NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING NEURAL NETWORKS IN CONSTRAINED OPTIMIZATION PROBLEMS

by

Jan Lim

June 2019

Thesis Advisor: Matthew Norton
Second Reader: Lyn R. Whitaker

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE USING NEURAL NETWORKS IN CONSTRAINED OPTIMIZATION PROBLEMS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jan Lim	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Neural Network (NN) models are typically set up as unconstrained optimization problems with a single objective. However, with NNs seeing widespread adoption across a variety of decision systems, there has been a growth in the need to produce NN models that satisfy important system-specific performance constraints in addition to their primary objective. We consider binary classification in which one must tightly control the false alarm rate while minimizing the false negative rate. Formulating and training NN models in a constrained setting is challenging since most constrained optimization algorithms are not well suited for this setting because the constraint and objective functions are nonconvex, stochastic, and involve potentially millions of parameters. We utilize a new variation of Stochastic Gradient Descent (SGD) called Cooperative-Stochastic Gradient Descent (C-SGD) in an attempt to solve this challenging optimization problem. Application of the C-SGD algorithm is not straightforward, and we explore the effect of its many hyperparameters on performance and efficiency. Overall, we find that C-SGD can be made effective with the right choice of hyperparameters.

14. SUBJECT TERMS neural networks, binary classification, hinge loss, buffered probability, stochastic gradient descent, Neyman-Pearson, false positive, false negative, constrained optimization, machine learning	**15. NUMBER OF PAGES** 97
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

USING NEURAL NETWORKS IN CONSTRAINED OPTIMIZATION PROBLEMS

Jan Lim
Major, Australian Army
BE, University of New South Wales, 2004
MSE, University of New South Wales, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2019

Approved by: Matthew Norton
Advisor

Lyn R. Whitaker
Second Reader

W. Matthew Carlyle
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Neural Network (NN) models are typically set up as unconstrained optimization problems with a single objective. However, with NNs seeing widespread adoption across a variety of decision systems, there has been a growth in the need to produce NN models that satisfy important system-specific performance constraints in addition to their primary objective. We consider binary classification in which one must tightly control the false alarm rate while minimizing the false negative rate. Formulating and training NN models in a constrained setting is challenging since most constrained optimization algorithms are not well suited for this setting because the constraint and objective functions are nonconvex, stochastic, and involve potentially millions of parameters. We utilize a new variation of Stochastic Gradient Descent (SGD) called Cooperative-Stochastic Gradient Descent (C-SGD) in an attempt to solve this challenging optimization problem. Application of the C-SGD algorithm is not straightforward, and we explore the effect of its many hyperparameters on performance and efficiency. Overall, we find that C-SGD can be made effective with the right choice of hyperparameters.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Acronyms and Abbreviations

bFNR Buffered False Negative Rate

bFPR Buffered False Positive Rate

C-SA Cooperative-Stochastic Approximation

C-SGD Cooperative-Stochastic Gradient Descent

DoD Department of Defense

FN false negative

FNR false negative rate

FP false positive

FPR false positive rate

NN neural network

NP Neyman-Pearson

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristics

SGD Stochastic Gradient Descent

SVM Support Vector Machine

UCI University of California Irvine

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Binary classification algorithms generally treat different types of errors (false positives [FP] and false negatives [FN]) as having the same importance. When different error types lead to dramatically different consequences, however, this approach is not appropriate. A better framework is provided by the Neyman-Pearson (NP) paradigm, which seeks to find a classifier that minimizes the False Negative Rate (FNR) subject to a constraint on the maximum allowable False Positive Rate (FPR), also called the false alarm rate. Despite its benefits over alternative cost-sensitive approaches, the NP-paradigm is rarely applied due to the challenges associated with formulating a tractable version of the associated constrained optimization problem and additionally finding a suitable optimization algorithm for solving it. This is further complicated when Neural Network (NN) classifiers are desired. The only available optimization method that has been shown to work consistently for training NN classifiers is Stochastic Gradient Descent (SGD), and this is only suitable for unconstrained optimization problems or those with simple constraints.

To find a classifier in this setting, we propose a tractable constrained optimization problem using a hinge-loss approximation and solve it with a new algorithm called Cooperative-Stochastic Gradient Descent (C-SGD) which is a variation of the Cooperative Stochastic Approximation (C-SA) algorithm originally proposed for expectation constrained stochastic convex optimization problems by Ghadimi et al. (2016). This new optimization algorithm is similar to SGD, relying only upon stochastic gradients, but is well-suited to solve constrained optimization problems. We find that it can be effective for solving the proposed tractable variant of the NP-classification problem directly, particularly when a neural network (NN) classifier is desired and the constrained optimization problem becomes nonconvex and computationally challenging.

While conceptually simple, the C-SGD algorithm is made complex by the number of hyperparameters it requires to be set. These hyperparameters are similar to those present in SGD and we explore their effect on the performance of C-SGD in our NP-classification setting. We utilize data from the University of California Irvine (UCI) machine learning repository (German Credit, Page Blocks, Pima, Spambase and Yeast) to analyze how each parameter affects the performance of the algorithm. In running these tests, we have two

primary questions: 1) When we fix our upper bound on the FPR, does our algorithm produce a NN classifier with FPR below this desired upper bound? 2) As we change this FPR upper bound, do we achieve the expected trade-off between FNR and FPR (with relaxation of FPR upper bound leading to lower FNR)? To evaluate the performance of our algorithm, we will utilize four primary metrics: The error rate, FNR, FPR, and the NP-score (Scott 2007).

Overall, we are able to find hyperparameter settings in which the C-SGD algorithm is able to train a NN with the desired behavior, especially for the Yeast and Spambase datasets. Specifically, we first observe that the found NNs have FPR below the desired FPR upper bound and thus satisfy the desired NP-classification constraint. Secondly, we see that we do gain control over the trade-off between FNR and FPR by relaxing or tightening the FPR constraint.

After developing an instance of the C-SGD algorithm that works for selected benchmark datasets from the UCI machine learning repository, we take a closer look at some important algorithmic components. Firstly, we point out that compared to SGD there is a computational cost associated with enforcing feasibility, meaning the enforcement of the FPR constraint. This is not surprising, since we are tackling a more challenging problem with constraints. We see, however, that the computational cost can be minimized by only checking FPR feasibility approximately, with the number of samples used to estimate feasibility being small enough (batch sizes of approximately 10 to 100) to be cheap computationally but also accurate enough to avoid negative impact upon the overall optimization algorithm.

In this thesis, we attempt to tackle a highly challenging variant of binary classification called NP-classification. The NP-classification setting, while challenging, is an important paradigm for application in which the FNs and FPs have dramatically different costs and it is difficult or unethical to assign actual costs to these types of errors. We find that the C-SGD algorithm is complex, but often can yield good solutions when hyperparameters are chosen carefully. Overall, we are able to show that NN classifiers can be trained via C-SGD in the NP-classification setting. Future work would seek to first utilize more complex NN architectures within the same framework, performing NP-classification and using C-SGD as the optimization method.

References

Ghadimi S, Lan G, Zhang H (2016) Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming* 155(1-2):267–305.

Scott C (2007) Performance measures for Neyman-Pearson classification. *IEEE Transactions on Information Theory* 53(8):2852–2863.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank my thesis advisor, Dr. Matthew Norton, for all the assistance and guidance throughout this journey. Without you, all of this would not have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

There are many methods to perform binary classification such as classification trees, random forests or support vector machines. These classification algorithms typically treat the different types of errors, false positives and false negatives, as having the same importance (Norton and Uryasev 2017). However, this attitude towards errors is often inappropriate. In particular, with decision making systems relying more heavily upon machine learning and classification systems than ever before, it is now critical to consider the system-specific risk associated with misclassification errors and the different consequences that could follow each type of prediction error.

A large literature exists on cost-sensitive classification, which attempts to treat different errors according to their actual associated costs. This approach, however, suffers from fundamental difficulties. Firstly, one must adequately specify the cost associated with each error type. This can often be unethical. Consider, for example, a medical diagnosis task. A false negative (FN) error in the detection of a disease clearly leads to worse consequences than a false positive (FP) diagnosis. However, what is the "cost" of a FN cancer screening? Is it ethical to to say that this FN is 10 times costlier than a FP? What if it were 100 times costlier? Additionally, consider applications relevant to the Department of Defense (DoD). Applications in this realm often involve tasks such as screening for explosives or detecting enemy vessel activity at sea (Cull 2018). It is at best difficult and, at worst, unethical to specify the cost of a FN error that could potentially lead to loss of human life.

In these circumstances, a more appropriate framework for approaching classification is the Neyman-Pearson (NP) paradigm. The NP-classification framework approaches the classification problem as an abstract constrained optimization problem. Specifically, it is the search for a classifier that minimizes the false negative rate (FNR) subject to a constraint on the maximum allowable false positive rate (FPR). For example, it would look something like the following:

$$\begin{aligned} \min_{h \in \mathcal{H}} \quad & \text{FNR}(h, X^+) \\ s.t. \quad & \text{FPR}(h, X^-) \leq \alpha_{fp}, \end{aligned} \tag{1.1}$$

where we are searching for the classifier $h \in \mathcal{H}$ among some set of possible classifiers $\mathcal{H}$ that achieves minimal FNR (the rate of falsely classifying a positive example X^+ as belonging to the negative class) while not having a FPR (the rate of falsely classifying a negative example X^- as belonging to the positive class) larger than a specified upper bound of $\alpha_{fp} \in [0, 1)$.

This framework is much more appropriate for situations in which FNs and FPs lead to very different consequences and "costs" cannot be specified. Instead of specifying costs, we specify an exact upper bound on the FPR, which could be much easier to specify in difficult classification scenarios. For example, the rate of FPs, also called false alarms, is often easier for a subject matter expert to specify than to perform an exact accounting of the "cost" associated with such a false alarm in comparison with the cost of a false negative.

1.1 Problem Statement

Despite its benefits over the cost-sensitive approach, the NP-paradigm is rarely applied due to the challenges associated with actually formulating a tractable version of the associated constrained optimization problem and additionally finding a suitable optimization algorithm for solving it. As discussed further in the Section 2.1.1, the typical procedure for finding a classifier that satisfies such constraints on FPR is to solve the problem indirectly via heuristic means. For example, a common routine is to solve a variation of an unconstrained cost-sensitive classification problem followed potentially by a post-processing step such as threshold selection via Receiver Operating Characteristics (ROC) analysis (James et al. 2013). As discussed in the next section, cost-sensitive unconstrained problems are typically preferred because they are easier to formulate and solve in a mathematical sense. This is particularly true when one wants to utilize neural network (NN) models as the classifier of choice. In this case, the predominant algorithm Stochastic Gradient Descent (SGD) is only suitable for unconstrained optimization problems. The standard method is to "tune" the cost parameters until the desired balance of error rates is achieved on some validation set. The solution found via this method, however, could be vastly suboptimal since it does not solve the problem directly. Additionally, the need to tune the cost parameters leads to the need to solve an unconstrained optimization problem multiple times. This can be computationally prohibitive, particularly with neural networks often requiring hours or days to train with SGD (Goodfellow et al. 2016). Additionally, training NNs is notoriously difficult, with

optimization often proving to be unstable. Thus, even with the addition of different cost parameters, there is no guarantee that it will have the intended affect on FNR and FPR.

1.2 Research Goals

In this thesis, we propose the use of a new optimization algorithm well-suited to solve a tractable variant of the NP-classification problem directly, particularly when a NN classifier is desired and the constrained optimization problem becomes nonconvex and computationally challenging. Specifically, we use Cooperative-Stochastic Gradient Descent (C-SGD), which is a variation of the Cooperative-Stochastic Approximation (C-SA) algorithm originally proposed for expectation constrained stochastic convex optimization problems by Ghadimi et al. (2016). The first goal of this thesis is to formulate a tractable version of the NP-classification problem (1.1). We utilize the average hinge loss as a tractable approximation of the FPR and FNR. This choice is justified by recent research in (Norton and Uryasev 2017) showing that the average hinge loss is effectively equivalent to the use of two new performance metrics called the Buffered False Positive Rate (bFPR) and Buffered False Negative Rate (bFNR). These metrics are probabilistic upper bounds on the FPR and FNR. While a detailed description of these quantities is beyond the scope of this thesis, it provides justification for our use of hinge loss as a good choice of loss function in the constrained NP setting. The second goal of this thesis is to solve the considered optimization problem via C-SGD. This requires development of the algorithm and a discussion of its hyperparameters. We show, critically, that application of C-SGD to our hinge-loss formulation of the NP-classification problem (1.1) is effective. Specifically, we are able to find NNs that satisfy the NP-paradigm, minimizing FNR while satisfying strict upper bounds on FPR.

After developing an instance of the C-SGD algorithm that works for selected benchmark datasets from the University of California Irvine (UCI) machine learning repository, we take a closer work at some important algorithmic components. Firstly, we point out that compared to SGD there is a computational cost associated with approximate checks, at each iteration, of the current solutions feasibility relative to satisfying the FPR constraint. This is not surprising, since we are tackling a more challenging problem with constraints. We see, however, that the computational cost can be minimized by only checking FPR feasibility approximately, with the number of samples used to estimate feasibility being small enough

to be cheap computationally but also accurate enough to avoid negative impact upon the overall optimization algorithm. Specifically, we show that higher accuracy feasibility checks come with diminishing returns and are not worth the computational cost. Secondly, we point out that C-SGD is very similar to SGD in that there exists a large number of hyperparameters and choices that control the performance of the algorithm. In addition to the batch size used to approximately check FPR feasibility, as mentioned before, we explore the effects on performance and algorithm stability of some other design choices. These include, for example, batch sizes for gradient computations, step size choices, and the use of momentum in the updates of gradient descent steps.

1.3 Organization of Thesis

Chapter 2 provides background details and highlights the challenges associated with NP-classification. Chapter 3 presents our specific problem formulation, the proposed C-SGD optimization algorithm, and discusses its design components. Chapter 4 presents our experimental results, first showing that it is effective as a method for finding a NN classifier that satisfies the desired FPR constraints while minimizing the FNR. We utilize five benchmark datasets from the UCI machine learning repository (German Credit, Page Blocks, Pima, Spambase and Yeast) to test the effectiveness of using C-SGD by measuring the error rates, i.e. FNR and FPR. Chapter 4 contains additional analysis and discusses the cost and effect of other design factors that are most important in obtaining useful results. Chapter 5 concludes with an outline of improvements and future work for our method.

CHAPTER 2: Background

Solving the NP-classification problem involves many design choices that are nontrivial. Firstly, one must select a family of classifiers to use, e.g. NNs or linear classifiers. Secondly, one must formulate a tractable variant of the NP classification problem. This involves a proper choice of loss function that appropriately approximates the original FPR constraint. Thirdly, an optimization algorithm must be selected. All of these choices are connected with significant difficulties which we now overview while providing context to motivate our particular choices and highlight the existing challenges in NP-classification. To begin, we require the following preliminaries.

2.1 Problem Setup

We consider a binary classification problem with n-dimensional data points, that are either positively or negatively labeled, i.e. they belong to the +1 class or the -1 class. The set of m^+ examples belonging to the +1 class is denoted by $\{x_1^+, x_2^+, ..., x_{m+}^+\}$. The set of m^- examples belonging to the -1 class is denoted by $\{x_1^-, x_2^-, ..., x_{m-}^-\}$. Let D denote the set of all N examples (x_i, y_i) where y_i are the corresponding labels, let D^+ denote the set of all m^+ positive examples x_i^+, and let D^- denote the set of all m^- negative examples of x_i^-. It follows that $m^+ + m^- = N$.

To find a classifier that can determine if an unlabeled point x belongs to the +1 or -1 class, we will learn a function $h : \mathbb{R}^n \to \mathbb{R}$ that gives each data point a score. The decision rule is to label the data point according to the sign of $h(x)$. In other words, let y_i denote the label given to the example x_i:

$$y_i = \begin{cases} +1 & \text{if } h(x_i) > 0 \\ -1 & \text{if } h(x_i) < 0 \end{cases}$$

If $h(x_i^-) > 0$, a FP, or Type I error, has occurred. If $-h(x_i^+) > 0$, a FN, or Type II error has occurred.

2.1.1 Formulating the NP-Classification Problem

The goal of NP classification is to minimize the FNR while satisfying a constraint that the FPR is below a specified threshold (Scott 2005). While traditional error rate minimization approaches treat different types of error equally, NP classification deals with each error separately, minimizing the FNR while enforcing a strict upper bound on the maximum allowable FPR (Norton and Uryasev 2017). Constraints let us control how much FPs we can tolerate at the expense of minimizing the FNR in the objective function. If we assume the selected classifier family is given by $\mathcal{H}$, the exact formulation would be the following where $I\{\cdot\}$ is the $0-1$ indicator function.

$$\begin{aligned} \min_{h\in\mathcal{H}} \quad & \frac{1}{m^+}\sum_{i=1}^{m^+} I\{-h(x_i^+) > 0\} \\ s.t. \quad & \frac{1}{m^-}\sum_{i=1}^{m^-} I\{h(x_i^-) > 0\} \leq \alpha_{fp}. \end{aligned} \tag{2.1}$$

Even for simple choices of $\mathcal{H}$, such as linear classifiers, this problem is numerically difficult to handle. It involves the challenging $0-1$ loss function which is not only nonconvex, but also discontinuous. Even if this could be optimized, it would not give a very good classifier.

Thus, the first task is to formulate a tractable variant of this problem where we replace the $0-1$ loss with some well-behaved surrogate. However, difficulties immediately arise given the fact that we have a **specific** upper bound on FPR given by α_{fp}. For example, consider the case in which a convex surrogate $\ell(\cdot)$ is used to replace $I\{\cdot\}$ and we have the formulation given by,

$$\begin{aligned} \min_{h\in\mathcal{H}} \quad & \frac{1}{m^+}\sum_{i=1}^{m^+} \ell(-h(x_i^+)) \\ s.t. \quad & \frac{1}{m^-}\sum_{i=1}^{m^-} \ell(h(x_i^-)) \leq R_{fp}. \end{aligned} \tag{2.2}$$

Notice that we now have a new constraint parameter R_{fp} since we may not know how tightly ℓ approximates $I\{\cdot\}$. Thus, we are immediately faced with a significant difficulty. We now need to determine how to appropriately set R_{fp} so that the solution to the tractable problem (2.2) satisfies the original constraint of (2.1) with α_{fp}. Due to this difficulty,

practitioners typically abandon the constrained problem altogether and attempt to tune traditional error-rate minimizers. We will discuss this further in the following sections.

