models* (DE-DM) and the b) *Permanently Hidden Models* (DE-PM). The *Dynamic Models* are the set of models, which can crossover to the *Prediction Ensemble*, if the new training data requires additional features (than available in PE), for the next defense cycle. This is especially the case when certain features of the PE model become uninformative over time for the prediction task, leading to un-usability of the same features for future classification of samples. In such a scenario, we can recognize low performing models in the PE (by evaluating on newly obtained labeled samples), and then replace them with a few models of the DE-DM set of models. The evaded models can be held in the system, to test if they become effective at a later attack cycle, to protect against recurring adversarial drifts. The *Permanently hidden Models* (DE-PM), are only used for detection, while never being included in the *Prediction Ensemble*. This ensures that the adversary at no time will have access to all the set of feature information, even if it aggregates information from past attack cycles. Inclusion of a few models in the DE-PM ensures long term insurance against attacks. The division of the models in the detection ensemble, into a permanently hidden category and a set for dynamic substitution, allows for the ability to recover from attacks and provide effective reactive security, while still maintaining long term benefits of feature importance hiding.

The blueprint of the *Predict-Detect* ensemble framework, is a proposed framework for integrating the idea of feature importance hiding into the design of reactive secure

systems. It allows developers and system designers, to integrate the core ideas of this design, while still providing flexibility of using their existing model designs and system architectures. This framework allows for modular extension and testing with several existing and proposed techniques, to best suit one's needs and data characteristics. The *Prediction Ensemble* provides for the ability to apply robust learning methodologies of [6, 71, 72, 98]. While the *Detection Ensemble* provides for the ability to maintain strategies advantages in a reactive environment. Together, the *Predict-Detect* ensemble provides for better security for a *Dynamic Adversarial Environment*, where attacks are expected, and the system is capable of reacting to them efficiently and swiftly. By including considerations for unsupervised drift indication, selective labeling of samples, and misleading attackers to launch low confidence attacks, the proposed framework provides for a generalized way for developers and data scientist, to design for long term security, in adversarial environments. The framework can be made automated, where the feature splitting, retraining and replacing models, can be done dynamically by the system, and the system is capable of requesting human intervention when it deems necessary. This makes it an intelligent dynamic system, which is capable of reacting to adversarial activity, and provides a domain agnostic framework for security. This also allows the framework to be applied as a black box extension to classification models, enabled to provide *'security by design'*. The design of the framework is domain agnostic and assumes a black box view of the defender, further allowing for application specific customization, by system designers.

6.7 Chapter summary

In this chapter, the *Predict-Detect* streaming classifier framework was presented, which is capable of detecting and handling adversarial drifts from streaming data. The framework used a coupled classifier design, where a set of features are shielded from adversarial evasion. This in turn enables an unsupervised mechanism for drift detection and also provides benefits for active learning, in a streaming environment. Experimental evaluation of the framework on 4 datasets, demonstrates that the frameworks is able to indicate drifts

from unlabeled data, and provides performance similar to that of a fully labeled drift detection approach (<5% difference, on average). This detection was possible even when a high confidence attack is launched against the classifier. This demonstrates the advantage of an adversarial aware design, where preemptive design strategies in the design phase, leads to better dynamic security of the system. The inability of traditional drift detection techniques, to account for the adversarial nature of the drifts, causes them to be ineffective and unusable in such environments. This was shown experimentally, specifically for the Margin Density Drift Detection (MD3) algorithm of Chapter 4. The ability to hide feature importance, was seen as an effective extension of the margin based approach of MD3, to dynamic adversarial environments. The proposed framework was shown to provide better unsupervised attack detection, better selection of adversarial samples in imbalanced streaming data, and the ability to perform continuous retraining and usage of the models. The following claims were tested in this chapter:

- ***Claim a:*** *Adversarial agnostic drift detection is vulnerable to evasion.*

  The margin density drift detection algorithm (MD3), was seen to be a reliable indicator of drift from streaming unlabeled data, in Chapter 4. However, since MD3 does not account for adversarial nature of drifts, it is vulnerable in dynamic adversarial domains. The attacker was able to use the information presented by the random subspace ensemble, to launch a high confidence attacks, which avoided the margin and thereby detection from MD3. Therefore, to deal with adversarial drift it is necessary to account for the specific characteristics of this drift.

- ***Claim b:*** *The design of the classification model at training time, can provide benefits for handling test time drifts.*

  In the design of the *Predict-Detect* classifier, it was seen that hiding of the *Detection* model, caused the attacks to fall in the disagreement region of the two classifiers. This allowed for unsupervised detection and also for detection of the minority adversarial samples in imbalanced streams. On the other hand, the presence of all information in

case of the MD3 methodology, caused the attacker to fall outside the margin region, which was undesirable. By changing the space of probe-able information, the proposed framework was able to direct drifts to manageable and detectable regions of space.

- ***Claim c:*** *Hiding feature importance, can provide long term security benefits in terms of detection and recovering from attacks.*

  This was seen for the *Predict-Detect* classifier framework, where adversarial drift was detected on simulated streams, without the need for labeled data. This allowed for an overall reduction in >94% of labeled data, on average. Also, by experimenting on data streams with multiple drift cycles, it was shown that the proposed methodology is able to recover from subsequent drifts and remain effective over time. The ability to use disagreement as an active learning signal, also demonstrated the added benefits of hiding features, in labeling and recovering from attacks.

This chapter established the foundation and provides initial experimental evaluation, for future works in the area of adversarial drift handling systems. It is shown that accounting for the adversarial nature of the problem, is important in designing system to be more effective and efficient. The principles of *Dynamic Adversarial Mining* are embodied by the *Predict-Detect* framework, and it is aimed to serve as a blueprint for designing future systems in its spirit.

CHAPTER 7

CONCLUSION AND POTENTIAL FUTURE WORK

7.1 Conclusion

The naive usage of machine learning in adversarial environments has made it vulnerable to a new gamut of attacks, which are themselves data driven. A static view of the data mining system, where models are learned from the obtained training data and then deployed to provide prediction, is not suitable in an adversarial environment. Such domains present a dynamic landscape, with an ongoing arms race between the system designer and the attackers. Any solution designed for such a domain, needs to take into account an active adversary and be able to evolve over time in the face of emerging threats. In this dissertation, we studied this problem of *Dynamic Adversarial Mining*, to highlight the need for a new paradigm in applying machine learning in the domain of cybersecurity. Existing work on the security of machine learning was shown to be divided into two schools of thought: a) Proactive security measures and b) Reactive security measures. Proactive measures concentrate on delaying the attacks on a system, by bolstering defenses. While, the Reactive security measures focus on handling attacks swiftly, after they occur. An integration of Proactive and Reactive measures was emphasized, for a dynamic view of the security of a system. This integrated view was proposed under the canopy of *Dynamic Adversarial Mining*, where attacks need to be prevented, easily detected and recovered from; to ensure continuous long term security. The major contributions of the dissertation are summarized in the following sections.