In this thesis, we use the hinge-loss function $[x+1]^+ = \max\{x+1, 0\}$ as a surrogate for the $0-1$ loss function. We do so based on recent research in (Norton and Uryasev 2017) which shows that the hinge-loss is closely connected to probabilistic upper bounds on the the FPR and FNR. A full review of these quantities is beyond the scope of this thesis, but results can be summed up by noting the following. For any fixed classifier $h : \mathbb{R}^n \to \mathbb{R}$ the following inequalities are valid:

$$\frac{1}{m^-}\sum_{i=1}^{m^-} I\{h(x_i^-) > 0\} \leq \min_{a\geq 0} \frac{1}{m^-}\sum_{i=1}^{m^-}[ah(x_i^-)+1]^+ \leq 1,$$

$$\frac{1}{m^+}\sum_{i=1}^{m^+} I\{-h(x_i^+) > 0\} \leq \min_{a\geq 0} \frac{1}{m^+}\sum_{i=1}^{m^+}[-ah(x_i^+)+1]^+ \leq 1.$$

The right side of both inequalities is shown in Norton and Uryasev (2017) to be a probabilistic upper bound of FNR and FPR called the bFPR and bFNR. This means that the hinge-loss, if scaled optimally, is actually equal to a probability. This can be made more precise by considering a general real-valued random variable Z and threshold $z \in \mathbb{R}$. It is discussed in Norton and Uryasev (2017) that Buffered Probability of Exceedance is given by,

$$\bar{p}_z(Z) = \min_{a\geq 0} \mathbb{E}[a(Z-z)+1]^+.$$

In general, we have that $P(Z > z) \leq \bar{p}_z(Z)$. Additionally, $\bar{p}_z(Z)$ equals the $P(Z > \gamma)$ where γ is selected so that $\mathbb{E}[Z \mid Z > \gamma] = z$ for the desired threshold z. The optimization over $a \geq 0$ essentially finds the required γ to make the tail expectation equal to z.[1] Therefore, the hinge-loss, if optimally scaled by $a \geq 0$, equals to the cumulative density in the γ-right-tail of the distribution of Z, where the outcomes in the γ-right-tail have expectation equal to z. In the context of classification, our random variable is a random "error" given by $h(x^-)$ and $-h(x^+)$, the threshold is $z = 0$, and we use empirical sums in place of expectations. Thus, in this case the hinge-loss, if scaled optimally by $a \geq 0$, gives the proportion of largest

[1] If a^* is optimal in the formula for calculating $\bar{p}_z(Z)$, then $\gamma = z - \frac{1}{a^*}$ unless $a^* = 0$, in which case $\gamma < \inf Z$.

"errors" with average value equal to $z = 0$.

Therefore, this justifies the use of the following formulation:

$$\begin{aligned} \min_{h \in \mathcal{H}} \quad & \frac{1}{m^+} \sum_{i=1}^{m^+} [-h(x_i^+) + 1]^+ \\ s.t. \quad & \frac{1}{m^-} \sum_{i=1}^{m^-} [h(x^-) + 1]^+ \leq \alpha_{fp}. \end{aligned} \tag{2.3}$$

Note that we can now justify the use of the originally intended parameter α_{fp} instead of R_{fp}. As we will show, even with the scaling variable a removed, the hinge loss is indeed an extremely effective upper bound to achieve classifier with FPR below the desired threshold α_{fp}.

2.1.2 Choice of Classifier

Although we have formulated the NP classification problem with a more tractable loss function, it remains to choose the classifier family $\mathcal{H}$. This has significant consequences when choosing the optimization algorithm, which we discuss in the next section. In this work we focus on the use of NNs.

A NN typically consists of an input layer, hidden layer and output layer. The hidden layer is where the transformation happens and there can be multiple hidden layers that often have non-linear activation functions. We choose to use a single layer fully-connected NN in this thesis as a starting point to propose a method to solve our binary classification problem. If we are successful in traiing NNs in this setting, it is a proof of concept that more complex NN architectures can be similarly trained.

2.1.3 Optimization

Once a classifier family has been selected, an optimization algorithm must be applied to solve the resulting NP classification problem. Viewed purely in terms of optimization, we

are now faced with a constrained optimization problem abstractly posed as:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ s.t. \quad & g(x) \leq 0. \end{aligned} \tag{2.4}$$

Although constrained optimization forms a rich subfield of optimization in general, major difficulties arise when $g(x)$ is nonconvex and high dimensional, such as is the case when g is our loss function with a NN classifier. Common methods for performing constrained optimization fail because they lack scalability, rely upon second order information, require multiple iterations of solving unconstrained penalty formulations, or require expensive projection operations. For example, many methods in constrained optimization solve multiple iterations of the penalized problem $\min f(x) + \lambda g(x)$ for multiple values of λ. This includes methods based upon the Lagrangian or barrier methods (see e.g. Boyd and Vandenberghe (2004)).

With NNs involved, gradient computations will also be expensive. Thus, stochastic first-order methods are advantageous. However, the most popular of these, SGD, is not suited for constrained optimization. Most variants rely upon an intermediate projection operation, projecting the current iterate onto the feasible region, which cannot be performed feasibly with a nonconvex constraint.

Our choice of optimization method is a variation of mini-batch stochastic approximation from Ghadimi et al. (2016). It crucially relies only upon stochastic gradients and batch-wise computations. Additionally, our method is promising for future NN configurations, given that it can be customized in many of the same ways that SGD is customized (e.g. with adaptive step size choices, acceleration, etc, see Goodfellow et al. (2016)).

2.1.4 Common Heuristic Strategies

With all of the difficulties presented in the previous sections, practitioners rarely solve the NP-classification problem directly or even attempt to solve constrained surrogate formulations as we do in this thesis. With (2.1) yielding a non-convex, discontinuous optimization problem, most current methods for NP classification take a simple approach. Specifically, a common strategy is to tune a traditional error rate minimization algorithm

with "cost" parameters assigning different penalties to FNs and FPs so that the resulting classifier satisfies the NP paradigm. This is often called cost-sensitive classification, with the Support Vector Machine (SVM) of Davenport et al. (2010) providing a good example. This approach, however, may be suboptimal, as it is not solving the problem directly. Additionally, tuning of an error rate minimization algorithm can be computationally expensive. Large grid searches over the space of potential cost assignments are required and care must be taken to provide accurate estimates of the FPR and FNR. As already mentioned in the introduction, assignment of costs in particular applications can also be viewed as unethical.

In addition to this, it is common to take a classifier that is trained to minimize error rates and to perform ROC analysis to determine a decision threshold that satisfies the original constraints on the FPR. This is, however, potentially suboptimal as already mentioned since it is not solving the problem directly, but only tuning a classifier trained on a different objective. The tuning of an error rate minimization algorithm can be computationally expensive. Large grid searches over the parameter space are often required and care must be taken to provide accurate estimates of the FPR and FNR. For example, while Davenport et al. (2010) focus on tuning SVMs, they spend considerable effort to devise their strategy for efficient and accurate cross-validation estimates of FPR and FNR. Additionally, as already mentioned, one may be forced to explicitly indicate the cost of different error types when this type of cost-benefit analysis is unethical (e.g. medical diagnosis).

CHAPTER 3:
Algorithm and Formulation

3.1 Problem Formulation with NN Classifier

Our simple NN classifier is given by:

$$h(x) = w^T \sigma(Wx + B) + b$$

Let n_1 be the number of nodes in the hidden layer, here, we have a hyperplane defined by $(w, b) \in \mathbb{R}^{n_1+1}$ at the last layer and vector $Wx + B$ that is a transformation of our data point x. Specifically, W is a $n_1 \times n$ matrix and B is a n_1-dimensional vector. Our choice of nonlinear activation function is the Rectified Linear Unit (ReLU) given by $\sigma(a) = \max\{a, 0\}$ where for a vector a, this represents the element-wise maximum of a and 0. We will label our point x according to the sign given by $h(x) = w^T \sigma(Wx + B) + b$.

A standard unconstrained optimization problem for binary classification with such a classifier is given by the following:

$$\min_{W,w,B,b} \quad \frac{1}{N} \sum_{(x_i,y_i)\in D} [-y_i(w^T \sigma(Wx_i + B) + b) + 1]^+ \tag{3.1}$$

However, we would like to solve the NP-classification problem. Thus, plugging classifier $h(x)$ into formulation (2.3), we have the following constrained NN optimization problem for finding a NN that satisfies the NP paradigm with upper bound on FPR given by $\alpha_{fp} \in (0, 1)$.

$$\begin{aligned} \min_{W,w,B,b} \quad & \frac{1}{m^+} \sum_{x_i \in D^+} [-(w^T \sigma(Wx_i + B) + b) + 1]^+ \\ s.t. \quad & \frac{1}{m^-} \sum_{x_i \in D^-} [(w^T \sigma(Wx_i + B) + b) + 1]^+ \leq \alpha_{fp} \end{aligned} \tag{3.2}$$

We now have multiple parameters to optimize (W, w, B, b), and the problem is not convex.

Additionally, while we only consider a neural network even with one layer, our optimization problem could be very high-dimensional. Furthermore, we would like to utilize an algorithm that would be feasible for NNs with more than one layer. Thus, we ideally desire an optimization routine that relies only upon first-order gradient information, like SGD, but which works to solve constrained optimization problems. Note that even if we were to use zero layers, and simply use a linear classifier, our proposed algorithm would still be extremely useful as a highly scalable optimization method for constrained optimization that only requires stochastic gradients.

3.2 C-SGD Algorithm

3.2.1 Background: SGD

Our proposed algorithm is an extension of the highly popular SGD algorithm, which is commonly utilized for solving unconstrained optimization problems with objective functions that can be decomposed with respect to individual data points. For example, optimization problem (3.1) can be viewed abstractly as the following unconstrained optimization problem, where we have a function $F(\theta, D) = \sum_{(x,y)\in D} f(\theta, x, y)$ which can be decomposed into individual loss functions $f(\theta, x, y)$ for each data point $(x, y) \in D$ and we optimize over parameters θ:

$$\min_{\theta} \quad F(\theta, D) \tag{3.3}$$

A pseudocode detailing the SGD algorithm is given at the end of this section and requires design choices of batch size and learning rate (also known as step size). Typically, batch sizes are between one to a one hundred (Goodfellow et al. 2016). Learning rates that are too small can lead to slow convergence while learning rates too large make optimization unstable without any convergence. For a single iteration, the SGD algorithm calculates an estimate of the gradient of $F(\theta, D)$ from a mini-batch of examples from the full dataset, instead of calculating the gradient of the real objective function, which can be slow or unreliable (Goodfellow et al. 2016). It then performs a parameter update, moving in the direction of the gradient. These two steps are repeated until a specified convergence criteria is reached. Convergence criteria can include rules such as when the error rate on a validation set is below a certain level, or if a fixed maximum number of gradient steps have been executed.

The SGD algorithm uses the following process. Note that the step size, or learning rate, ν_i can be chosen using a variety of rules and thus we leave it as a generic value chosen at every iteration.

- Step 0: Pick an initial starting point θ_0.
- Step 1.1: Let the current point be θ_i, and let ν_i be some chosen step-size.
- Step 1.2: Take a random sample S from D of size BS_o.
- Step 1.3: Let $F(\theta, S) = \sum_{(x,y)\in S} f(\theta, x, y)$ be the reduced objective function which includes only the examples in mini-batch S. Calculate the estimate of true gradient at the current point using the mini-batch S: $\nabla F(\theta_i, S) = \sum_{(x,y)\in S} \nabla f(\theta_i, x, y)$
- Step 1.4: Take a step in the direction of the negative gradient to get a new point: $\theta_{i+1} = \theta_i - \nu_i \nabla F(\theta_i, S)$
- Step 1.5: If convergence criteria are reached, stop. Otherwise, return to step 1.1 with $i \leftarrow i + 1$ and $\theta_i \leftarrow \theta_{i+1}$.

The SGD algorithm is effective for training large models on large datasets (Goodfellow et al. 2016) but is not suitable for constrained optimization problems.

3.2.2 C-SGD

To solve the constrained optimization problem (3.2), we use the C-SGD method, which is a modification of SGD for optimization problems with objective and constraint that can be decomposed similar to $F(\theta, D)$. The C-SGD method works by first checking if a batch satisfies the feasibility of the constraint at the beginning of the iteration and then chooses to improve either the objective function or constraint function.

Using similar notation as before, we introduce a general objective function and constraint function given by $F^+(\theta, D^+) = \sum_{x_j^+ \in D^+} f^+(\theta, x_j^+)$ and $F^-(\theta, D^-) = \sum_{x_j^- \in D^-} f^-(\theta, x_j^-)$. To detail our algorithm we write our optimization problem as:

$$\begin{aligned} \min_{\theta} \quad & F^+(\theta, D^+) \\ s.t. \quad & F^-(\theta, D^-) \leq \alpha_{fp}. \end{aligned} \tag{3.4}$$

The C-SGD algorithm uses the following process:

- Step 0.0: Initialize maximum number of overall gradient steps T.
- Step 0.1: Initialize $\eta_1, \eta_2, ..., \eta_T$, where $1 \geq \eta_1 \geq \eta_2 \geq ... \geq \eta_T = \alpha_{fp}$.
- Step 0.2: Initialize step sizes $\nu_1, \nu_2, ..., \nu_T$.
- Step 0.3: Initialize batch sizes BS_o, BS_c, BS_f.
- Step 0.4: Pick an initial starting point θ_0.
- For $t = 1, \ldots, T$:
- Step 1: Let the current point be θ_t.
- Step 2: Take random sample S^- from D^- of size BS_f.
- Step 3: If $F^-(\theta_t, S^-) \leq \eta_t$, perform procedure A to improve the objective function. If $F^-(\theta_t, S^-) > \eta_t$, perform procedure B to improve the constraint function.
- Procedure A:
  - Step A.1: Take random sample S^+ from D^+ (the set of positively labeled examples) of size BS_o.
  - Step A.2: Calculate estimate of objective function gradient at the current point using the mini-batch S^+: $\nabla F^+(\theta_t, S^+)$
  - Step A.3: Take gradient step using some step size ν_t: $\theta_{t+1} = \theta_t - \nu_t \nabla F^+(\theta_t, S^+)$
  - Step A.4: If convergence criteria are reached, stop. Otherwise, go back to step 1 with $t \leftarrow t + 1$.
- Procedure B:
  - Step A.1: Take random sample S^- from D^- (the set of negatively labeled examples) of size BS_c.
  - Step A.2: Calculate estimate of constraint function gradient at the current point using the mini-batch S^-: $\nabla F^-(\theta_t, S^-)$
  - Step A.3: Take gradient step using some step size ν_t: $\theta_{t+1} = \theta_t - \nu_t \nabla F^-(\theta_t, S^-)$
  - Step A.4: If convergence criteria are reached, stop. Otherwise, go back to step 1 with $t \leftarrow t + 1$.

C-SGD works on only one class of examples at a time while SGD works on a mix of class examples at the same time. The most costly difference between regular SGD and C-SGD is that C-SGD comes with the cost of approximately checking the constraint at the beginning of every iteration (step 3). Depending on the constraint batch size used, this could become computationally expensive. We will investigate the cost of checking the constraint in Chapter 4.

3.2.3 Critical Components of C-SGD

The SGD and C-SGD algorithm share many design choices, such as step size and batch size choices. However, the C-SGD algorithm requires more design choices which interact in complex ways due to the interaction of the objective and constraint function. Here, we detail these critical choices and comment upon the coming experiments in Chapter 4, where we analyze their potential effect on the success and stability of the optimization procedure.

Batch Sizes for Gradient Computation

Batch size is one of the hyperparameter that has to be selected for C-SGD. We test different batch sizes to determine if this affects the ability of the C-SGD algorithm to choose between improving the objective function or constraint function. Larger batch sizes produce more accurate estimation, but increase computational costs. Note that batches can be selected by random sampling or by randomly partitioning the full dataset. In the algorithm, we see that we have batch sizes of BS_o and BS_c and these are used in the calculation of the gradient at either step A.2 or B.2.

Learning Rate and Momentum

Just like SGD, learning rate (also known as step size) ν_i is an important parameter and can also be combined with many adaptive step size/direction rules such as the use of acceleration (momentum). Using a learning rate that is too low may cause learning to proceed slowly, while a high learning rate may cause instability in the algorithm. Momentum is a commonly used technique which calculates the step direction by use of not only the current gradient, but also past gradients. In our experiments, the learning rate and momentum are chosen by trial and error.

Batch Size for Approximate Feasibility Checks

The most costly difference between SGD and C-SGD is the time taken to approximately check the constraint at the beginning of every iteration. Larger batch sizes should result in a more accurate feasibility check so that the algorithm correctly chooses to improve either the objective function or the constraint function, but this comes at a cost of incurring extra time to perform the feasibility check. We try different batch sizes for the feasibility check to determine the best batch size for BS_f in the algorithm.

Decrement Schedules

A particularly important parameter of C-SGD is the choice of what we call the *decrement schedule*, which is precisely the rate at which η_t decreases from 1 to α_{fp} during the T iterations of the algorithm. The algorithm seeks a balance between improvements made to the objective and constraint function and the primary control over this is the choice of η_t. We will explore how different decrement schedules for η_t affect how the models are trained. We use three different decrement schedules: linear decrement (decreasing η_t by the same amount for $t = 1, ..., k$), exponential decrement (larger decrease at the beginning of the experiment, which then slows down towards the end of the experiment) and a constant schedule (algorithm uses the desired threshold α_{fp} from the beginning). We also experiment with a "burn-in" period, in which set $\eta_t = 1$ for a fixed number of gradient steps. This effectively allows the algorithm to work exclusively on the objective function at the beginning of the algorithm before beginning to enforce constraint feasibility. Figure 1 shows the different schedules we will use in the experiments.

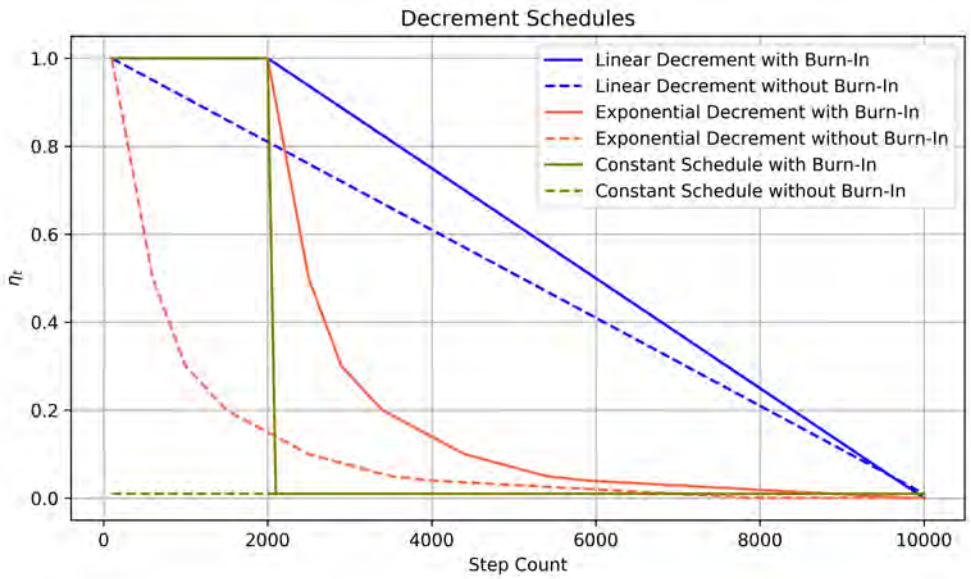


Figure 1. Decrement schedules used in the experiments

The different schedules fit the models differently. For example, we find that the constant schedule tends to overfit the constraint by taking too many gradient steps to achieve $F^- \leq \alpha_{fp}$, while decreasing η_t too slowly will overfit the objective. Thus, a balanced strategy is

required and we find that the experiments show that the most balanced decrement schedule is the linear schedule.