7.1.1 White hat analysis of the vulnerabilities of classifiers

Starting with an adversary's view of the machine learning system, the developed work provides an analysis of vulnerabilities of binary classifiers, to probing based exploratory attacks. A data driven framework, called the *Seed-Explore-Exploit* framework, was developed as a white hat analysis of the system's security, to characterize attackers and their capabilities, in Chapter 3. Attacks were modeled, ranging from simple random attacks, to sophisticated reverse engineering of the defender's classifier model. It was seen that, having information only about the feature space, was sufficient to launch an attack, with over 90% accuracy of evasion. This attack was launched without any specific domain information, information about classifiers or about training datasets of defender, on 10 different classification datasets. This highlighted the vulnerability of traditional machine learning systems, in environments where adversaries constantly try to evade the learned model, leading to non-stationarity in the data distribution. Here, the accuracy of the learned model leads to a false sense of security, as it has little meaning if the classifier can be easily evaded. Vulnerability of commercial black box based cloud prediction services, particularly the Google Cloud Platform's prediction service, to probing based attacks, was also demonstrated.

7.1.2 Reliable detection of drifts from unlabeled streaming data

The non-stationarity of data distribution leads to concept drift, which needs to detected in a timely manner, to ensure the effectiveness of the deployed models. However, detecting these changes requires labeled data, which is scarce and expensive in real world streaming data applications. As such, an unsupervised drift detection scheme was introduced in Chapter 4, called the Margin Density Drift Detection (MD3) framework, which detects drifts from unlabeled data and has a low false alarm rate. The MD3 framework monitors changes in the density of uncertain regions (margins), to detect drifts which can cause the performance of a model to degrade. This framework was developed as a model independent, streaming, unlabeled, distribution independent and robust change detector. The MD3 framework, when compared to a fully labeled drift detector, was shown to provide a

comparable performance with 88.3% less labeled data. Also, comparisons with a traditional unlabeled change detector, showed an increased robustness to changes which do not affect classification performance, and a reduced dependence on labeling by 8.3%. This ability of the MD3 framework to incorporate the classification process in the change detection, led to fewer false alarms. This makes it particularly attractive for usage in cybersecurity domains, where labeling is expensive and false alarms can cause trust in the system to deteriorate. The MD3 algorithm addresses a very important issue with unsupervised drift detectors: *Reliability*. Automatic curation of insignificant data distribution changes are performed, in a domain agnostic manner. This ensures applicability to a wide variety of applications, in dynamic environments.

7.1.3 Characterizing effects of classifier design on dynamic adversarial capabilities

Analysis was performed on the impact of the different classifier design strategies, on the adversarial capabilities at test time, in Chapter 5. In dynamic environments, the ability to safeguard training data information was highlighted as the primary objective. Use of excessively informative models were shown to leak information about the training data, which made subsequent retraining and detection of attacks, intractable. Data nullification attacks were presented, where an adversary uses the available probe-able information to launch sophisticated attacks, causing them to become indistinguishable from benign traffic. The vulnerability of existing security mechanisms of - Restrictive one class models, Robust feature bagged models and Randomization based classifier, was demonstrated. The ability to hide feature importance information, as a mechanism to mislead adversaries at test time, was demonstrated to provide long term security benefits in dynamic adversarial environments.

7.1.4 Handling adversarial concept drift from streaming data

Adversarial concept drift was introduced in Chapter 6, as a special case of Concept Drift. The uniqueness and characteristics of this category of drifts was established and a framework was developed for simulating adversarial drifts on real world datasets (*SEE-Stream* framework). The ability of an adversary to evade drift detection was demonstrated. Based on an understanding of the adversarial capabilities (Chapter 3) and reliability requirements of streaming data (Chapter 4), an adversarial aware drift handling framework was presented. This framework, called the *Predict-Detect* classifier framework provides unsupervised drift detection, active learning and maintains retrain-ability, in adversarial domains. The modular design and extensibility of the framework was presented, for easy extension and customization to suit different application needs.

7.2 Potential future work

Our understanding of the capabilities and applicability of machine learning, is still in its budding stages. While early results shows the immense potential of artificial intelligence and data driven learning, it is important to adopt a principled approach in the development of the field, without getting carried away with the hype that surrounds it. Taking into consideration the requirements of a domain and the goals that a machine learning based system hopes to achieve, will lead to better tailored solutions, with greater respect for the user requirements. In this dissertation, we presented the development of machine learning classification paradigms, in dynamic and adversarial environments. We incorporate the adversarial aspects of real world environments into the design of machine learning models, to better analyze its vulnerabilities and possible solutions which can make it more secure and usable. This analysis warrants a new paradigm in the application of machine learning, where the never ending and active nature of an adversary is taken into consideration while designing defensive measures. This work is introduced under a new interdisciplinary field of *Dynamic Adversarial Mining*, which we hope will bring together the research in the fields of Machine Learning, Streaming Data and Cybersecurity, towards a more integrated solution

for secure and life-long learning. Directions for extension and future work, is presented in the following sections.

7.2.1 Extending utility of margin density in streaming data environments

The developed margin density based drift detection algorithm (MD3) provides a mechanism for utilizing the margin density information, to reliably indicate distribution changes in the data. This information can be extended to benefit active learning over imbalanced streaming data, to better recover from drifts. Additionally, selective replacement of classifiers in the robust classifier ensemble can be evaluated for better fine grained response to drifts. The MD3 approach, demonstrated the ability to use the hitherto ignored margin information, intrinsic to the classifier deployed in the streaming environment. Including context of the classification task was seen to benefit drift detection. This paradigm can be extended to benefit other streaming data tasks, such as outlier detection, clustering and active learning. Describing the change to users of the framework, can also be a direction which can benefit from the analysis of the margin characteristics of the drifting data, to increase the adoption of reactive approaches, in the real world.

7.2.2 Handling adversarial activity by preemptive classifier designs

The ability to account for adversarial activity, during the training of the classifier model, was shown to benefit the dynamic aspects of the machine learning based system. While, the *Predict-Detect* classifier design provides one way to improve dynamic response of a system, by hiding feature importance, other approaches can be analyzed in the same spirit. The ability to include randomness into the classification model, to prevent reverse engineering attacks, and to include feature honeypots, to detect evasion of the system, can be included into the system to differentially deal with varied adversarial capabilities, at test time. Prevention of causative attacks, by robust learning methodologies, can also be included in the design of the classifier.