Algorithm Termination Criteria

How do we know when to stop the algorithm? For C-SGD, defining a termination criteria can be complex. We stick to a simple strategy of a fixed number of steps. However, we note that more complex rules can be established. These may, however, make it difficult to analyze other aspects of the algorithm and the effect of other design choices we explore. Therefore, we simply choose our termination criteria to be when a fixed maximum number of gradient steps T have been executed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Experiments

In this Chapter we begin by showing that, indeed, the C-SGD algorithm is an effective optimization approach to solve our considered constrained NN problem. Specifically, we find that it is able to train a NN to satisfy the NP-classification framework, training a NN that minimizes the FNR while satisfying a strict upper bound on the allowable FPR. We focus less on exploring the different components of the C-SGD algorithm and simply illustrate a setting which works to solve this challenging optimization task. We will comment more specifically on these choices in the sections that follow.

After illustrating that the C-SGD algorithm can be made to work with proper choice of hyperparameters, we take a closer look into the effect of some critical design choices on algorithm performance and stability. We begin by exploring one of the most computationally costly design choices: the choice of batch size used to approximate feasibility at every iteration. This aspect of C-SGD is arguably the most important difference between SGD and C-SGD, with the computational complexity of this feasibility check essentially being the extra price one pays over SGD in the constrained setting. We explore optimal choices of the batch size used to check feasibility, finding that increasing batch size comes with diminishing returns. Specifically, we find that too few samples leads to noisy estimates and unstable algorithmic performance, and too many samples beyond a certain point are simply not worth the extra cost in terms of algorithm performance.

Next, we explore the effect of different schedules for η_t, controlling the strictness with which we enforce approximate feasibility $F^-(\theta_t, S^-) \leq \eta_t$ at every iteration. We explore three different schedules for decreasing η_t: a schedule of linear decrements, an exponentially decreasing schedule (larger decrease at the beginning of the experiment, which then slows down towards the end of the experiment) and a constant schedule (algorithm uses the desired threshold α_{fp} from the beginning). We also combine these different schedules with a burn-in period, to determine if the algorithm performs better with η_t held at 1 for a number of iterations before starting the decrement schedule. We find that the linear schedule without a burn-in period performs the best.

Finally, we comment on other design choices: training batch sizes, constraint function learning rate and constraint function momentum. While our exploration here is less systematic than the other section, we point out specific trends noticed in the course of this thesis that may be important in future applications of C-SGD.

4.1 Performance Measures and Experimental Setup

4.1.1 Performance Measures

To evaluate the performance of our algorithm, we will utilize four primary metrics: The error rate, FNR, FPR, and the NP-score. The first three of these have already been introduced in section 2.1. The NP-score is another useful performance metrics for the NP setting introduced by Scott (2007). It gives a weighted score based on FPR and FNR with an extra penalty for violating a constraint on FPR. Specifically, it is given as follows:

$$\frac{1}{\alpha_{fp}}(\max\{\text{FPR} - \alpha_{fp}, 0\}) + \text{FNR} \tag{4.1}$$

Run-time will be considered when we look at the computational cost of checking approximate feasibility. However, we are primarily concerned with the ability of C-SGD to solve our formulation and find a classifier that satisfies the NP-classification criteria: Minimal FNR with FPR no larger than α_{fp}.

4.1.2 Experimental Setup

The algorithm developed for this thesis utilizes PyTorch (PyTorch 2019), a machine learning package for Python. We use a simple NN with one fully connected layer followed by a ReLU transformation and a final linear output layer.

There are multiple parameters that require tuning for the algorithm to produce effective results while taking a reasonable amount of computing time. We utilize data from the UCI machine learning repository (German Credit, Page Blocks, Pima, Spambase and Yeast) to analyze how each parameter affects the performance of the algorithm. Initial analysis included using different decrement schedules for η_t, training batch size, constraint learning

rate and constraint momentum. The thesis will then focus on exploring the effectiveness of the C-SGD method and the price of checking the approximate feasibility of constraints.

We also investigate the effects of using different learning rates and momentum for optimizing the objective function and constraint, but find that there is negligible benefit in tuning these parameters. Performance decreases when learning rate and momentum of the constraint exceed the learning rate and momentum of the objective function respectively. Preliminary analyses on the five benchmark datasets are able to determine the most appropriate learning rates and momentum for the datasets. We therefore use a learning rate of 0.01 and momentum of 0.5 for both objective function and constraint for the remainder of the analysis. Appendix A.2 discusses the choice of the constraint function learning rate while Appendix A.3 explains the importance of the constraint function momentum.

We split the data into training and test sets (80/20 split for each dataset). For the experiments in Section 4.2, we use training batch sizes of 1 and run the algorithm for 10,000 iterations. Appendix A.1 discusses the choice of training batch sizes. The data loader randomly samples batches of positively and negatively labeled data for use in the training and test sets. It also samples batches of negatively labeled data to check for approximate feasibility of constraints in the training model.

The training loop first checks whether a sample of negatively labeled data is approximately feasible relative to η_t. If this is satisfied, the algorithm then performs procedure A (as described in section 3.2.2) to calculate the stochastic gradient of the objective function using a mini-batch from the positively labeled data. If this feasibility is not satisfied, it performs procedure B to calculate the stochastic gradient of the constraint function using a mini-batch from the negatively labeled data.

The testing loop counts the number of correctly and incorrectly classified data on the test set, and records the FNR, FPR, error rate (average of FNR and FPR) and NP score (Scott 2007). We generate plots of error rate, FNR, FPR and NP scores from the final reading for each α_{fp} threshold.

To enable this algorithm to be run on a laptop computer over a reasonable period, we use 10,000 iterations for each experiment. We use decrement schedules as explained in Section 3.2.3. A single experiment with 10,000 iterations will take approximately 20 minutes on a

personal laptop with an Intel 4-core i7 processor and 16 GB RAM.

4.2 Does It Control the False Positive Rate?

We test the algorithm with data from the UCI machine learning repository (German Credit, Page Blocks, Pima, Spambase and Yeast). Using a training batch size BS_c, BS_o of 1 and constraint batch size BS_f of 100, we analyze the performance of the algorithm for various α_{fp} values. The algorithm is set up to start with $\eta_1 = 1$ and to decrease η_t in a linear manner over $T = 10,000$ total gradient steps. This linear rate was implemented by decreasing η_t by $\frac{1-\alpha_{fp}}{100}$ after 100 gradient steps, which we call an epoch. Thus, there are 100 decrements between 1 and the final α_{fp} value. This is repeated for 100 epochs (with 100 steps per epoch) and the final performance statistics on the test set are recorded. We repeat the cycle for a number of α_{fp} thresholds (0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, and 0.99) and generate plots to view the results.

In running these tests, we have two primary questions: 1) When we fix α_{fp}, does our algorithm produce a NN classifier with FPR below this desired upper bound? 2) As we change α_{fp}, do we achieve the expected trade-off between FNR and FPR (with relaxation of α_{fp} leading to lower FNR)? For the first question, we find that our algorithm does often produce a NN classifier that satisfies the desired FPR constraint. Some datasets, such as German Credit, are obviously not well suited for our single layer NN. However, results are hopeful for most of the other datasets and FPR thresholds. For the second question, we see similar results. Specifically, for most datasets we are able to achieve an informative trade-off between FNR and FPR by simply relaxing the FPR constraint and reapplying our algorithm. To illustrate these results, we plot the FNR, FPR, error rate, and NP-score achieved on the out-of-sample test set when our algorithm is trained with various levels of α_{fp}. Note that when NP-score is reported, it has been calculated with the corresponding value of α_{fp} in equation (4.1) in Section 4.1.1. For example, in Figure 2, the x-axis provides the value of α_{fp} used to both train the classifier and to calculate the NP-score.

Figure 2 shows the results for the Yeast dataset.

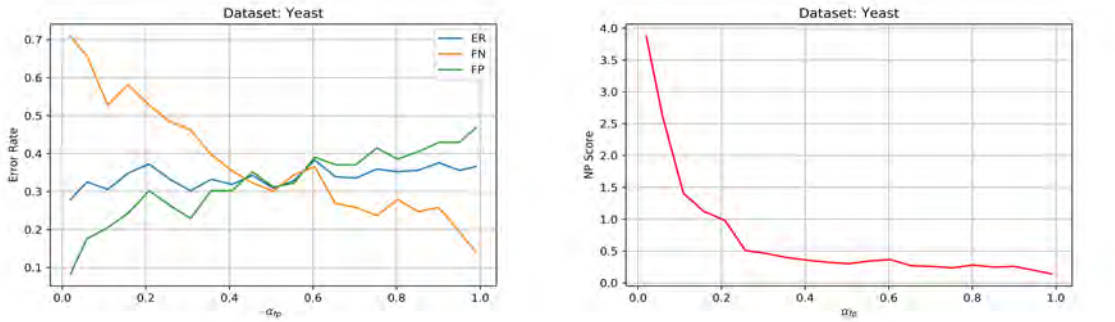


Figure 2. Yeast dataset results for various α_{fp} parameter

For the Yeast dataset, we successfully show that the algorithm is able to control the FNR by relaxing the control on the FPR. The algorithm is also able to find a NN classifier that almost always achieves a FPR that is below the desired upper bound given by α_{fp}. For α_{fp} values above 0.5, we see that the FNR is always lower than FPR.

Figure 3 shows the results for the Spambase dataset.

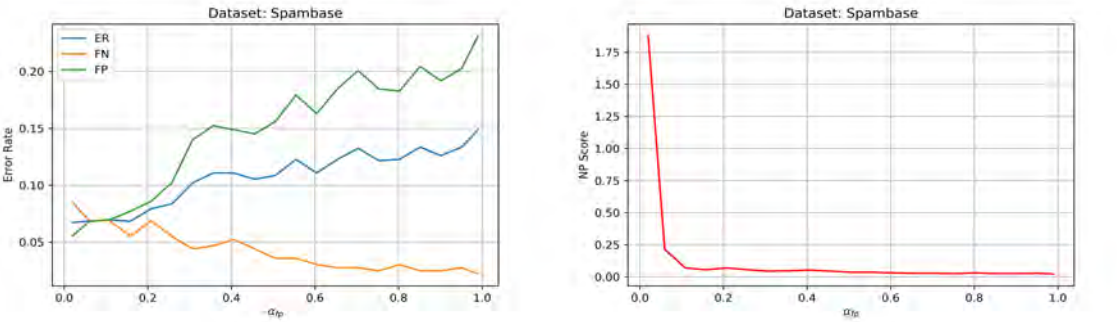


Figure 3. Spambase dataset results for various α_{fp} parameter

We also obtain good results for the Spambase dataset as we can see that the algorithm is able to control the FNR when the constraint on the FPR is relaxed. The measured value of the FPR on the test set almost always falls below the desired upper bound given by α_{fp}.

Figure 4 shows the results for the Pima dataset.

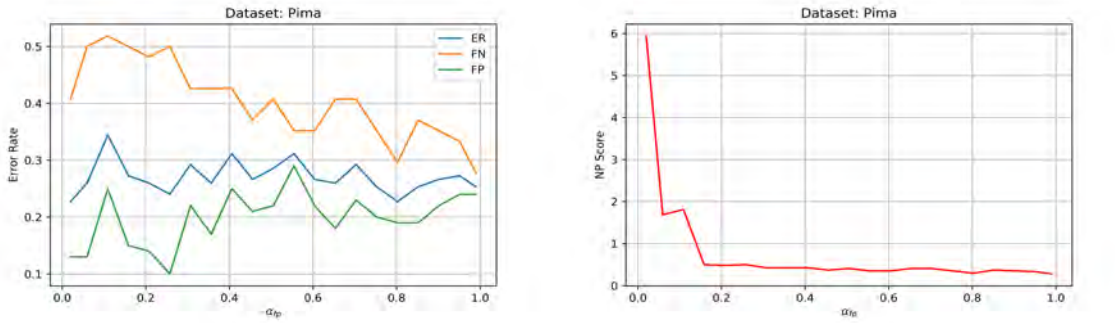


Figure 4. Pima dataset results for various α_{fp} parameter

The Pima dataset shows improvement in the FNR as we relax the control of the FPs. It also is almost always able to achieve a FPR below the desired upper bound given by α_{fp}. It is interesting to note the behavior of the algorithm when $\alpha_{fp} > 0.3$. While the FPRs do not change significantly, the FNR is clearly decreasing as α_{fp} is relaxed. Thus, this means that the algorithm was essentially able to find better solutions when α_{fp} was relaxed enough. This deserves further investigation in future work, but shows that while the algorithm is able to control the FPR so that it stays below α_{fp}, it can be difficult to control the optimality of the classifier with respect to minimizing the FNR, or objective function.

Figure 5 shows the results for the Page Blocks dataset.

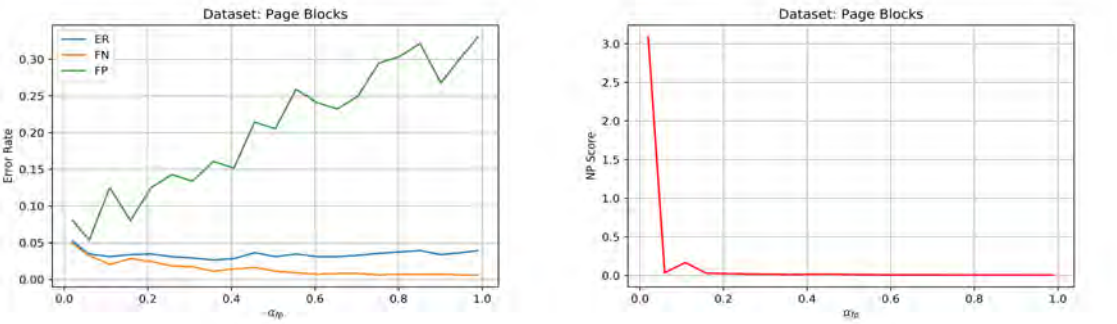


Figure 5. Page Blocks dataset results for various α_{fp} parameter

For the Page Blocks dataset, results are optimistic when evaluating the ability of the FPR constraint to control the FNR. Indeed, we see that the measured value of the FPR on the test set almost always falls below the desired upper bound given by α_{fp}. We also see that relaxation of the FPR constraint does, in fact, lead to a decrease in the FNR, which is the

desired behavior. Although the trade-off is steep for this dataset, the ability to achieve such solutions is informative similar to the way ROC analysis is informative.

Figure 6 shows the results for the German Credit dataset. Note that the x-axis represents the final cycles at the respective α_{fp} thresholds.

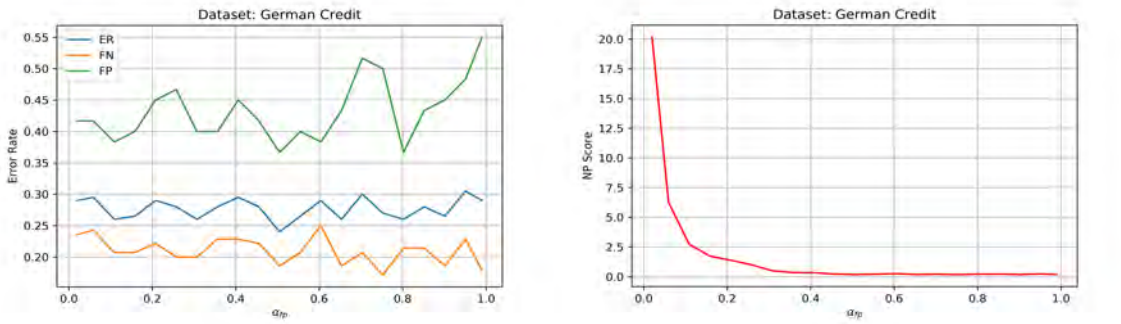


Figure 6. German Credit dataset results for various α_{fp} parameter

There appears to be no clear trend of improvement in the FNR for the German Credit dataset as we relax the control on FPs. We suspect that this dataset is not well suited for a single layer NN classifier and that other classifier families would perform better. This, however, is left for future work.

Overall, the algorithm achieves the desired behavior, especially for the Yeast and Spambase datasets, in which we 1) observe that we are able to reduce the FNR by relaxing control on the FPR and 2) observe that we are able to keep the FPR below the desired α_{fp} upper bound. We can see clearly that there are tradeoff points in the plots for the Yeast and Spambase at which FNRs drop below FPRs as we relax the desired upper bound given by α_{fp}. The Page Blocks dataset showed negligible improvement in FNRs when the FPR thresholds are relaxed as FNRs were already low even with tight control on FPRs. The German Credit dataset does not show a clear trend of improvement for the FNR when we relax the control on the FPR, indicating that the NN classifier that we used might not suitable for this dataset.

4.3 The Price of Constraints

We can see from the analysis that our method of using C-SGD works well for achieving classifiers that are feasible in a NP setting, but this comes with the price of having to

estimate, using a batch of training examples, if the constraint is satisfied at the start of every step in the training loop. If the constraint is not satisfied, the algorithm then proceeds to improve on the constraint function. Checking the constraint incurs additional time in the algorithm compared to regular SGD. Depending on the constraint batch size BS_f used, this could become computationally expensive. Additionally, we must balance the accuracy of the approximation with the computational cost. Too few samples implies inaccurate approximation, which could lead to diminished performance. Using too many samples, however, may have diminishing returns in terms of performance improvements versus computational cost. To investigate the role of BS_f in performance and computational cost of C-SGD, we set up the algorithm to record the amount of time taken to check feasibility and compare the computational cost against the overall performance in terms of the usual FPR, FNR, and NP-score metrics. We limit our experiments to a single α_{fp} per dataset, focusing on values of α_{fp} that yielded feasible classifiers for that selection of α_{fp}.

4.3.1 Batch Size (B) versus Run-time of Training (T) and Final Solution Performance (P)

We conduct experiments using different batch sizes (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096) for checking the constraint to find the most appropriate batch size to use while keeping computational time reasonable. We use the same five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast). To keep the experiment achievable within a reasonable time using a laptop computer, we reduce the number of epochs in the training loop to 10, and also the number of gradient steps per epoch to 10.