7.2.3 Designing dynamic adversarial aware machine learning frameworks

To account for a holistic view of the adversarial landscape in which machine learning models operate, it is necessary to integrate adversarial awareness into the design of the system. This work primarily focused on the attack detection component of the framework. Effects of causative attacks, to mislead the retraining of the streaming data models, warrants further research. Also, the ability of an adversary to influence the labeling component of the framework, needs analysis. Ideas along the work of Adversarial drift [114] and Adversarial active learning [115], are initial steps towards a comprehensive solution. Dealing with multiple simultaneous adversaries, could also be analyzed. An ensemble framework which integrates clustering with classification, such as the GC3 framework of [102], could be beneficial for such scenarios. Additionally, the extension of the framework to machine learning tasks beyond classification, to tasks such as - unsupervised clustering, recommendation systems, outlier detection, rule induction and data pre-processing, needs to be analyzed using an adversarial aware approach.

7.2.4 Applying dynamic adversarial mining to real world applications

The developed methodologies and strategies, have been presented from a data driven perspective, being agnostic of the domain of application. This generic design makes them applicable for usage in a wide variety of systems. However, submitting probes and getting feedback from the defender's model, is domain dependent. The domain specific implementation characteristics need to be further analyzed, to motivate the ubiquitous inclusion of adversarial awareness in the design of machine learning based systems. Application of the *Predict-Detect* framework, as an automated self adjusting dynamic system, will enable the seamless integration of the approach in most black box machine learning services. Remote black box prediction services, such as Amazon AWS Machine Learning[1] and Google Cloud Platform[2], could include the automated detection and recovery mechanism of the

[1] `https://aws.amazon.com/machine-learning/`

[2] `cloud.google.com/machine-learning`

Predict-Detect framework, to provide for a black box secured classifier capable of dealing with dynamic and adversarial changes. This will ensure that security is provided, without any extra onus to the developer, as a part of the black box service. Developing black box classifier models with feature importance hiding, and deploying as a predictive service with application specific modifications, would make for interesting research for motivating a *Dynamic Adversarial Mining* approach to the development of predictive systems. The domains of cybersecurity can see immediate benefits from such a design, but eventually all systems using machine learning at their core need to be made adversarial aware, so that they are not helpless when faced with nefarious activity, after being deployed in the real world.

The new paradigm of *Dynamic Adversarial Mining*, requires input from several interdisciplinary areas of research, including machine learning, cybersecurity, stream data mining, and human-computer interaction. A combined holistic approach will provide for improved and secure systems, which are *'secure by design'* and are able to continuously operate effectively in the real world.

REFERENCES

[1] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar, "Can machine learning be secure?," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.

[2] Nedim Rndic and Pavel Laskov, "Practical evasion of a learning-based classifier: A case study," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2014, pp. 197–211.

[3] D Sculley, Matthew Eric Otey, Michael Pohl, Bridget Spitznagel, John Hainsworth, and Yunkai Zhou, "Detecting adversarial advertisements in the wild," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 274–282.

[4] Michał Woźniak, Manuel Graña, and Emilio Corchado, "A survey of multiple classifier systems as hybrid systems," *Information Fusion*, vol. 16, pp. 3–17, 2014.

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases*, pp. 387–402. Springer, 2013.

[6] Ibrahim M Alabdulmohsin, Xin Gao, and Xiangliang Zhang, "Adding robustness to support vector machines against adversarial reverse engineering," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 231–240.

[7] Morteza Zi Hayat and Mahmoud Reza Hashemi, "A dct based approach for detecting novelty and concept drift in data streams," in *International Conference of Soft Computing and Pattern Recognition (SoCPaR)*. IEEE, 2010, pp. 373–378.

[8] Tegjyot Singh Sethi and Mehmed Kantardzic, "Data driven exploratory attacks on black box classifiers in adversarial domains," *arXiv preprint arXiv:1703.07909*, 2017.

[9] Tegjyot Singh Sethi and Mehmed Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Expert Systems with Applications*, 2017.

[10] Tegjyot Singh Sethi, Mehmed Kantardzic, and Elaheh Arabmakki, "Monitoring classification blindspots to detect drifts from unlabeled data," in *17th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2016, pp. 142–151.

[11] Tegjyot Singh Sethi and Mehmed Kantardzic, "Dont pay for validation: Detecting drifts from unlabeled data using margin density," *Procedia Computer Science*, vol. 53, pp. 103–112, 2015.

[12] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu, "Cloudy with a chance of breach: Forecasting cyber security incidents," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 1009–1024.

[13] Amal Saha and Sugata Sanyal, "Application layer intrusion detection with combination of explicit-rule-based and machine learning algorithms and deployment in cyber-defence program," *arXiv preprint arXiv:1411.3089*, 2014.

[14] KyoungSoo Park, Vivek S Pai, Kang-Won Lee, and Seraphin B Calo, "Securing web service by automatic robot detection.," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 255–260.

[15] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[16] Tyson Condie, Paul Mineiro, Neoklis Polyzotis, and Markus Weimer, "Machine learning for big data," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 939–942.

[17] Shan Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.

[18] Big Data Working Group et al., "Big data analytics for security intelligence," *Cloud Security Alliance*, 2013.

[19] Varun Chandola, Eric Eilertson, Levent Ertoz, Gyorgy Simon, and Vipin Kumar, "Data mining for cyber security," *Data Warehousing and Data Mining Techniques for Computer Security*, vol. 20, 2006.

[20] Bhavani M Thuraisingham, "Data mining for security applications," in *Intelligence and security informatics*, pp. 1–3. Springer, 2006.

[21] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck, "Learning intrusion detection: supervised or unsupervised?," in *Image Analysis and Processing–ICIAP 2005*, pp. 50–57. Springer, 2005.

[22] Mahdi Zamani and Mahnush Movahedi, "Machine learning techniques for intrusion detection," *arXiv preprint arXiv:1312.2177*, 2013.

[23] Engen Vegard, *Machine Learning for Network Based Intrusion Detection*, Ph.D. thesis, Ph. D thesis, Bounemouth University, 2010.

[24] Hua Tang and Zhuolin Cao, "Machine learning-based intrusion detection algorithms," *Journal of Computational Information Systems*, vol. 5, no. 6, pp. 1825–1831, 2009.