Figure 7 shows the results of varying constraint batch sizes for the Yeast dataset, using $\alpha_{fp} = 0.5$.

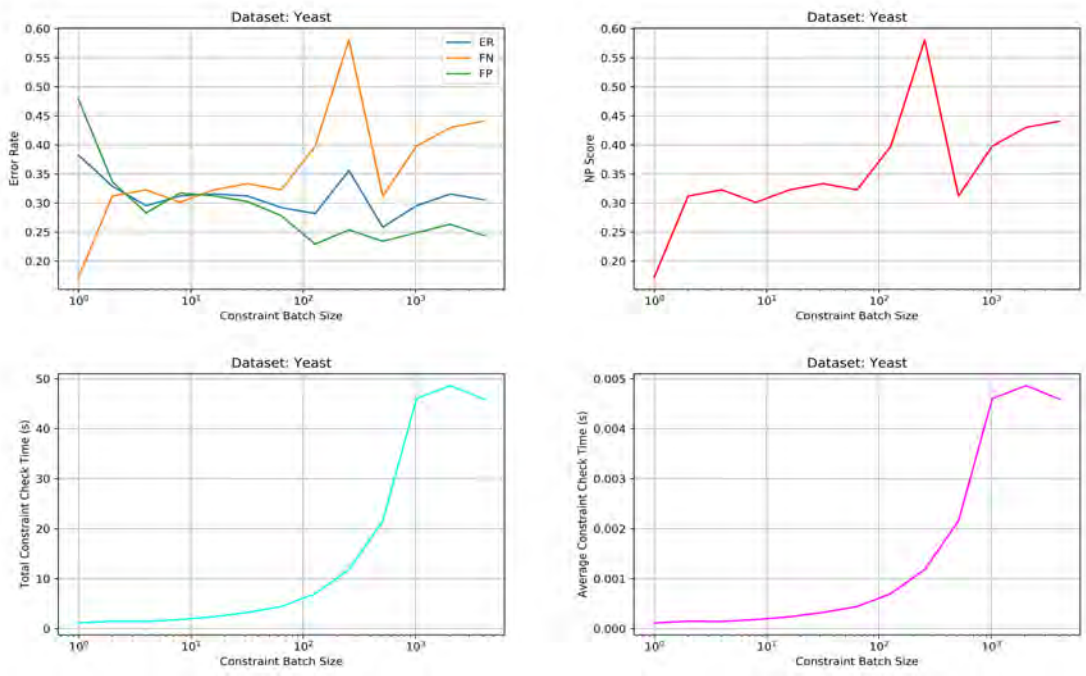


Figure 7. Yeast dataset results for various constraint batch sizes, $\alpha_{fp} = 0.5$

The Yeast dataset shows the best FNR and FPR when constraint batch sizes of between 4 and 64 are used. Interestingly, we see that larger batch sizes cause the algorithm to find more conservative solutions in terms of making sure that the constraint is satisfied and FPR is smaller than α_{fp}. This makes sense, with larger batches leading to more precise estimates of feasibility during optimization. We suspect also that the noise from small batch size estimates of feasibility led to more evaluations as "feasible" and thus more gradient steps taken to improve the objective. Using constraint batch sizes of larger than 100 does not improve the performance as this causes the FNR to increase while also increasing the total time taken to check constraints. The diminishing returns from increases in batch size are also reflected in the NP-score. As batch size increases, the performance in terms of this summary metric actually decreases, reflecting the fact that although feasibility is maintained relative to α_{fp}, there is a price being paid in terms of the FNR for a more conservative solution.

Figure 8 shows the results of varying constraint batch sizes for the Spambase dataset, using $\alpha_{fp} = 0.1$.

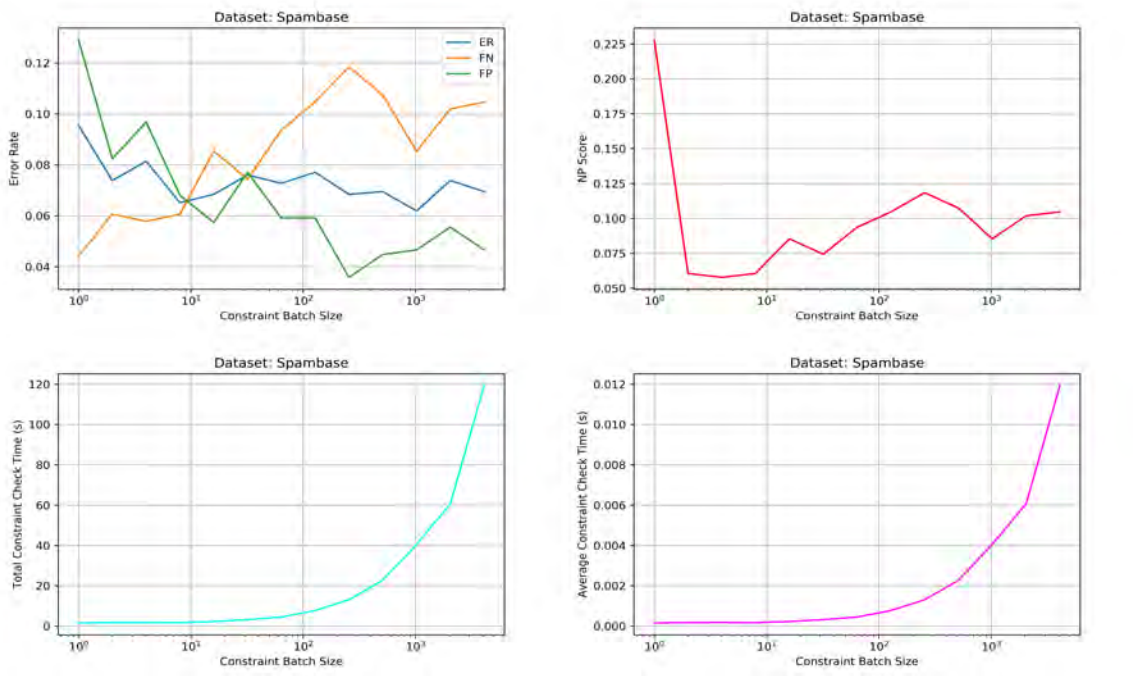


Figure 8. Spambase dataset results for various constraint batch sizes, $\alpha_{fp} = 0.1$

The Spambase dataset shows the best FNR and FPR when a constraint batch size of 10 is used. Again, we see that larger batch sizes lead our algorithm to find more conservative solutions in terms of FPR being below α_{fp}. A price, however, is paid for this conservatism in terms of FNR. Using constraint batch sizes of larger than 10 does not improve the performance as this causes the FNR to increase while also increasing the total time taken to check constraints. We also notice similar behavior to the Page Blocks experiments where larger batch sizes typically lead to more conservative solutions, with the FPR going farther and farther below the desired α_{fp} threshold.

Figure 9 shows the results of varying constraint batch sizes for the Page Blocks dataset, using $\alpha_{fp} = 0.05$.

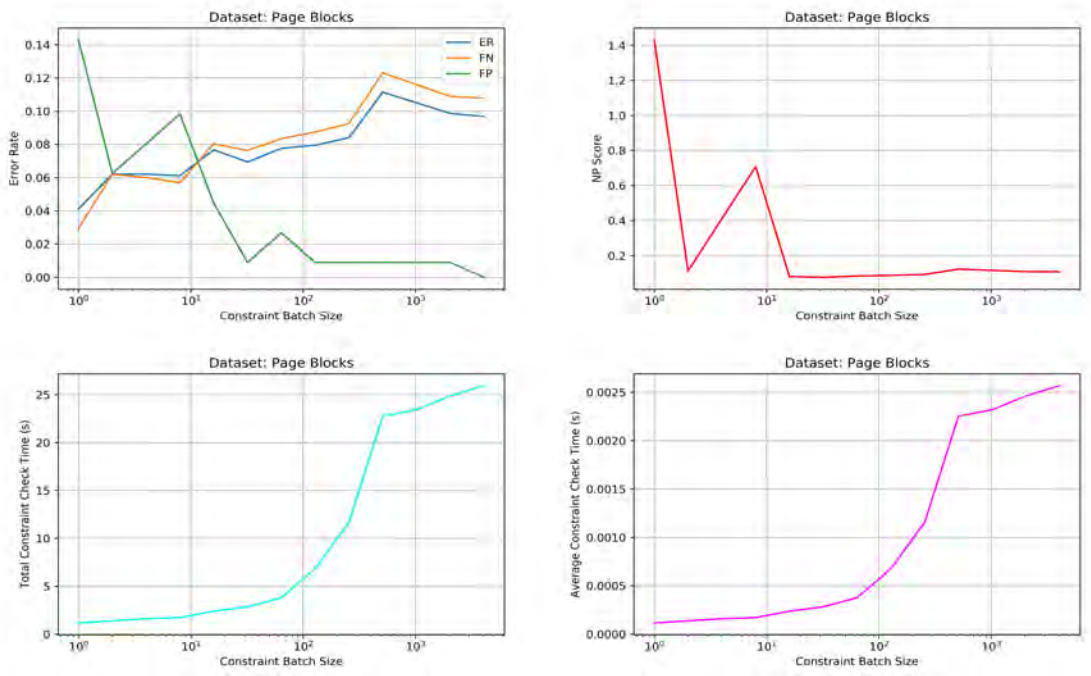


Figure 9. Page Blocks dataset results for various constraint batch sizes, $\alpha_{fp} = 0.05$

The Page Blocks dataset shows the best FNR and FPR when a constraint batch size of 2 is used, although this did not satisfy the α_{fp} threshold of 0.05 by a small margin. Again, we witness that larger batch sizes lead to unnecessarily conservative solutions, with FPR farther below αfp than is required at the price of increasing FNR. In order to satisfy the α_{fp} threshold of 0.05, we see that we need to use constraint batch sizes of larger than 10, but beyond this does not improve the performance as it causes the FNR to increase while also increasing the total time taken to check constraints.

Figure 10 shows the results of varying constraint batch sizes for the Pima dataset, using $\alpha_{fp} = 0.8$.

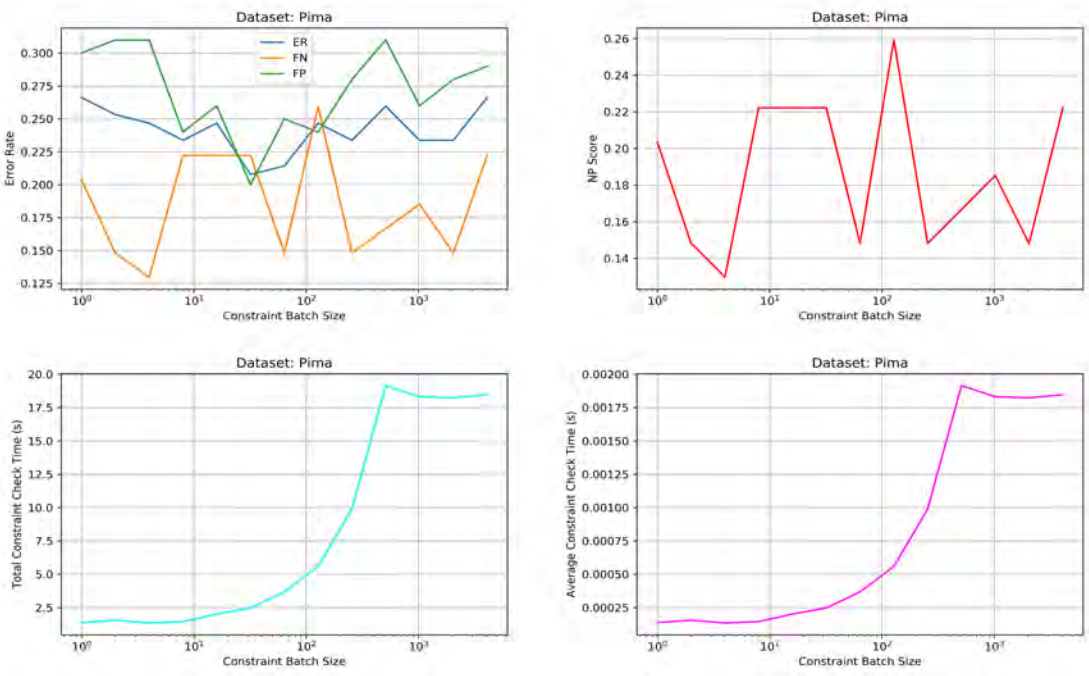


Figure 10. Pima dataset results for various constraint batch sizes, $\alpha_{fp} = 0.8$

The Pima dataset shows that using constraint batch sizes of between 10 and 100 produces the best performance for the FNR and FPR.

Figure 11 shows the results of varying constraint batch sizes for the German Credit dataset, using $\alpha_{fp} = 0.5$.

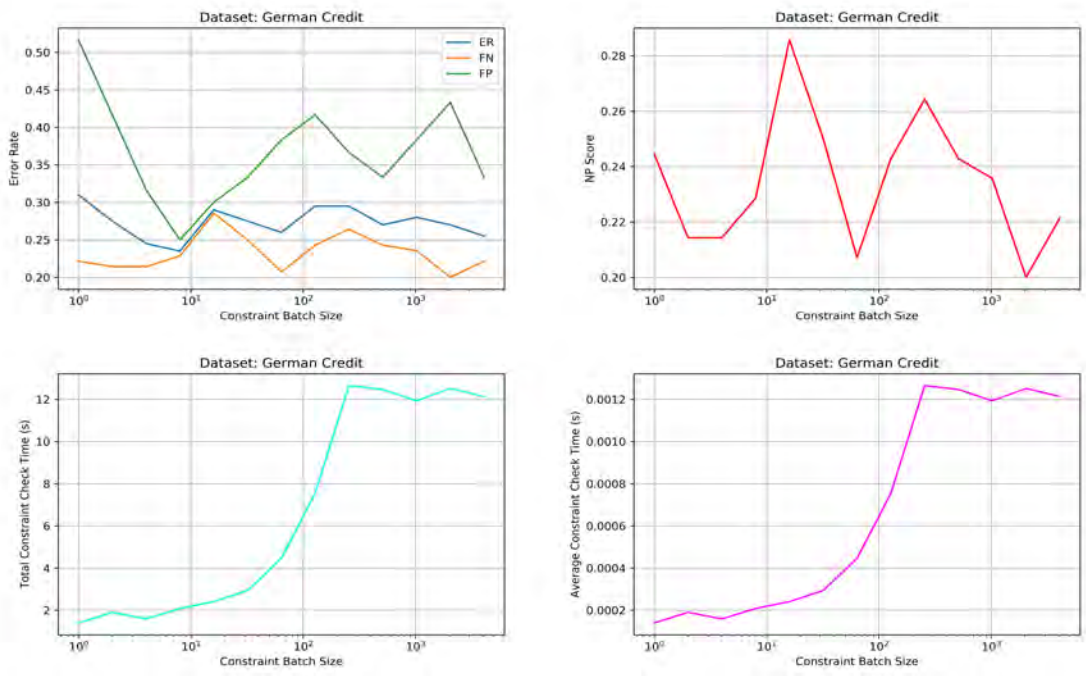


Figure 11. German Credit dataset results for various constraint batch sizes, $\alpha_{fp} = 0.5$

This dataset is difficult since the performance from Section 4.2 was inconsistent. We see that the best performance statistics, in terms of NP-score, are achieved with larger batch sizes, unlike the other datasets. However, we do see that there are clearly diminishing return for using constraint batch sizes larger than 100, with overall solution quality not showing large improvement.

From the results of all the datasets, we can see that, in general, having a constraint batch size BS_f of approximately 10 to 100 gives the best performance statistics while keeping the total time spent on checking the constraints reasonable. Using constraint batch sizes of above 100 does not significantly improve performance statistics and in some cases leads to overly conservative solutions with the FPR far below α_{fp} and increases the FNR (Page Blocks, Spambase and Yeast). It is also interesting to note the role of noise in the behavior of the algorithm. With large batch sizes, and thus low noise, the algorithm is very conservative and seemingly overfits the constraint function. Adding sufficient noise, however, by using a smaller sample size when checking the constraint enables the algorithm to obtain better solutions with smaller FNR while still satisfying FNR$\leq \alpha_{fp}$. This heuristic argument is similar to that made in favor of SGD on nonconvex problems. Using noisy estimates of

the gradient allow one to escape poor local optima and find better solutions. We see that this principle applies in the case of C-SGD for the batch size used to check approximate feasibility. We do find, however, that batch sizes that are too small are simply too noisy and can lead to instability of the algorithm. More investigation is required in this direction. We also note that with the exception of Page Blocks and Spambase, the graphs of average constraint check time appear to flatten out for large sample sizes. This is because the batch sizes are now larger than the entire datasets (for Yeast, Pima and German Credit) and the algorithm is using the entire datasets, and therefore no increase in average constraint check time.

4.3.2 Schedule

In addition to the design choices already discussed in the previous sections, we implement different schedules of decreasing η_t during the experiments. The experiments in the previous sections use a linear schedule (equal decrements for a total of 100 decrements) when decreasing η_t from 1 down to the α_{fp} threshold. We now explore other decrement schedules such as an exponential decrement and a constant schedule (all the iterations are performed at the α_{fp} threshold). We also use a burn-in period of 2,000 steps to determine if this would improve the results for each of the different schedule.

The experiments are repeated for the same five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast) using the different decrement schedules, as outlined in Section 3.2.3.

Figures 12 to 14 show the results for the Yeast dataset.