[25] Robin Sommer and Vern Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2010, pp. 305–316.

[26] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck, "A close look on n-grams in intrusion detection: anomaly detection vs. classification," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 2013, pp. 67–76.

[27] Michael Crawford, Taghi M Khoshgoftaar, Joseph D Prusa, Aaron N Richter, and Hamzah Al Najada, "Survey of review spam detection using machine learning techniques," *Journal Of Big Data*, vol. 2, no. 1, pp. 1–24, 2015.

[28] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia, "Exploiting machine learning to subvert your spam filter.," *LEET*, vol. 8, pp. 1–9, 2008.

[29] Seongwook Youn and Dennis McLeod, "A comparative study for email classification," in *Advances and innovations in systems, computing sciences and software engineering*, pp. 387–391. Springer, 2007.

[30] Tony A Meyer and Brendon Whateley, "Spambayes: Effective open-source, bayesian based, email classification system.," in *CEAS*. Citeseer, 2004.

[31] Darryl Felix DSouza, "Avatar captcha: telling computers and humans apart via face classification and mouse dynamics.," 2014.

[32] J Zico Kolter and Marcus A Maloof, "Learning to detect and classify malicious executables in the wild," *The Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[33] Charles Smutz and Angelos Stavrou, "Malicious pdf detection using metadata and structural features," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 239–248.

[34] Chamila Walgampaya and Mehmed Kantardzic, "Cracking the smart clickbot," in *13th IEEE International Symposium on Web Systems Evolution (WSE)*. IEEE, 2011, pp. 125–134.

[35] Long Lu, Roberto Perdisci, and Wenke Lee, "Surf: detecting and measuring search poisoning," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 467–476.

[36] Jun Ho Huh and Hyoungshick Kim, "Phishing detection with popular search engines: Simple and effective," in *Foundations and Practice of Security*, pp. 194–207. Springer, 2011.

[37] Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y Zhao, "Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 239–254.

[38] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara, "Experimental study with real-world data for android app security analysis using machine learning," in *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM, 2015, pp. 81–90.

[39] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[40] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[41] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[42] Davide Maiorca, Igino Corona, and Giorgio Giacinto, "Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, pp. 119–130.

[43] Philippe Golle, "Machine learning attacks against the asirra captcha," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 535–542.

[44] Pedro H Calais Guerra, Dorgival Guedes, J Wagner Meira, Cristine Hoepers, MHPC Chaves, and Klaus Steding-Jessen, "Exploring the spam arms race to characterize spam evolution," in *Proceedings of the 7th Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*. Citeseer, 2010.

[45] César Ferri, José Hernández-Orallo, and R Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009.

[46] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[47] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart, "Stealing machine learning models via prediction apis," in *USENIX Security*, 2016.

[48] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[49] Tao Stein, Erdong Chen, and Karan Mangla, "Facebook immune system," in *Proceedings of the 4th Workshop on Social Network Systems*. ACM, 2011, p. 8.

[50] Philip Kegelmeyer and et. al., "Counter adversarial data analytics," 2015.

[51] Daniel Lowd and Christopher Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 641–647.

[52] Battista Biggio, Giorgio Fumera, and Fabio Roli, "Security evaluation of pattern classifiers under attack," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, 2014.

[53] Charles Smutz and Angelos Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," 2016.

[54] Yujiao Li and Daniel S Yeung, "A causative attack against semi-supervised learning," in *Machine Learning and Cybernetics*. Springer, 2014, pp. 196–203.

[55] Han Xiao, Huang Xiao, and Claudia Eckert, "Adversarial label flips attack on support vector machines.," in *ECAI*, 2012, pp. 870–875.

[56] Daniel Lowd and Christopher Meek, "Good word attacks on statistical spam filters.," in *CEAS*, 2005.

[57] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao, "You are how you click: Clickstream analysis for sybil detection," in *Proc. USENIX Security*. Citeseer, 2013, pp. 1–15.

[58] Indrė Žliobaitė, "Learning under concept drift: an overview," *arXiv preprint arXiv:1010.4784*, 2010.

[59] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 44, 2014.

[60] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar, "Learning in non-stationary environments: A survey," *Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[61] Mohammad M Masud, Clay Woolam, Jing Gao, Latifur Khan, Jiawei Han, Kevin W Hamlen, and Nikunj C Oza, "Facing the reality of data stream classification: coping with scarcity of labeled data," *Knowledge and information systems*, vol. 33, no. 1, pp. 213–244, 2012.

[62] "Email statistics report, 2015 -2019. The Radicati Group Inc., Palo Alto, CA.," 2015.

[63] Paulo M Goncalves, Silas GT de Carvalho Santos, Roberto SM Barros, and Davi CL Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8144–8156, 2014.

[64] Gregory Ditzler and Robi Polikar, "Hellinger distance based drift detection for non-stationary environments," in *IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*. IEEE, 2011, pp. 41–48.

[65] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms for a New Society*, pp. 91–114. Springer, 2016.

[66] Battista Biggio, Giorgio Fumera, and Fabio Roli, "Pattern recognition systems under attack: Design issues and research challenges," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 07, pp. 1460002, 2014.

[67] Blaine Nelson, Battista Biggio, and Pavel Laskov, "Understanding the risk factors of learning in adversarial environments," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 87–92.

[68] Adam Barth, Benjamin IP Rubinstein, Mukund Sundararajan, John C Mitchell, Dawn Song, and Peter L Bartlett, "A learning-based approach to reactive security," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 482–493, 2012.

[69] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye, "An evasion and counter-evasion study in malicious websites detection," in *IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2014, pp. 265–273.

[70] Moritz Hardt, Nimrod Megiddo, Christos Papadimitriou, and Mary Wootters, "Strategic classification," in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. ACM, 2016, pp. 111–122.

[71] Fei Wang, "Robust and adversarial data mining," 2015.

[72] Battista Biggio, Giorgio Fumera, and Fabio Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

[73] David Stevens and Daniel Lowd, "On the hardness of evading combinations of linear classifiers," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 2013, pp. 77–86.

[74] Jun Xu, Pinyao Guo, Mingyi Zhao, Robert F Erbacher, Minghui Zhu, and Peng Liu, "Comparing different moving target defense techniques," in *Proceedings of the First ACM Workshop on Moving Target Defense*. ACM, 2014, pp. 97–107.

[75] Richard Colbaugh and Kristin Glass, "Predictive defense against evolving adversaries," in *IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2012, pp. 18–23.

[76] Yevgeniy Vorobeychik and Bo Li, "Optimal randomized classification in adversarial settings," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 485–492.