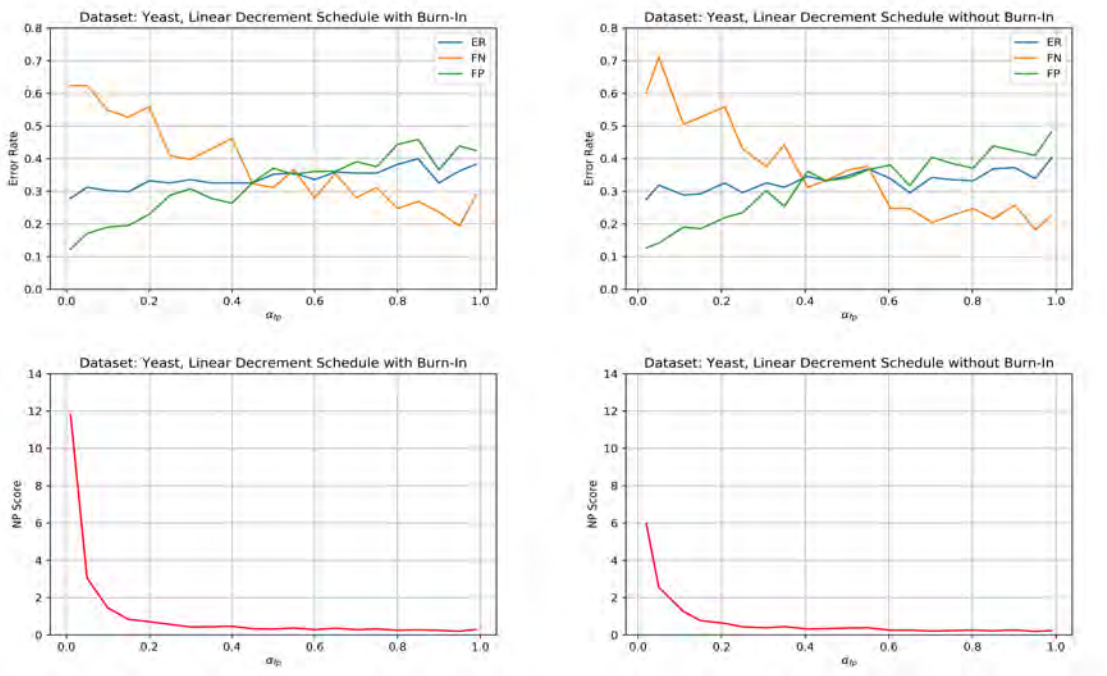


Figure 12. Yeast dataset results using a linear schedule for decreasing η_t, with and without burn-in periods

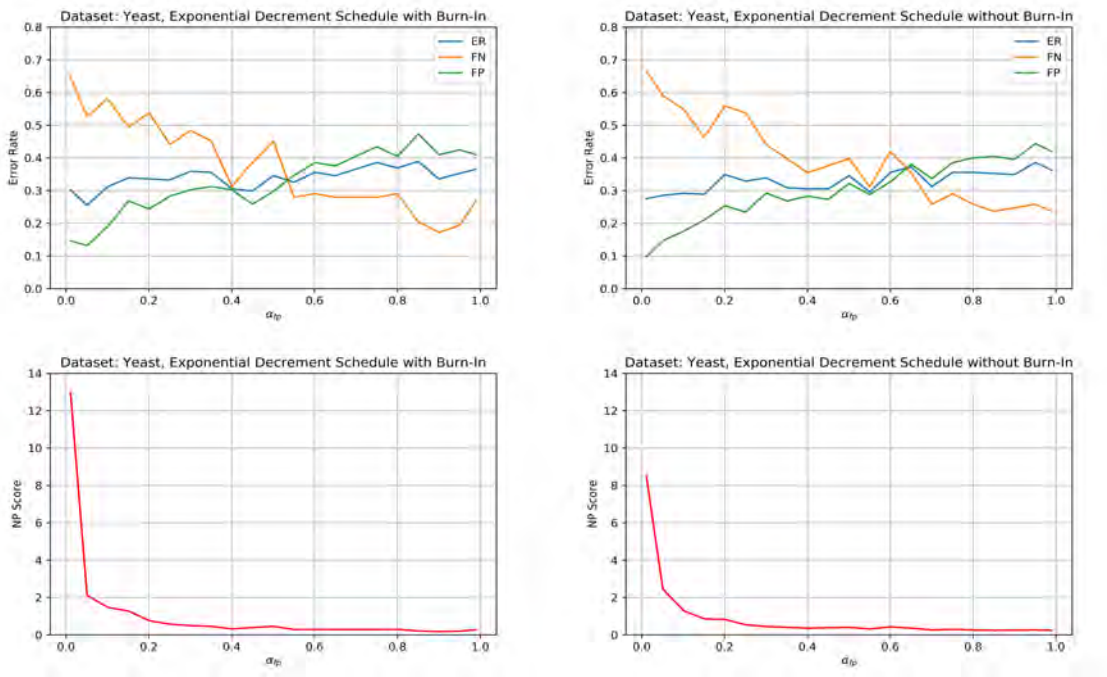


Figure 13. Yeast dataset results using an exponential schedule for decreasing η_t, with and without burn-in periods

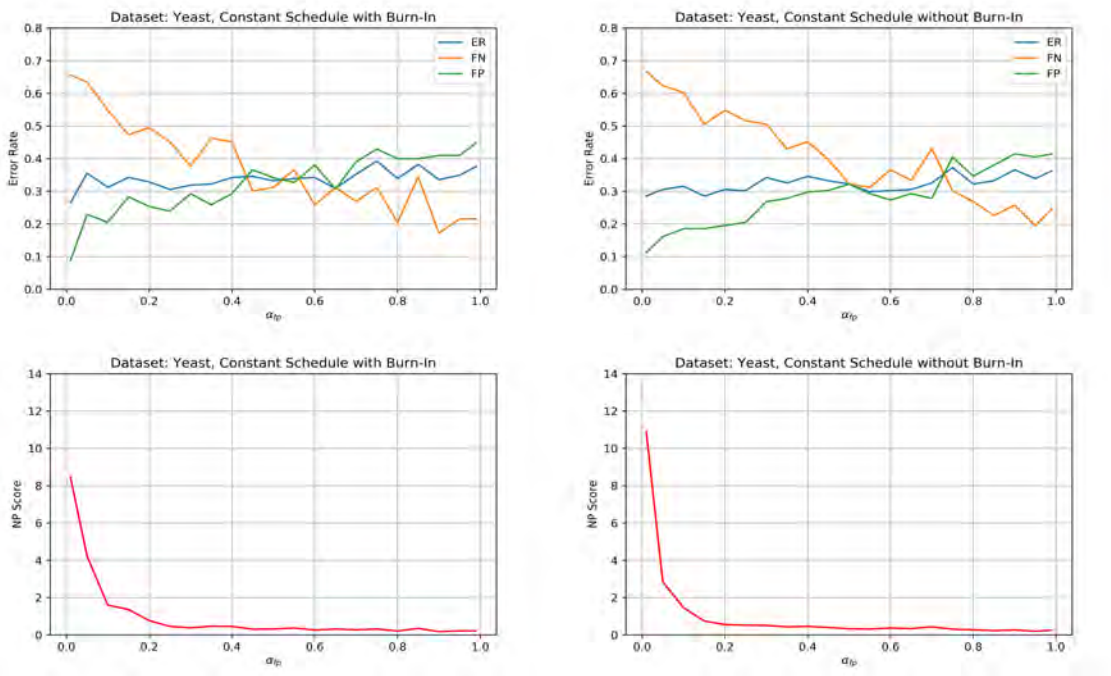


Figure 14. Yeast dataset results using a constant schedule for η_t, with and without burn-in periods

From Figures 12 to 14, the linear schedule without burn-in period performs the best for lower α_{fp} values, as evident from the lower NP score. The linear schedule without burn-in period was able to find a NN classifier that almost always achieves a FPR that is below the desired upper bound given by α_{fp}. It is interesting to note that almost all the schedules appear not to satisfy the α_{fp} threshold of 0.2 or less. We also see that the linear schedule without burn-in is achieves the trade-off point where the FNR is lower than the FPR for α_{fp} value of 0.4. This leads us to conclude that the linear schedule works best for the Yeast dataset.

Figures 15 to 17 show the results for the Spambase dataset.

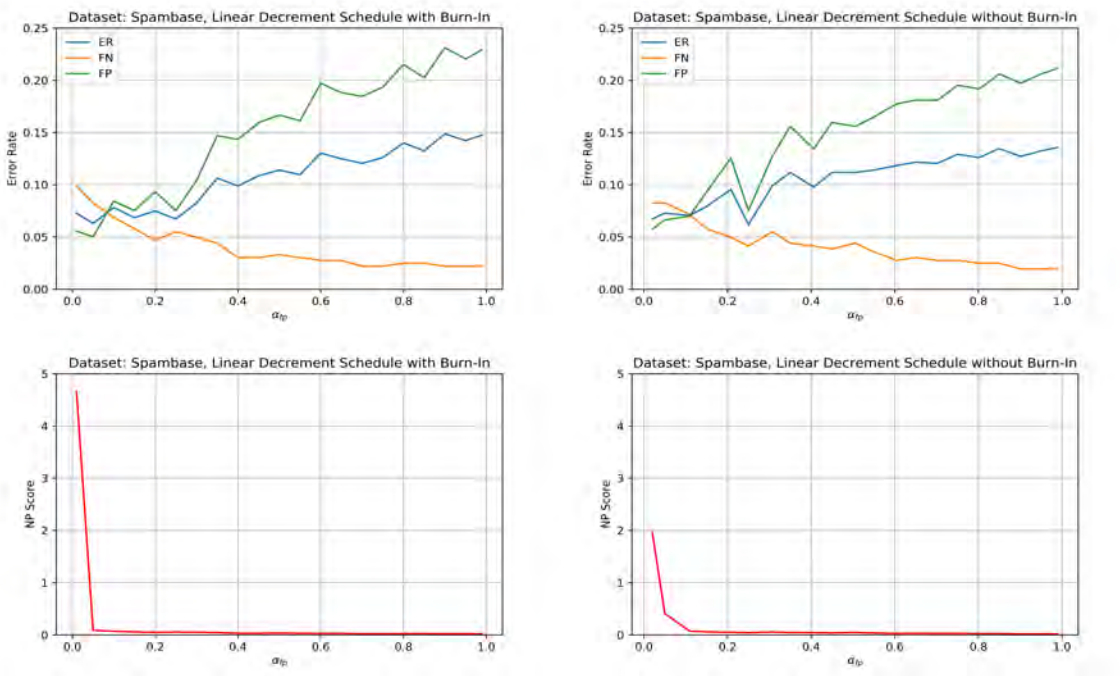


Figure 15. Spambase dataset results using a linear schedule for decreasing η_t, with and without burn-in periods

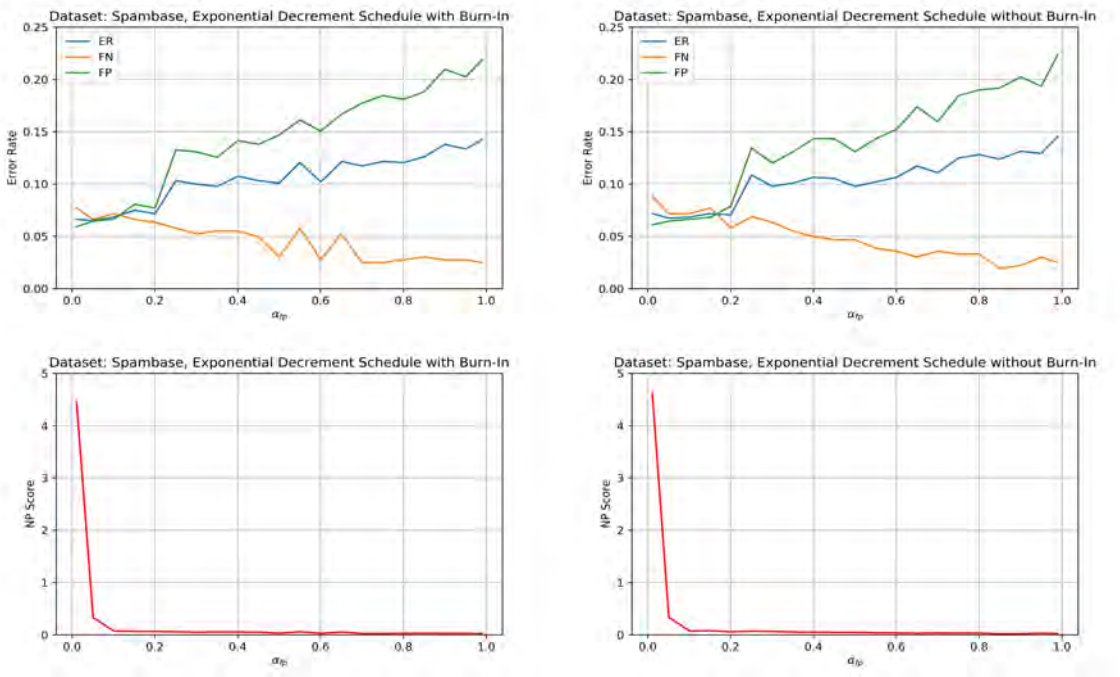


Figure 16. Spambase dataset results using an exponential schedule for decreasing η_t, with and without burn-in periods

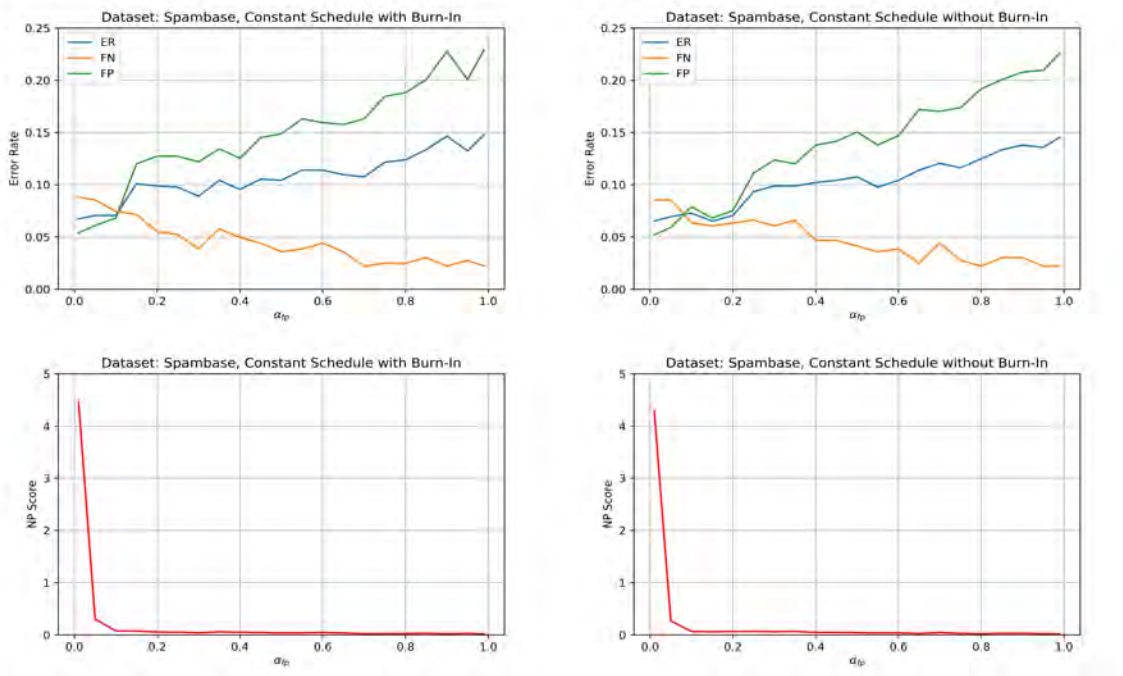


Figure 17. Spambase dataset results using a constant schedule for η_t, with and without burn-in periods

For the Spambase dataset, all decrement schedules are able to achieve FPR that are below the desired upper bound given by α_{fp} thresholds of 0.1 or more. The linear decrement schedule without burn-in is able to achieve the best FNR, while maintaining the lowest FPR as we relax the desired upper bound fiven by α_{fp}. We also note that the constant schedule, particularly without the burn-in period, show higher FNRs compared to the other schedules.

Figures 18 to 20 show the results for the Pima dataset.

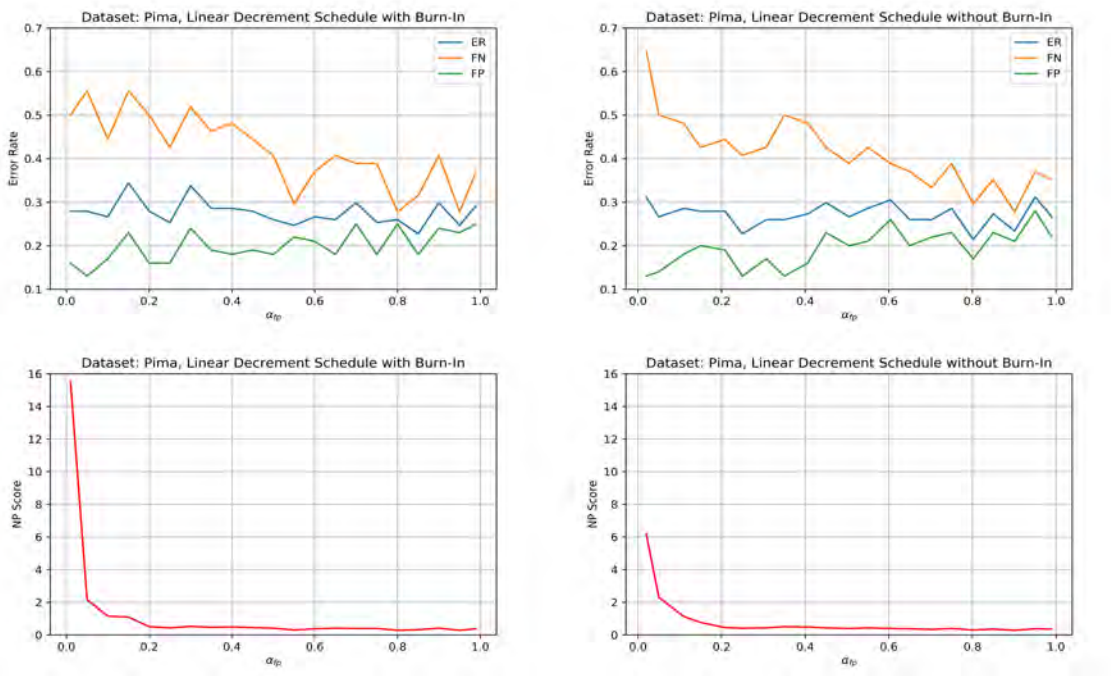


Figure 18. Pima dataset results using a linear schedule for decreasing η_t, with and without burn-in periods

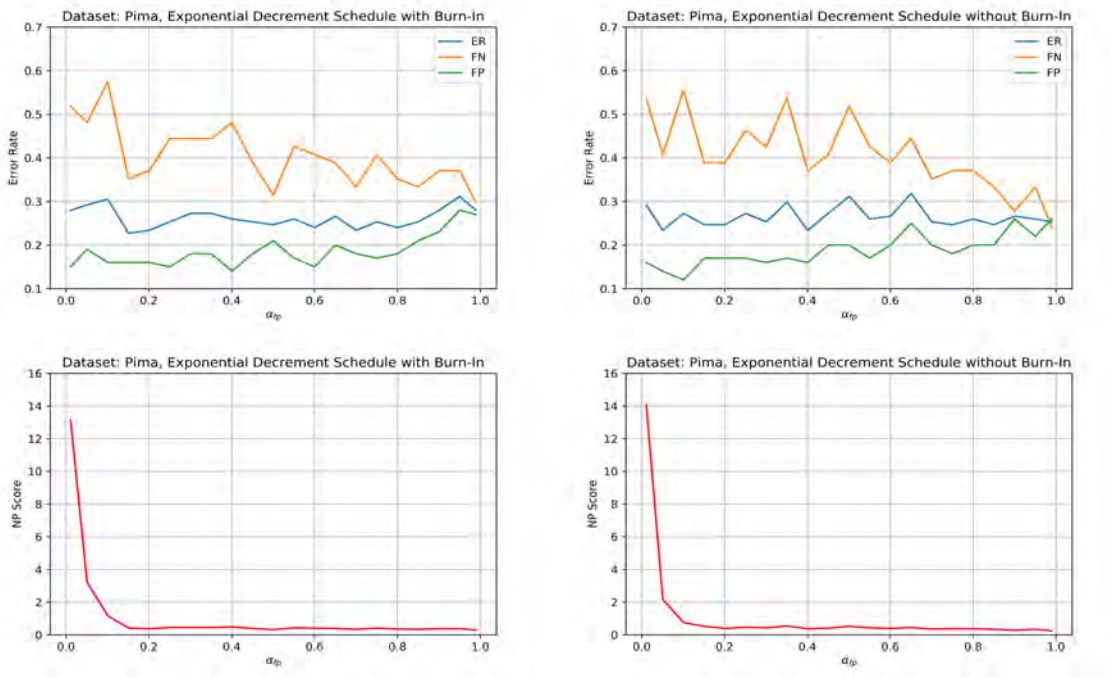


Figure 19. Pima dataset results using an exponential schedule for decreasing η_t, with and without burn-in periods

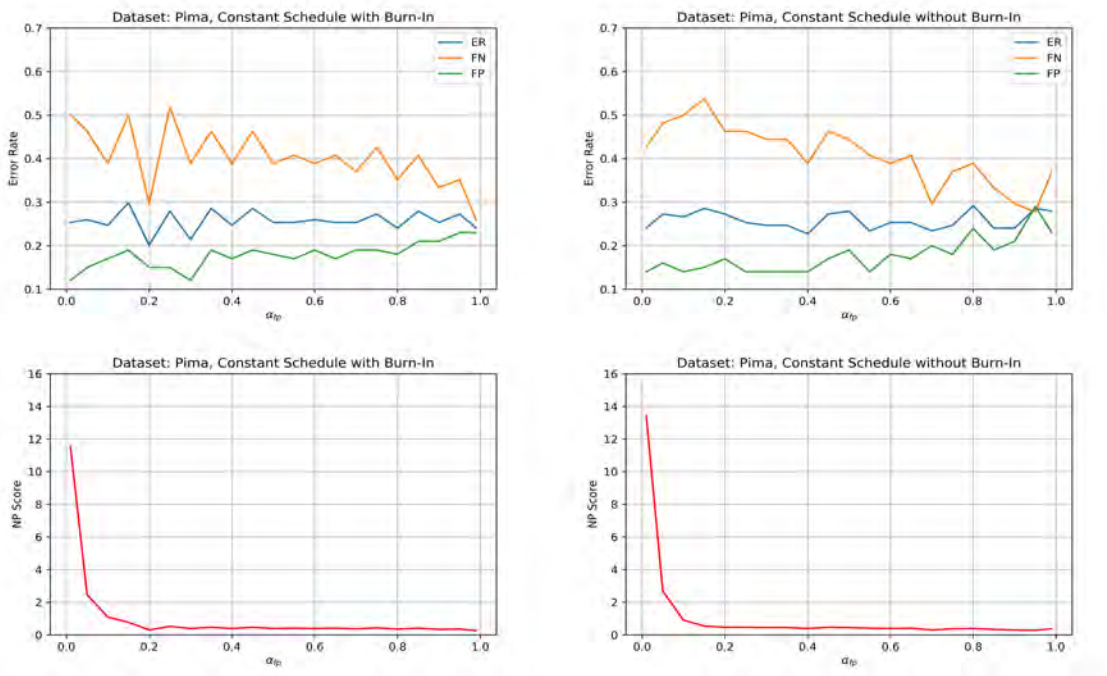


Figure 20. Pima dataset results using a constant schedule for η_t, with and without burn-in periods

The constant schedule with burn-in period appears to work better on the Pima dataset, producing FNRs that are lower compared to the other schedules. The linear decrement schedule (with and without burn-in periods) do not seem to be as effective as the other schedules, as can be seen by the higher FNR across most upper bound threshold of α_{fp}.

Figures 21 to 23 show the results for the Page Blocks dataset.

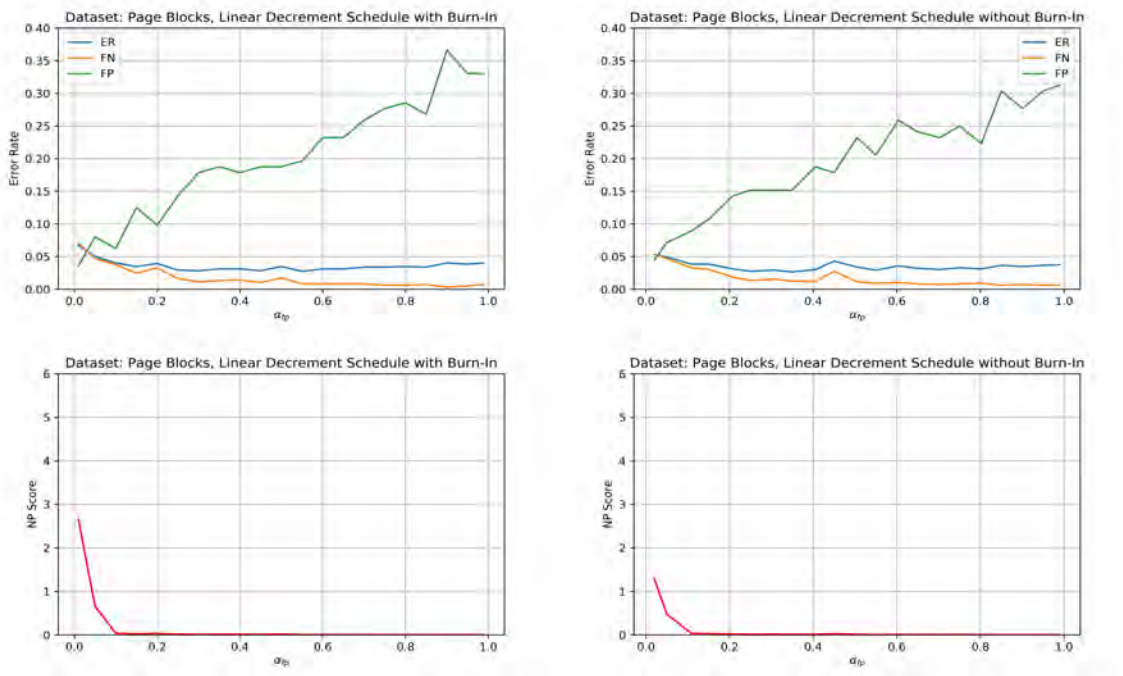


Figure 21. Page Blocks dataset results using a linear schedule for decreasing η_t, with and without burn-in periods

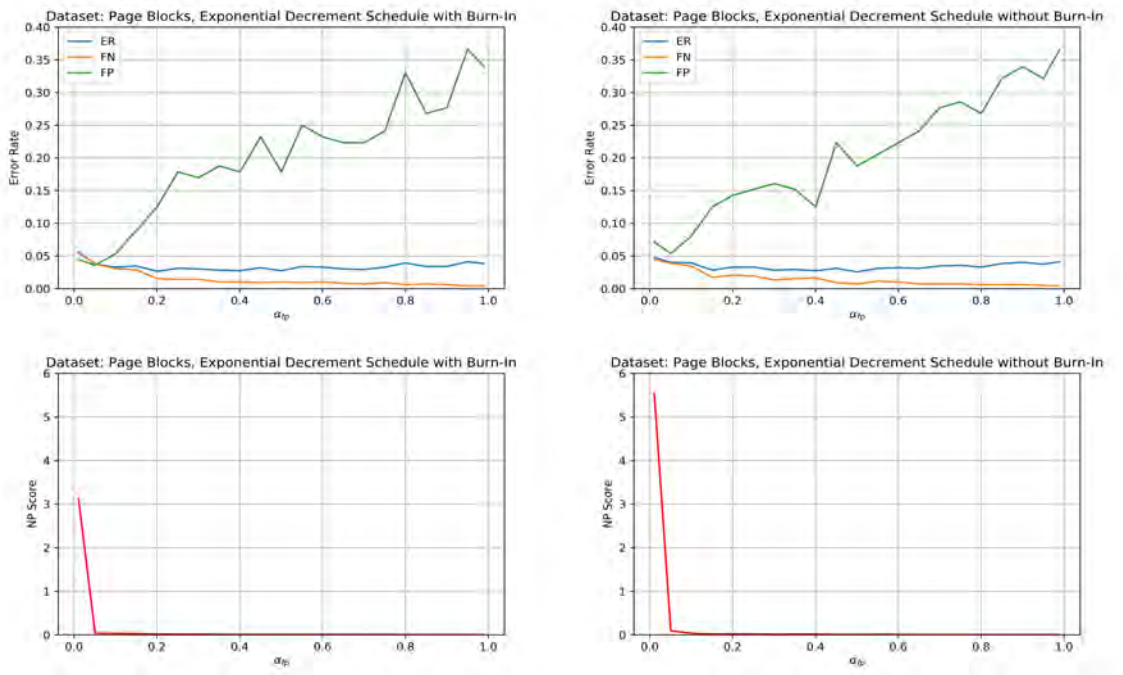


Figure 22. Page Blocks dataset results using an exponential schedule for decreasing η_t, with and without burn-in periods

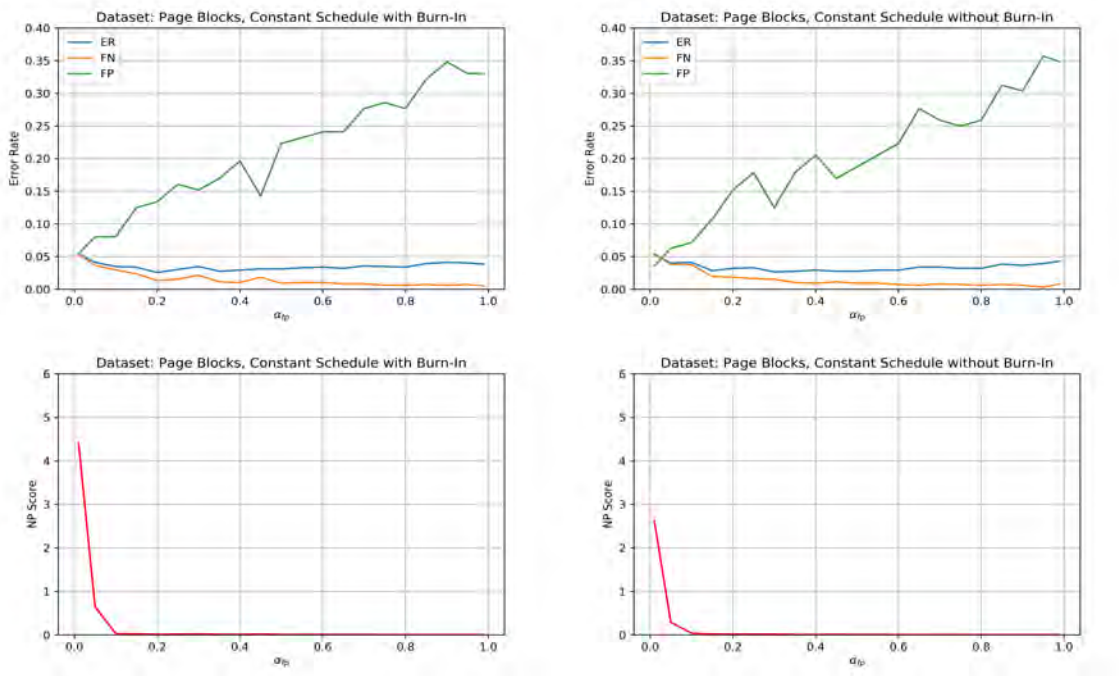


Figure 23. Page Blocks dataset results using a constant schedule for η_t, with and without burn-in periods

There appears to be negligible difference in performance statistics when using different decrement schedules for the Page Blocks dataset. All the decrement schedules were able to satisfy the α_{fp} thresholds of 0.1 or more, but we do not see much improvement in the FNR as we relax the constraint on the FPR.

Figures 24 to 26 show the results for the German Credit dataset.

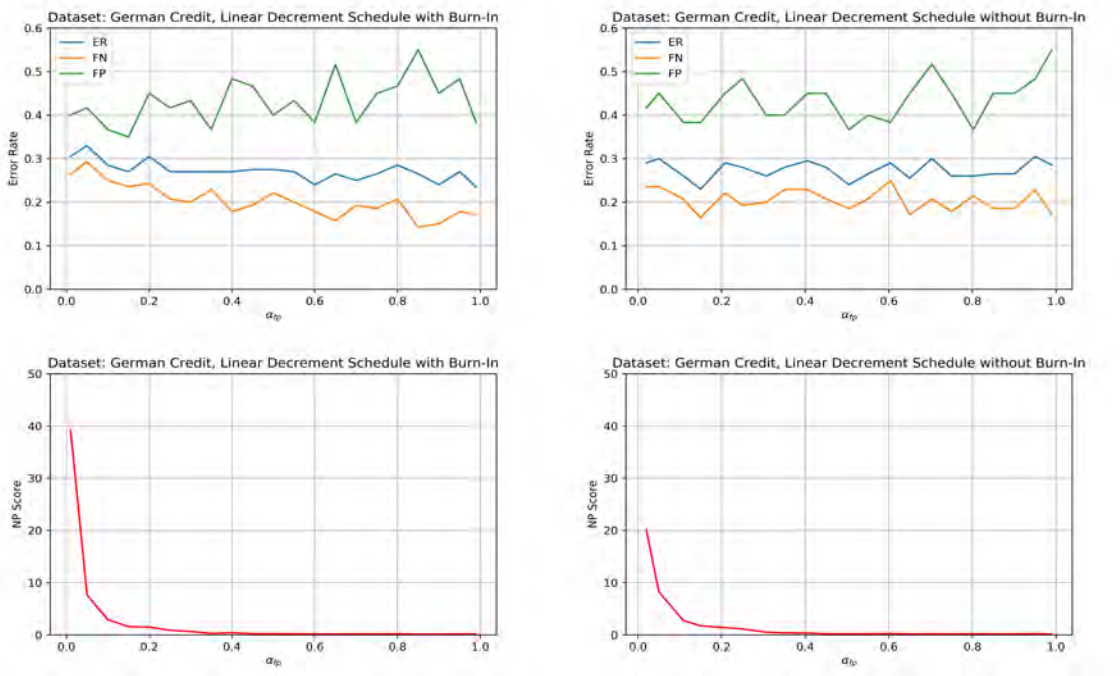


Figure 24. German Credit dataset results using a linear schedule for decreasing η_t, with and without burn-in periods

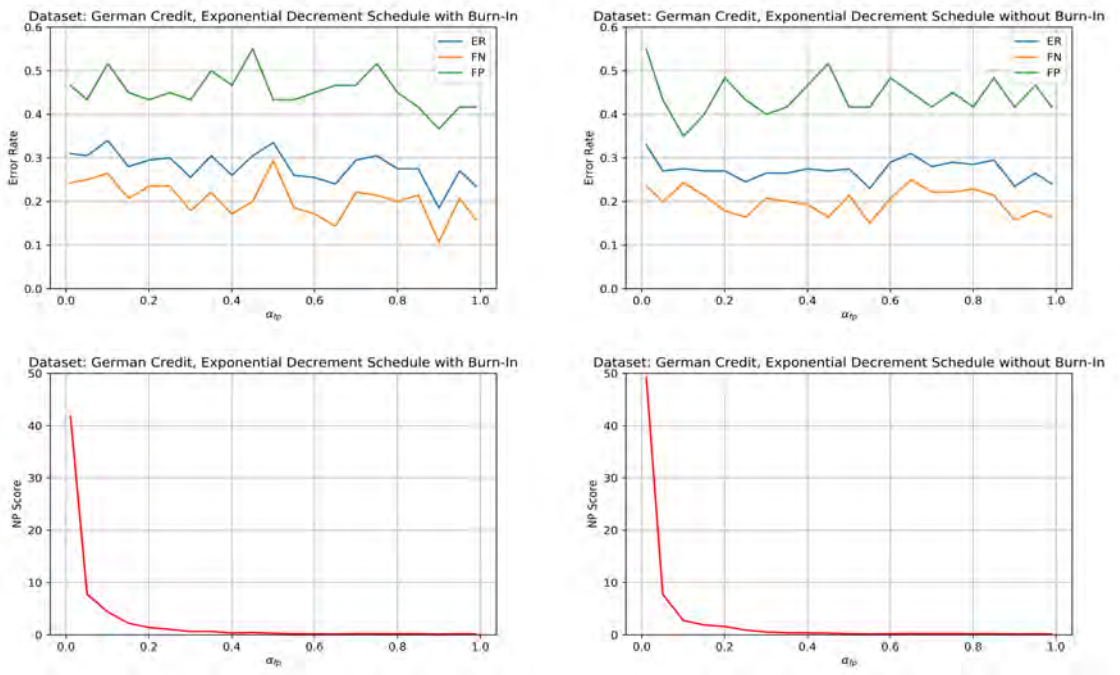


Figure 25. German Credit dataset results using an exponential schedule for decreasing η_t, with and without burn-in periods

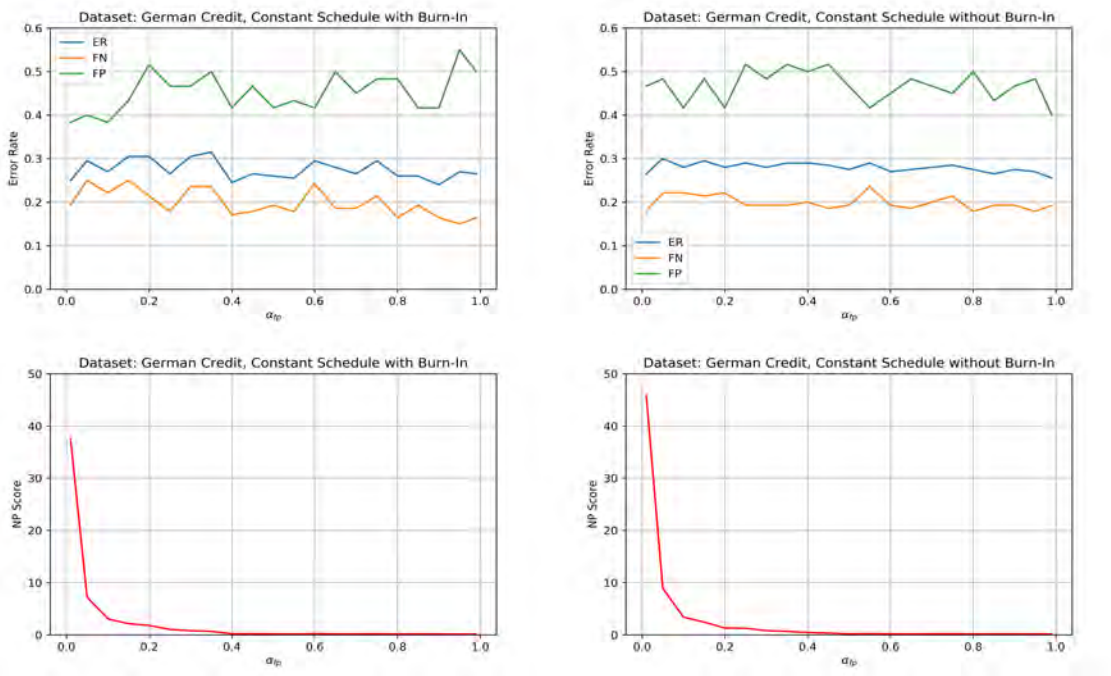


Figure 26. German Credit dataset results using a constant schedule for η_t, with and without burn-in periods

As discussed in previous sections, the algorithm does not seem to perform well on the German Credit dataset and decrement schedule seem to have little effect on improving the results. All the decrement schedules are not able to achieve FPRs that are below the desired upper bound given by α_{fp} for α_{fp} thresholds of 0.4 and lower. We conclude that the German Credit dataset is not well suited for a single layer NN classifier and that future work could include exploring the use of other classifier families for this dataset.