[77] Márcia Henke, Eduardo Souto, and Eulanda M dos Santos, "Analysis of the evolution of features in classification problems with concept drift: Application to spam detection," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 874–877.

[78] Leandro L Minku and Xin Yao, "Ddd: A new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, 2012.

[79] Dariusz Brzezinski and Jerzy Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 81–94, 2014.

[80] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli, "Poisoning behavioral malware clustering," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. ACM, 2014, pp. 27–36.

[81] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al., "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 99–108.

[82] Fei Wang, Wei Liu, and Sanjay Chawla, "On sparse feature attacks in adversarial learning," in *IEEE International Conference on Data Mining (ICDM)*. IEEE, 2014, pp. 1013–1018.

[83] Amir Globerson and Sam Roweis, "Nightmare at test time: robust learning by feature deletion," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 353–360.

[84] Wei Liu, Sanjay Chawla, James Bailey, Christopher Leckie, and Kotagiri Ramamohanarao, "An efficient adversarial learning strategy for constructing robust classification boundaries," in *AI 2012: Advances in Artificial Intelligence*, pp. 649–660. Springer, 2012.

[85] Murat Kantarcıoğlu, Bowei Xi, and Chris Clifton, "Classifier evaluation and attribute selection against active adversaries," *Data Mining and Knowledge Discovery*, vol. 22, no. 1-2, pp. 291–335, 2011.

[86] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, and JD Tygar, "Classifier evasion: Models and open problems," in *Privacy and Security Issues in Data Mining and Machine Learning*, pp. 92–98. Springer, 2010.

[87] Battista Biggio, Giorgio Fumera, and Fabio Roli, "Multiple classifier systems under attack," in *Multiple Classifier Systems*, pp. 74–83. Springer, 2010.

[88] Aleksander Kołcz and Choon Hui Teo, "Feature weighting for improved classifier robustness," in *CEAS09: sixth conference on email and anti-spam*, 2009.

[89] Fei Zhang, Patrick PK Chan, Battista Biggio, Daniel S Yeung, and Fabio Roli, "Adversarial feature selection against evasion attacks," 2015.

[90] Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Bowei Xi, "Adversarial support vector machine learning," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1059–1067.

[91] Battista Biggio, Giorgio Fumera, and Fabio Roli, "Adversarial pattern classification using multiple classifiers and randomisation," in *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 500–509. Springer, 2008.

[92] Battista Biggio, Igino Corona, Zhi-Min He, Patrick PK Chan, Giorgio Giacinto, Daniel S Yeung, and Fabio Roli, "One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time," in *Multiple Classifier Systems*, pp. 168–180. Springer, 2015.

[93] Neil C Rowe, E John Custy, and Binh T Duong, "Defending cyberspace with fake honeypots," *Journal of Computers*, vol. 2, no. 2, pp. 25–36, 2007.

[94] Myriam Abramson, "Toward adversarial online learning and the science of deceptive machines," in *2015 AAAI Fall Symposium Series*, 2015.

[95] A Kerckhoffs, "La cryptographie militaire (military cryptography), j," *Sciences Militaires (J. Military Science, in French)*, 1883.

[96] Sasa Mrdovic and Branislava Perunicic, "Kerckhoffs' principle for intrusion detection," in *The 13th International Telecommunications Network Strategy and Planning Symposium, 2008*. IEEE, 2008, pp. 1–8.

[97] Iqra Basharat, Farooque Azam, and Abdul Wahab Muzaffar, "Database security and encryption: A survey study," *International Journal of Computer Applications*, vol. 47, no. 12, 2012.

[98] Richard Colbaugh and Kristin Glass, "Predictability-oriented defense against adaptive adversaries," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2012, pp. 2721–2727.

[99] Kyumin Lee, James Caverlee, and Steve Webb, "Uncovering social spammers: social honeypots+ machine learning," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2010, pp. 435–442.

[100] Reza Shokri, Marco Stronati, and Vitaly Shmatikov, "Membership inference attacks against machine learning models," *arXiv preprint arXiv:1610.05820*, 2016.

[101] Faisal Farooq, Nalini Ratha, Tsai-Yang Jea, and Ruud Bolle, "Security and accuracy trade-off in anonymous fingerprint recognition," in *First IEEE International Conference on Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007.* IEEE, 2007, pp. 1–6.

[102] Tegjyot Singh Sethi, Mehmed Kantardzic, and Hanquing Hu, "A grid density based framework for classifying streaming data in the presence of concept drift," *Journal of Intelligent Information Systems*, vol. 46, no. 1, pp. 179–211, 2016.

[103] Ludmila I Kuncheva, "Classifier ensembles for detecting concept change in streaming data: Overview and perspectives," in *2nd Workshop SUEMA*, 2008, vol. 2008, pp. 5–10.

[104] Burr Settles, "Active learning literature survey," *University of Wisconsin, Madison*, vol. 52, no. 55-66, pp. 11, 2010.

[105] Indre Zliobaite, Albert Bifet, Bernhard Pfahringer, and Graham Holmes, "Active learning with drifting streaming data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 27–39, 2014.

[106] D Sculley, "Online active learning methods for fast label-efficient spam filtering.," in *CEAS*, 2007, vol. 7, p. 143.

[107] Indre Zliobaite, "Change with delayed labeling: when is it detectable?," in *2010 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2010, pp. 843–850.

[108] Karl B Dyer, Robert Capo, and Robi Polikar, "Compose: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 12–26, 2014.

[109] Joseph Sarnelle, Anthony Sanchez, Robert Capo, Joshua Haas, and Robi Polikar, "Quantifying the limited and gradual concept drift assumption," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[110] Rushi Longadge and Snehalata Dongre, "Class imbalance problem in data mining review," *arXiv preprint arXiv:1305.1707*, 2013.

[111] Sheng Chen, Haibo He, Kang Li, and Sachi Desai, "Musera: Multiple selectively recursive approach towards imbalanced stream data mining," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[112] Elaheh Arabmakki, Mehmed Kantardzic, and Tegjyot Singh Sethi, "Rls-a reduced labeled samples approach for streaming imbalanced data with concept drift," in *15th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2014, pp. 779–786.

[113] Hanqing Hu, Mehmed M Kantardzic, and Tegjyot Singh Sethi, "Selecting samples for labeling in unbalanced streaming data environments," in *XXIV International Symposium on Information, Communication and Automation Technologies (ICAT)*. IEEE, 2013, pp. 1–7.

[114] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D Joseph, and JD Tygar, "Approaches to adversarial drift," in *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. ACM, 2013, pp. 99–110.