Overall, the algorithm performs best with a schedule that has linear decrement of η_t without a burn-in period. Having an exponential decrement schedule does not appear to produce better results than the linear decrement schedule as the algorithm is forced to work on improving the constraint function with a faster decrease in η_t at the beginning of the experiment. The constant schedule tends to overfit the constraint by forcing the algorithm to work predominantly on improving the constraint function, causing not much improvement for FNR as we relax the α_{fp} thresholds. Schedules without the burn-in period tend to have a higher FPR as the alrogithm is probably overfitting the objective function during the burn-in period, therefore we see higher FPRs.

4.4 Other Design Choices

In the previous section we utilized algorithm settings that were found via trial and error. We comment specifically on some of these choices and the effects witnessed when they were changed. This includes our choices of batch size equal to 1 for calculating the gradients and the use of the same momentum and learning rate for both constraint and objective function gradient steps.

Training Batch Sizes

Training batch sizes of 1 appear to have the best performance for all datasets. Larger batch sizes give more accurate estimation of the true gradient and do not seem to add sufficient "noise" to the algorithm. Appendix A.1 shows the plots of error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using training batch sizes of 1, 10 and 100.

Constraint Function Learning Rate

The learning rate for Procedure A and B of C-SGD do not need to be equal. However, we found that constraint function learning rates that are equal with the objective function learning rate achieve the best results for most of the datasets. Therefore, we use a constraint function learning rate of 0.01 for the analyses. Appendix A.2 shows the plots of error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using objective function learning rate of 0.01, and varying constraint function learning rates of 0.005, 0.0075, 0.01, 0.0125 and 0.015.

Constraint Function Momentum

The magnitude of the momentum parameter for Procedure A and Procedure B also do not need to be equal. However, we find there is negligible improvement in using constraint function momentums that are smaller than the objective function momentum. Therefore, we use a constraint and objective function momentum of 0.5 for the rest of the analyses. Appendix A.3 shows the plots of error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using objective function momentum of 0.01, and varying constraint momentums of 0.1, 0.2, 0.3, 0.4 and 0.5.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions and Future Work

In this thesis, we have tackled a highly challenging variant of binary classification called NP classification. The NP-classification setting, while challenging, is an important paradigm for application in which the FNs and FPs have dramatically different costs and it is difficult or unethical to assign actual costs to these types of errors. Overall, we saw that this important framework poses multiple challenges. Firstly, we must formulate a tractable optimization problem to work with since the original problem, (2.1), involves the 0-1 loss function which is intractable when optimization enters the picture. We utilized a hinge-loss approximation of the 0-1 loss, which is justified by recent work in (Norton and Uryasev 2017) which shows that an optimal scaling of this loss is a probabilistic upper bound on the FNR and FPR. Secondly, we must select a classifier family and we selected NNs. This selection is motivated by the recent success NNs have had in pattern recognition and their flexibility for processing different types of data, such as images or text. If we can successfully propose a method to train NNs for this setting, it provides hope that other more complex NN architectures can be trained in a similar manner in the NP-classification setting. Thirdly, we find that our formulation with hinge-loss and NN classifiers presents a challenging nonconvex, high dimensional, constrained optimization problem. Thus, we seek an efficient first-order optimization routine, similar to SGD, that works in the constrained optimization setting. We propose the use of C-SGD, a new method for solving constrained optimization problems with stochastic gradients.

We find that the C-SGD algorithm is complex, but often can yield good solutions when hyperparameters are chosen carefully. Overall, we are able to show that NN classifiers can be trained via C-SGD in the NP-classification setting. Specifically, using our tractable hinge-loss formulation and C-SGD, we are able to find NN classifiers with FPR below our desired threshold while simultaneously minimizing the FNR.

The C-SGD method, as already mentioned, is complex. Thus, we explore some of its critical components. We first point out that, while similar to SGD, comes with the price of having to check if the constraint is satisfied at the start of every step in the training loop. Checking the constraint incurs additional time in the algorithm compared to regular SGD

and the amount of additional time depends on the number examples used to estimate the true value of the constraint function. We find that larger sample sizes, and increasingly accurate estimates, comes with diminishing returns. Using constraint batch sizes of approximately 10 to 100 gives the best performance statistics versus the computational cost. Additionally, we see that the rate at which feasibility is strictly enforced via the schedule of η_t is important for performance. We find that a balanced enforcement policy, where η_t gradually decreases from 1 to α_{fp} is the best choice, at least for our experiments and datasets.

5.1 Future Work

Overall, we find that C-SGD provides promising results for training NNs in a constrained setting. Future work would seek to first utilize more complex NN architectures within the same framework, performing NP-classification and using C-SGD as the optimization method. For example, while we utilized a simple single layer fully-connected NN, it would be informative to utilize a convolutional architecture on an image classification dataset. Additionally, the role of hyperparameters would likely change as the network became deeper, with millions more parameters.

Another area for future improvement is by taking a closer look into the optimization algorithm. Specifically, there are many different heuristic strategies for altering SGD that have been shown to be effective when NN classifiers are being trained. These include adaptive step size choices and by taking the average value of your parameters over the final steps of your gradient descent algorithm. Overall, this work is one of the first in the literature to explore the training of NNs in a constrained setting with a first-order constrained optimization algorithm. Thus, this work serves as a proof-of-concept that it can be effective and that further investigation is warranted in more challenging NN settings.

APPENDIX: Preliminary Design Choices

A.1 Training Batch Size

We test the algorithm using different training batch sizes to determine if this plays an important factor in the analysis.

Figures A.13 to A.3 show the error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using training batch sizes of 1, 10 and 100.

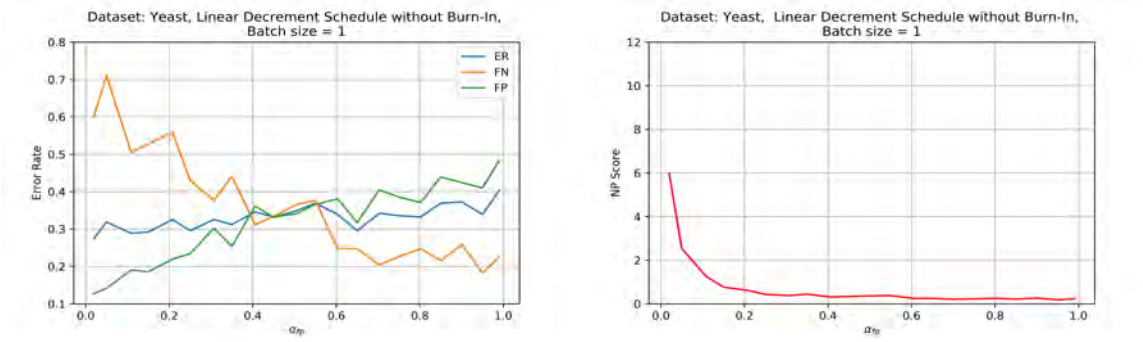


Figure A.1. Yeast dataset results for batch size = 1

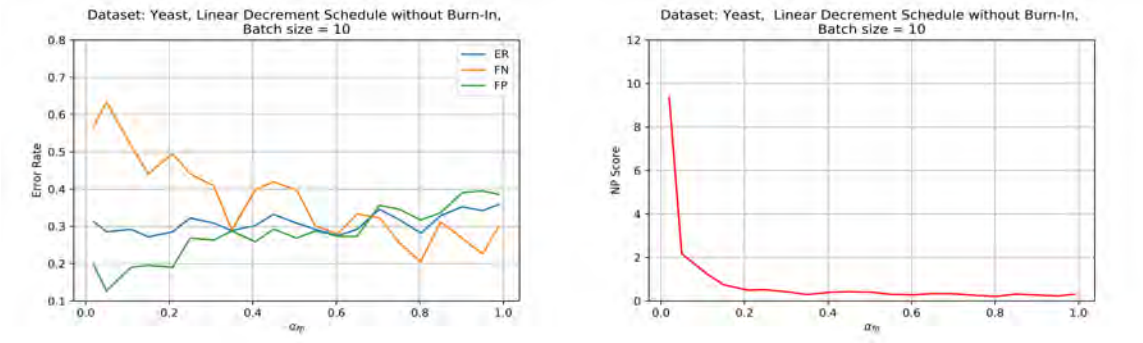


Figure A.2. Yeast dataset results for batch size = 10

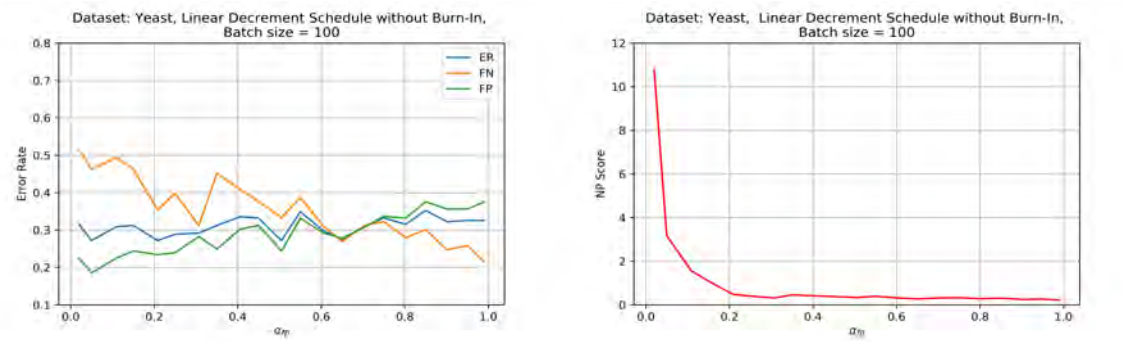


Figure A.3. Yeast dataset results for batch size = 100

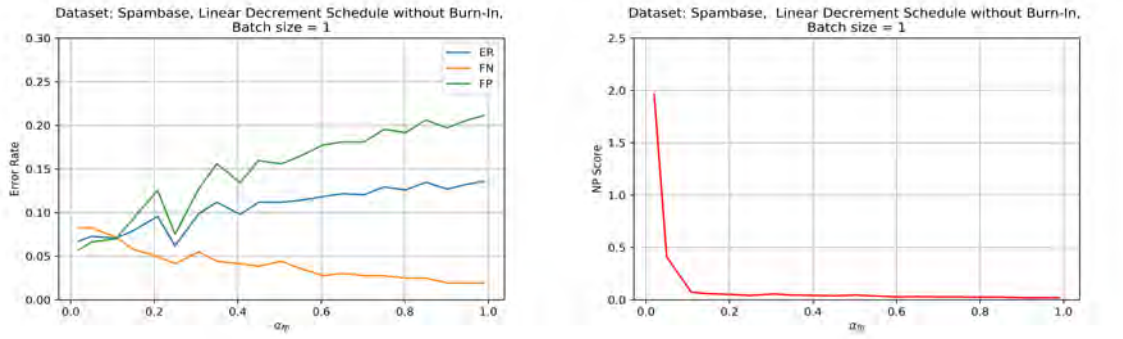


Figure A.4. Spambase dataset results for batch size = 1

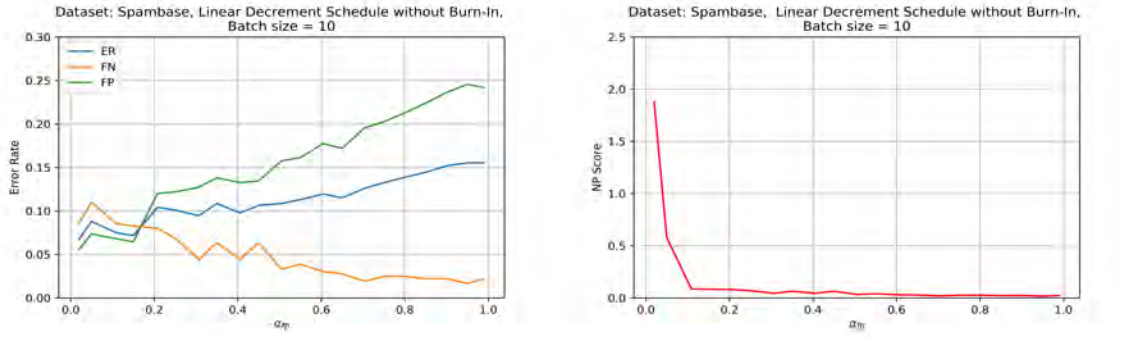
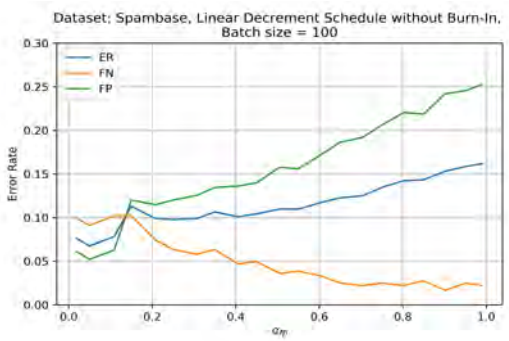


Figure A.5. Spambase dataset results for batch size = 10



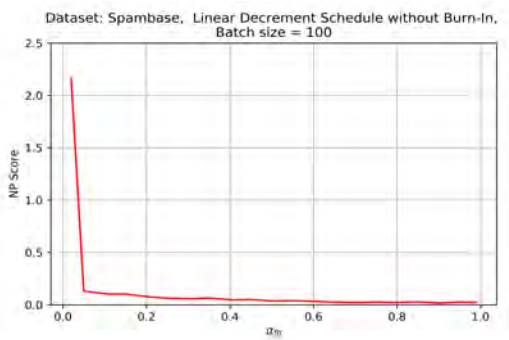
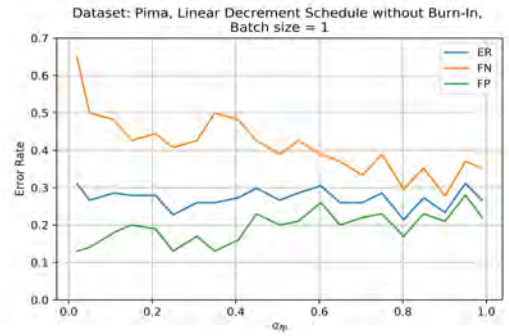


Figure A.6. Spambase dataset results for batch size = 100



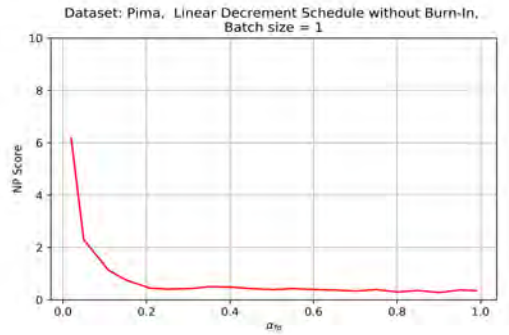
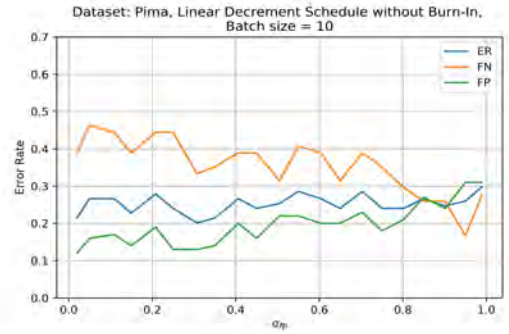