[115] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Carl Tschantz, Anthony D Joseph, and J Doug Tygar, "Adversarial active learning," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. ACM, 2014, pp. 3–14.

[116] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia, "Tracking concept drift in malware families," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012, pp. 81–92.

[117] Deepak Chinavle, Pranam Kolari, Tim Oates, and Tim Finin, "Ensembles in adversarial classification for spam," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 2015–2018.

[118] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek, "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets," *Pattern recognition*, vol. 36, no. 6, pp. 1291–1302, 2003.

[119] Linda Mthembu and Tshilidzi Marwala, "A note on the separability index," *arXiv preprint arXiv:0812.1107*, 2008.

[120] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[121] Thomas G Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*, pp. 1–15. Springer, 2000.

[122] Dewan Md Farid, Li Zhang, Alamgir Hossain, Chowdhury Mofizur Rahman, Rebecca Strachan, Graham Sexton, and Keshav Dahal, "An adaptive ensemble classifier for mining concept drifting data streams," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5895–5906, 2013.

[123] Marina Skurichina and Robert PW Duin, "Bagging, boosting and the random subspace method for linear classifiers," *Pattern Analysis & Applications*, vol. 5, no. 2, pp. 121–135, 2002.

[124] Alceu S Britto, Robert Sabourin, and Luiz ES Oliveira, "Dynamic selection of classifiersa comprehensive review," *Pattern Recognition*, vol. 47, no. 11, pp. 3665–3680, 2014.

[125] Dymitr Ruta and Bogdan Gabrys, "An overview of classifier fusion methods," *Computing and Information systems*, vol. 7, no. 1, pp. 1–10, 2000.

[126] Tegjyot Singh Sethi, Mehmed Kantardzic, Elaheh Arabmakki, and Hanquing Hu, "An ensemble classification approach for handling spatio-temporal drifts in partially labeled data streams," in *15th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2014, pp. 725–732.

[127] Tejas Joshi, Somnath Dey, and Debasis Samanta, "Multimodal biometrics: state of the art in fusion techniques," *International Journal of Biometrics*, vol. 1, no. 4, pp. 393–417, 2009.

[128] Battista Biggio, Blaine Nelson, and Pavel Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[129] Indranil Palit and Chandan K Reddy, "Scalable and parallel boosting with mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 10, pp. 1904–1916, 2012.

[130] Indranil Palit and Chandan K Reddy, "Parallelized boosting with map-reduce," in *2010 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2010, pp. 1346–1353.

[131] Sun Tzu, "The art of war, translated by samuel b. griffith," *New York: Oxford University*, vol. 65, 1963.

[132] Sergio Pastrana Portillo, "Attacks against intrusion detection networks: evasion, reverse engineering and optimal countermeasures," 2014.

[133] Weilin Xu, Yanjun Qi, and David Evans, "Automatically evading classifiers," .

[134] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, Shing-hon Lau, Steven J Lee, Satish Rao, Anthony Tran, and J Doug Tygar, "Near-optimal evasion of convex-inducing classifiers," *arXiv preprint arXiv:1003.2751*, 2010.

[135] Blaine Nelson, Benjamin IP Rubinstein, Ling Huang, Anthony D Joseph, Steven J Lee, Satish Rao, and JD Tygar, "Query strategies for evading convex-inducing classifiers," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1293–1332, 2012.

[136] Zahid Akhtar, Battista Biggio, Giorgio Fumera, and Gian Luca Marcialis, "Robustness of multi-modal biometric systems under realistic spoof attacks against all traits," in *2011 IEEE Workshop on Biometric Measurements and Systems for Security and Medical Applications (BIOMS)*. IEEE, 2011, pp. 1–6.

[137] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.

[138] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[139] David Wagner and Paolo Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 255–264.

[140] Jie Chen, Bin Xin, Zhihong Peng, Lihua Dou, and Juan Zhang, "Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 3, pp. 680–691, 2009.

[141] Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta, "Phishnet: predictive blacklisting to detect phishing attacks," in *2010 Proceedings IEEE - INFOCOM*. IEEE, 2010, pp. 1–5.

[142] Leyla Bilge and Tudor Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 833–844.

[143] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[144] Liantao Wang, Xuelei Hu, Bo Yuan, and Jianfeng Lu, "Active learning via query synthesis and nearest neighbour search," *Neurocomputing*, vol. 147, pp. 426–434, 2015.

[145] M. Lichman, "UCI machine learning repository," 2013.

[146] Jingrui He and Jaime G Carbonell, "Nearest-neighbor-based active learning for rare category detection," in *Advances in neural information processing systems*, 2007, pp. 633–640.

[147] Bakir Lacevic and Edoardo Amaldi, "Ectropy of diversity measures for populations in euclidean space," *Information Sciences*, vol. 181, no. 11, pp. 2316–2339, 2011.

[148] Chih-Chung Chang and Chih-Jen Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 27, 2011.

[149] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[150] David Haussler, *Probably approximately correct learning*, University of California, Santa Cruz, Computer Research Laboratory, 1990.

[151] Thomas M Cover and Peter E Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[152] Wang Peng Zhu Xiaoyan, "Model selection of svm with rbf kernel and its application," *Computer Engineering and Applications*, vol. 24, pp. 021, 2003.

[153] J Ross Quinlan, "C4. 5: Programming for machine learning," *Morgan Kauffmann*, 1993.

[154] Leo Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[155] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[156] Eduardo J Spinosa, André Ponce de Leon F de Carvalho, and João Gama, "Olindda: A cluster-based approach for detecting novelty and concept drift in data streams," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 448–452.

[157] Mohammad M Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE - TKDE*, vol. 23, no. 6, pp. 859–874, 2011.

[158] Joung Woo Ryu, Mehmed M Kantardzic, Myung-Won Kim, and A Ra Khil, "An efficient method of building an ensemble of classifiers in streaming data," in *Big data analytics*, pp. 122–133. Springer, 2012.

[159] Jing Gao, Wei Fan, and Jiawei Han, "On appropriate assumptions to mine data streams: Analysis and practice," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 143–152.

[160] ES Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[161] Heng Wang and Zubin Abraham, "Concept drift detection for streaming data," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–9.

[162] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues, "Learning with drift detection," in *Advances in artificial intelligence–SBIA 2004*, pp. 286–295. Springer, 2004.

[163] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, 2006, vol. 6, pp. 77–86.

[164] Kyosuke Nishida and Koichiro Yamauchi, "Detecting concept drift using statistical testing," in *Discovery Science*. Springer, 2007, pp. 264–269.

[165] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters*, vol. 33, no. 2, pp. 191–198, 2012.

[166] Albert Bifet and Ricard Gavalda, "Learning from time-changing data with adaptive windowing.," in *SDM*. SIAM, 2007, vol. 7.

[167] Parinaz Sobhani and Hamid Beigy, *New drift detection method for data streams*, Springer, 2011.

[168] Maayan Harel, Shie Mannor, Ran El-Yaniv, and Koby Crammer, "Concept drift detection through resampling," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1009–1017.

[169] Elaine R Faria, João Gama, and André CPLF Carvalho, "Novelty detection algorithm for data streams multi-class problems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 795–800.

[170] Jeonghoon Lee and Frederic Magoules, "Detection of concept drift for learning from stream data," in *14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*. IEEE, 2012, pp. 241–245.

[171] Ludmila I Kuncheva and William J Faithfull, "Pca feature extraction for change detection in multidimensional unlabeled data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 69–80, 2014.

[172] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang, "A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *Proc. of the 21th ACM SIGKDD ICKDDM*. ACM, 2015, pp. 935–944.

[173] Mark Dredze, Tim Oates, and Christine Piatko, "We're not in kansas anymore: detecting domain changes in streams," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2010, pp. 585–595.

[174] Patrick Lindstrom, Brian Mac Namee, and Sarah Jane Delany, "Drift detection using uncertainty distribution divergence," *Evolving Systems*, vol. 4, no. 1, pp. 13–25, 2013.

[175] Anton Dries and Ulrich Rückert, "Adaptive concept drift detection," *Statistical Analysis and Data Mining*, vol. 2, no. 5-6, pp. 311–327, 2009.

[176] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan, "Chernoff-hoeffding bounds for applications with limited independence," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 2, pp. 223–250, 1995.

[177] Stephen H Bach and Marcus A Maloof, "Paired learners for concept drift," in *Eighth IEEE International Conference on Data Mining (ICDM'08)*. IEEE, 2008, pp. 23–32.

[178] Ron Kohavi et al., "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, 1995, vol. 14, pp. 1137–1145.

[179] Sung-Hyuk Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *City*, vol. 1, no. 2, pp. 1, 2007.

[180] Patrick Lindstrom, Sarah Jane Delany, and Brian Mac Namee, "Handling concept drift in text data stream constrained by high labelling cost," 2010.

[181] Hongshik Ahn, Hojin Moon, Melissa J Fazzari, Noha Lim, James J Chen, and Ralph L Kodell, "Classification by ensembles from random partitions of high-dimensional data," *Computational Statistics & Data Analysis*, vol. 51, no. 12, pp. 6166–6179, 2007.

[182] Wlodzislaw Duch, Tadeusz Wieczorek, Jacek Biesiada, and Marcin Blachnik, "Comparison of feature ranking methods based on information entropy," in *2004 IEEE International Joint Conference on Neural Networks*. IEEE, 2004, vol. 2, pp. 1415–1419.

[183] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas, "Tracking recurring contexts using ensemble classifiers: an application to email filtering," *Knowledge and Information Systems*, vol. 22, no. 3, pp. 371–391, 2010.

[184] Ioannis Katakis, Grigorios Tsoumakas, Evangelos Banos, Nick Bassiliades, and Ioannis Vlahavas, "An adaptive personalized news dissemination system," *Journal of Intelligent Information Systems*, vol. 32, no. 2, pp. 191–212, 2009.

[185] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali-A Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[186] Janez Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[187] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank, "Fast perceptron decision tree learning from evolving data streams," *Advances in knowledge discovery and data mining*, pp. 299–310, 2010.

[188] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[189] Malek Ben Salem, Shlomo Hershkop, and Salvatore J Stolfo, "A survey of insider attack detection research," *Insider Attack and Cyber Security*, pp. 69–90, 2008.

[190] Takashi Onoda and Mai Kiuchi, "Analysis of intrusion detection in control system communication based on outlier detection with one-class classifiers," in *Neural Information Processing*. Springer, 2012, pp. 275–282.

[191] Tin Kam Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[192] Zvi Kons et al., "Enhancing decision-level fusion throughcluster-based partitioning of feature set," in *The MENDEL Soft Computing journal: International Conference on Soft Computing MENDEL*, 2014, pp. 259–264.

[193] Mohamed A Aly and Amir F Atiya, "Novel methods for the feature subset ensembles approach," *International Journal of Artificial Intelligence and Machine Learning*, vol. 6, no. 4, 2006.

APPENDIX A

PUBLICATIONS REUSE LICENSES

License Number	4134910822642
License date	Jun 23, 2017
Licensed Content Publisher	Springer
Licensed Content Publication	Springer eBook
Licensed Content Title	'Security Theater': On the Vulnerability of Classifiers to Exploratory Attacks
Licensed Content Author	Tegjyot Singh Sethi
Licensed Content Date	Jan 1, 2017
Type of Use	Thesis/Dissertation
Portion	Full text
Number of figures/tables/illustrations	
Number of copies	1
Author of this Springer article	Yes and you are the sole author of the new work
Order reference number	5024208991
Title of your thesis / dissertation	DYNAMIC ADVERSARIAL MINING - EFFECTIVELY APPLYING MACHINE LEARNING IN ADVERSARIAL NON-STATIONARY ENVIRONMENTS
Expected completion date	Aug 2017
Estimated size(pages)	293
Requestor Location	Mr. Tegjyot Singh Sethi university of louisville LOUISVILLE, KY 40217 United States Attn: Mr. Tegjyot Singh Sethi
Billing Type	Invoice
Billing address	Mr. Tegjyot Singh Sethi university of louisville LOUISVILLE, KY 40217 United States Attn: Mr. Tegjyot Singh Sethi
Total	0.00 USD

Figure A.1: License for reuse of [8].

License Number	4134911070021
License date	Jun 23, 2017
Licensed Content Publisher	Elsevier
Licensed Content Publication	Expert Systems with Applications
Licensed Content Title	On the reliable detection of concept drift from streaming unlabeled data
Licensed Content Author	Tegjyot Singh Sethi,Mehmed Kantardzic
Licensed Content Date	Oct 1, 2017
Licensed Content Volume	82
Licensed Content Issue	n/a
Licensed Content Pages	23
Type of Use	reuse in a thesis/dissertation
Portion	full article
Format	both print and electronic
Are you the author of this Elsevier article?	Yes
Will you be translating?	No
Order reference number	5024208992
Title of your thesis/dissertation	DYNAMIC ADVERSARIAL MINING - EFFECTIVELY APPLYING MACHINE LEARNING IN ADVERSARIAL NON-STATIONARY ENVIRONMENTS
Expected completion date	Aug 2017
Estimated size (number of pages)	293
Elsevier VAT number	GB 494 6272 12
Requestor Location	Mr. Tegjyot Singh Sethi university of louisville LOUISVILLE, KY 40217 United States Attn: Mr. Tegjyot Singh Sethi
Publisher Tax ID	98-0397604
Total	0.00 USD

Figure A.2: License for reuse of [9].

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Figure A.3: License for reuse of [10].

Creative Commons Attribution-NonCommercial-No Derivatives License (CC BY NC ND)

This article is published under the terms of the Creative Commons Attribution-NonCommercial-No Derivatives License (CC BY NC ND).
For non-commercial purposes you may copy and distribute the article, use portions or extracts from the article in other works, and text or data mine the article, provided you do not alter or modify the article without permission from Elsevier. You may also create adaptations of the article for your own personal use only, but not distribute these to others. You must give appropriate credit to the original work, together with a link to the formal publication through the relevant DOI, and a link to the Creative Commons user license above. If changes are permitted, you must indicate if any changes are made but not in any way that suggests the licensor endorses you or your use of the work.

Permission is not required for this non-commercial use. For commercial use please continue to request permission via Rightslink.

Figure A.4: License for reuse of [11].

CURRICULUM VITAE

NAME: Tegjyot Singh Sethi

ADDRESS: Department of Computer Engineering & Computer Science
Speed School of Engineering
University of Louisville
Louisville, KY 40292

EDUCATION:

Ph.D., Computer Science & Engineering
University of Louisville, *Louisville, Kentucky*
August 2017

Graduate Certificate in Data Mining
University of Louisville, *Louisville, Kentucky*
December 2014

M.S., Computer Science
University of Louisville, *Louisville, Kentucky*
August 2013

B.Tech., Computer Science & Engineering
GITAM University, *India*
May 2012

RELATED PUBLICATIONS:

1. **Sethi, T. S.**, & Kantardzic, M. On the reliable detection of concept drift from streaming unlabeled data. Expert Systems with Applications. Elsevier, 2017.

2. **Sethi, T. S.**, & Kantardzic, M. Data driven exploratory attacks on black box classifiers in adversarial domains. arXiv preprint arXiv:1703.07909.

3. **Sethi, T. S.**, Kantardzic, M.,& Ryu, J. Security theater: On the vulnerability of classifiers to exploratory attacks. 12th Pacific Asia Workshop on Intelligence and Security Informatics (PAISI). Springer, 2017.

4. **Sethi, T. S.**, Kantardzic, M., & Arabmakki, E. Monitoring blindspots to detect drifts from unlabeled data. *(Best Paper Awarded)* 17th International Conference on Information Reuse and Integration (IRI 2016) . IEEE, 2016.

5. **Sethi, T. S.**, Kantardzic, M., & Hu, H. A grid density based framework for classifying streaming data in the presence of concept drift. Journal of Intelligent Information Systems, 1-33. Springer, 2016.

6. **Sethi, T. S.**, & Kantardzic, M. Dont Pay for Validation: Detecting Drifts from Unlabeled data Using Margin Density. Procedia Computer Science, 53, 103-112. Elsevier, 2015.

7. **Sethi, T. S.**, Kantardzic, M., Arabmakki, E., & Hu, H. An ensemble classification approach for handling spatio-temporal drifts in partially labeled data streams. In: Proceedings of 15th International Conference on Information Reuse and Integration (IRI 2014), pp. 725-732. IEEE, 2014.

8. Arabmakki, E., Kantardzic, M., & **Sethi, T. S.**. RLS-A reduced labeled samples approach for streaming imbalanced data with concept drift. In: Proceedings of 15th International Conference on Information Reuse and Integration (IRI 2014), pp. 779-786. IEEE, 2014.

9. **Sethi, T. S.**. The GC3 framework: grid density based clustering for classification of streaming data with concept drift. Master's thesis, University of Louisville, 2013.

10. Hu, H., Kantardzic, M. M., & **Sethi, T. S.**. Selecting samples for labeling in unbalanced streaming data environments. In: Proceedings of XXIV International Sym-

posium on Information, Communication and Automation Technologies (ICAT 2013), pp. 1-7. IEEE, 2013.

PROFESSIONAL EXPERIENCE:

- Software Engineering Intern, Google Inc., Mountain View, California, USA

  Worked with the Knowledge Graph team, as a part of Google Search, to improve music actions coverage. Launched music actions for Bollywood actors. Also, worked on third-party data ingestion pipelines.

  May 2016 - August 2016

- Software Engineering Intern, Google Inc., Mountain View, California, USA

  Worked with Google Travel. Designed, developed and deployed an end to end tool in Python which churns huge amounts of metrics data every single day and generates diff reports to highlight data consistency issues. Pushed over 3 KLOC to production with extensive test coverage.

  May 2015 - August 2015

- Graduate Teaching/Research Assistant, University of Louisville, Louisville, Kentucky, USA

  * Research in streaming data mining, never ending learning, ensemble classification, distributed mining using Hadoop and Spark, active learning, crowd sourcing.

  * Instructor for the spring 2016: Survey of CECS graduate level course, which covers Data Structures and Operating Systems concepts.

  * Graduate teaching assistant for the fall 2015/2016 online graduate Data Mining course, responsible for course delivery, grading, project creation and providing assistance with several data science tools. Taught undergraduate introduction to programming languages (CECS 130)- fall 2012, spring 2013 and summer 2013; instructor for online introduction to C (CECS 121) during summer 2013.

* Support for data science tools and analysis, for students in the data mining class/labs; co-supervising master student projects.

* Program Committee member and Reviewer for INNS-BigData 2016; CECS representative for the Graduate Student Council, 2015-2016; Completed the Grant Writing Academy, University of Louisville, 2016; Organizer and co-founder of SpeedUp Startup Program, 2015.

August 2012 - August 2017

HONORS AND AWARDS:

1. Best Student Paper, IEEE-IRI, 2016
2. IEEE Outstanding CECS Student award, Departmental Award, 2016
3. Graduate Student Council's Research Grant, University of Louisville, 2016
4. Travel grant for INNS Big Data Conference, San Francisco, 2015
5. Finalist in the Linkedin Economic Graph Challenge, 2015
6. University of Louisville Doctoral Fellowship award, 2013-2015
7. Winner of HackKentucky, 2014
8. CECS Master of Science, Departmental Award, 2013