Figure A.7. Pima dataset results for batch size = 1



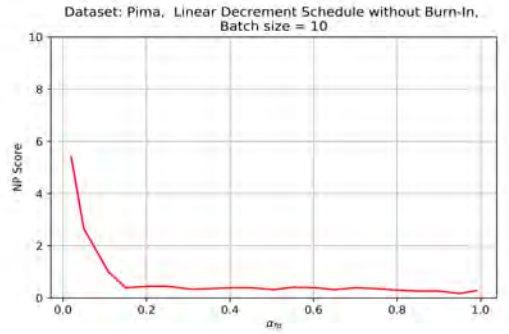


Figure A.8. Pima dataset results for batch size = 10

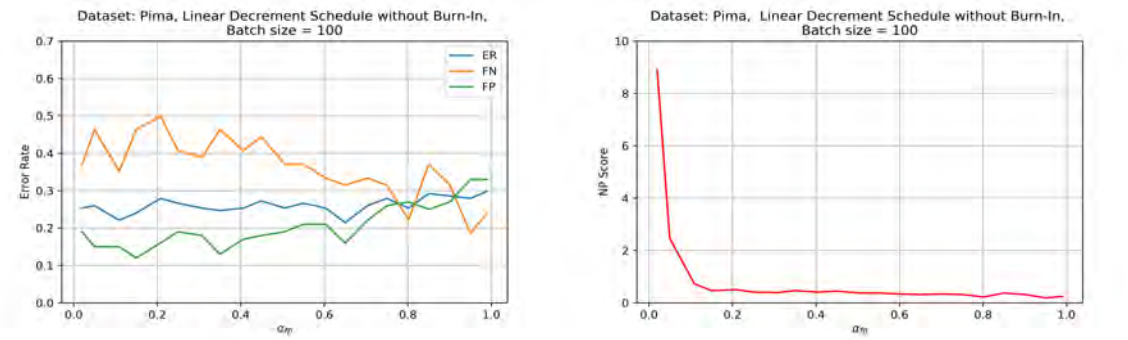


Figure A.9. Pima dataset results for batch size = 100

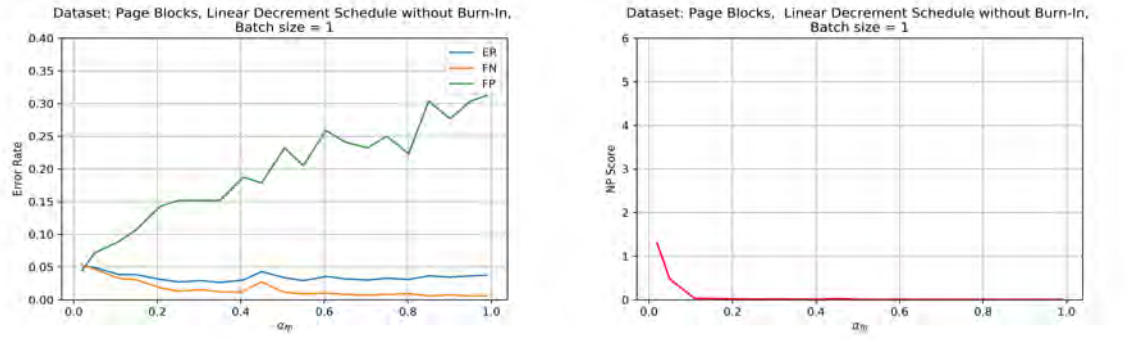


Figure A.10. Page Blocks dataset results for batch size = 1

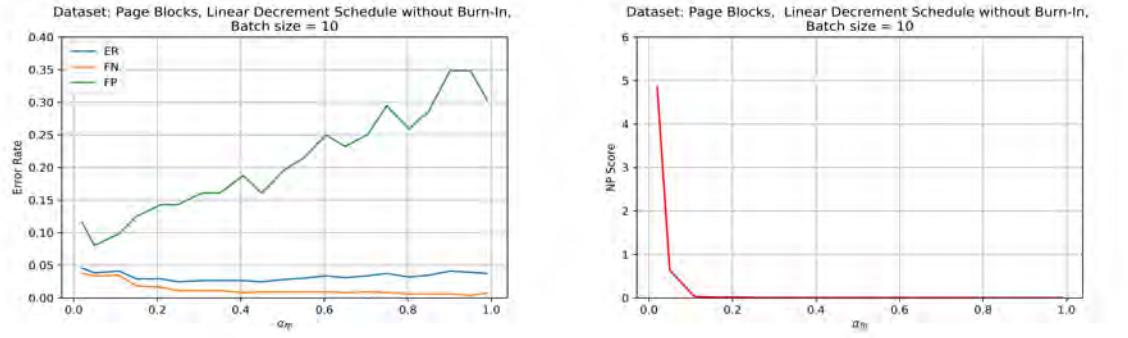


Figure A.11. Page Blocks dataset results for batch size = 10

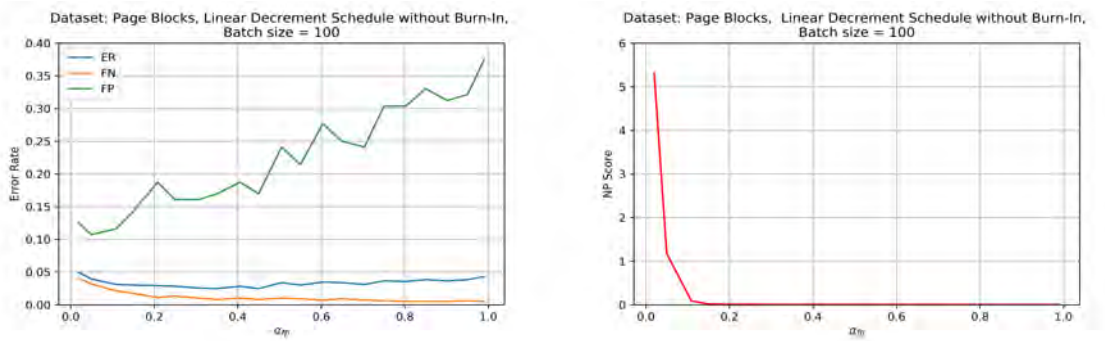


Figure A.12. Page Blocks dataset results for batch size = 100

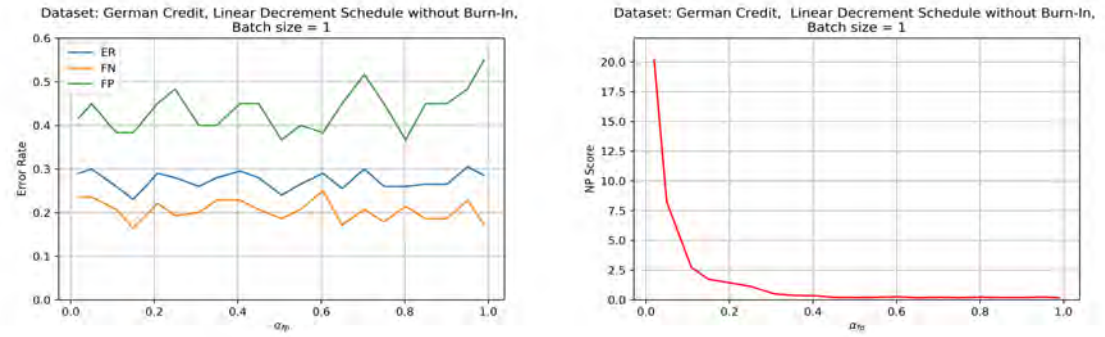


Figure A.13. German Credit dataset results for batch size = 1

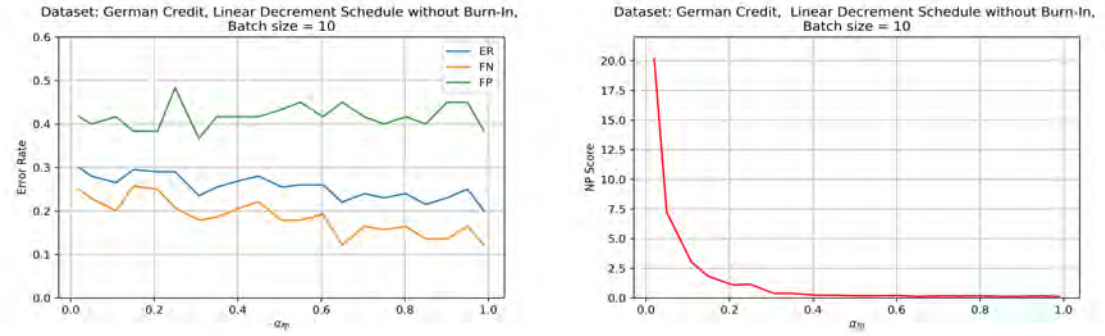


Figure A.14. German Credit dataset results for batch size = 10

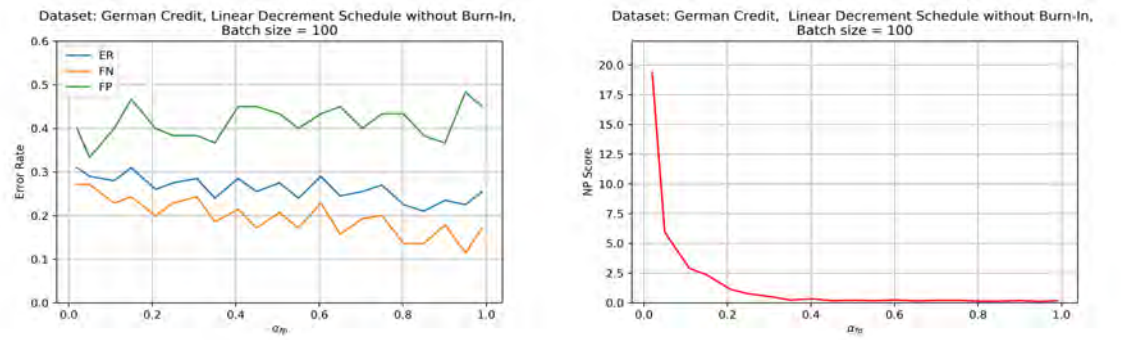


Figure A.15. German Credit dataset results for batch size = 100

As can be seen from Figures A.13 to A.3, training batch sizes of 1 appear to have the best performance for all datasets.

A.2 Constraint Function Learning Rate

We test the algorithm using different constraint function learning rates to determine if this plays an important factor in the analysis.

Figures A.36 to A.20 show the error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using objective function learning rate of 0.01, linear decrement schedule down to the α_{fp} threshold of 0.01, and varying constraint function learning rates of 0.005, 0.0075, 0.01, 0.0125 and 0.015. Note that these plots are tracking performance on the test set during training.

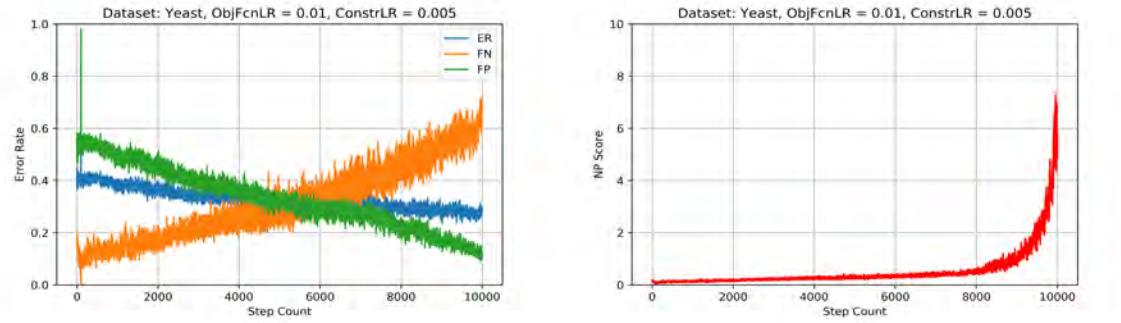


Figure A.16. Yeast dataset results for objective function learning rate = 0.01, constraint learning rate = 0.005

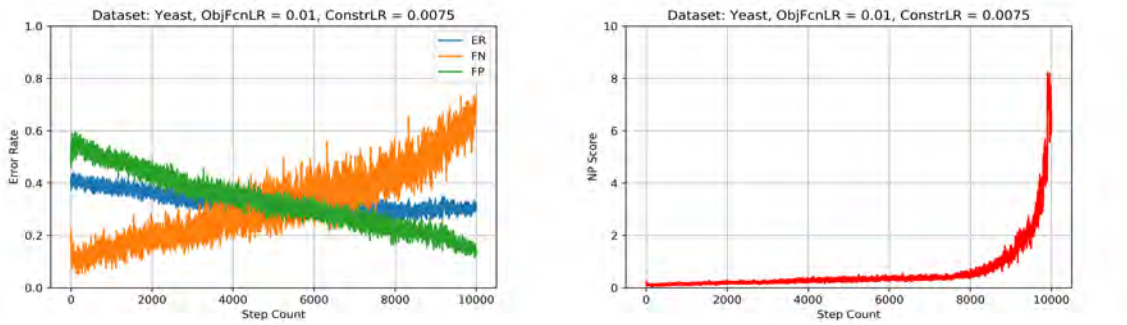


Figure A.17. Yeast dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0075

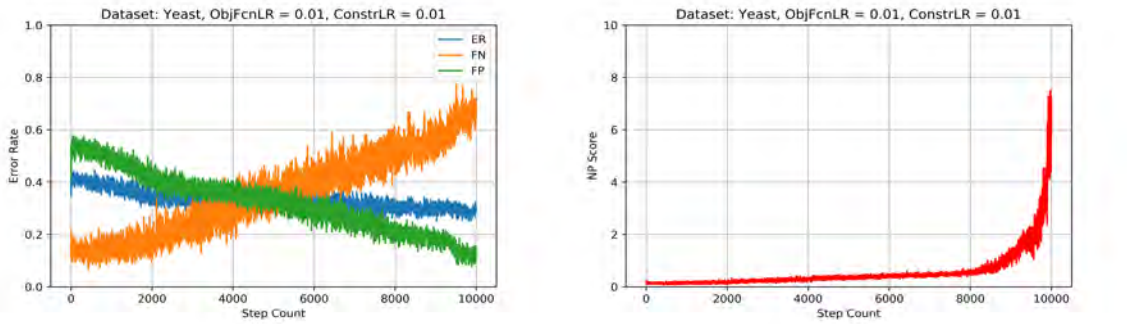


Figure A.18. Yeast dataset results for objective function learning rate = 0.01, constraint learning rate = 0.01

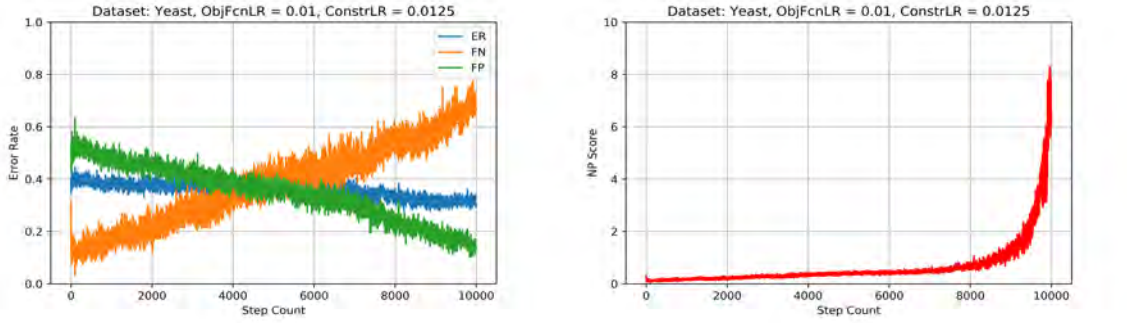


Figure A.19. Yeast dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0125

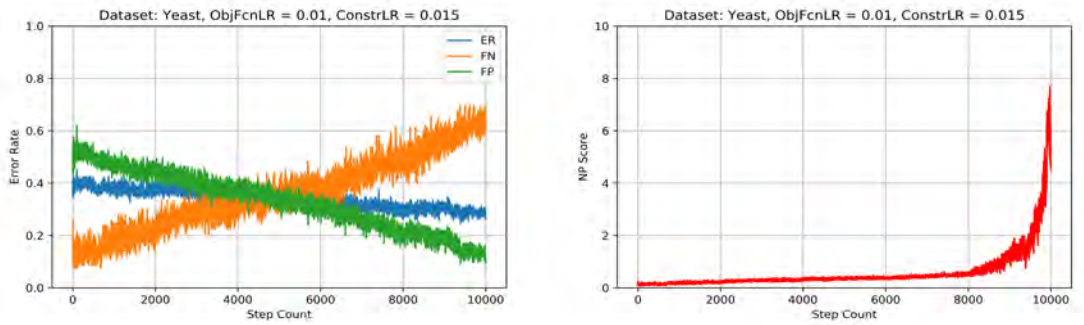


Figure A.20. Yeast dataset results for objective function learning rate = 0.01, constraint learning rate = 0.015

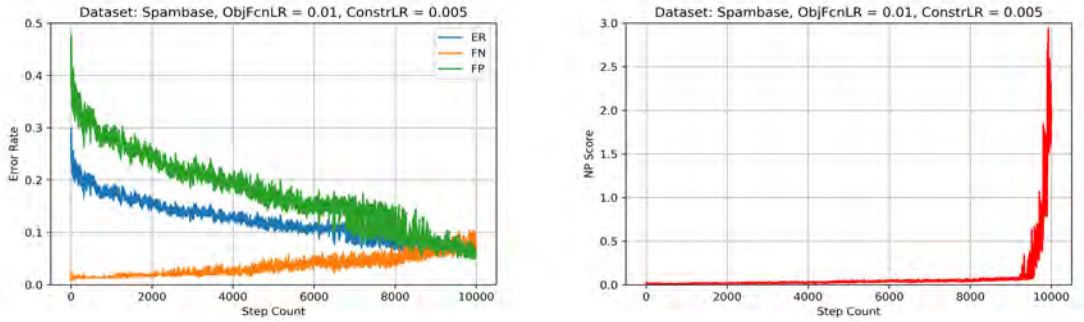


Figure A.21. Spambase dataset results for objective function learning rate = 0.01, constraint learning rate = 0.005

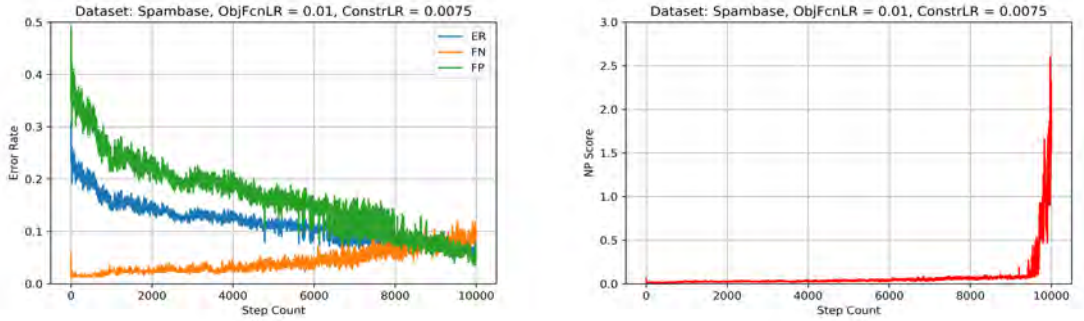


Figure A.22. Spambase dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0075

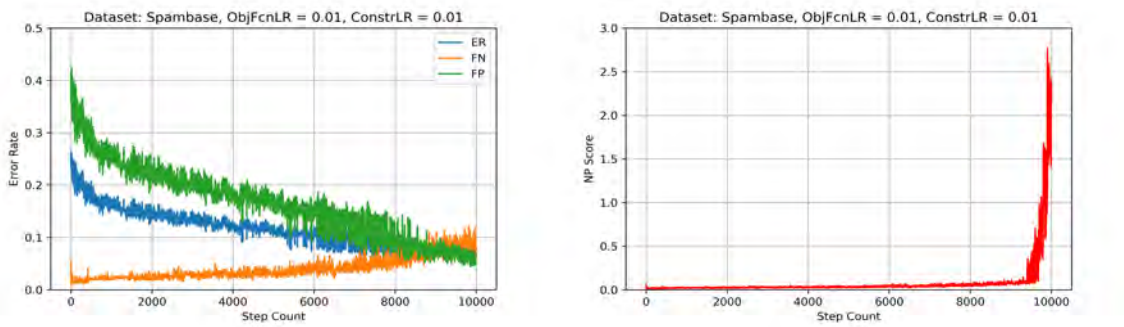


Figure A.23. Spambase dataset results for objective function learning rate = 0.01, constraint learning rate = 0.01

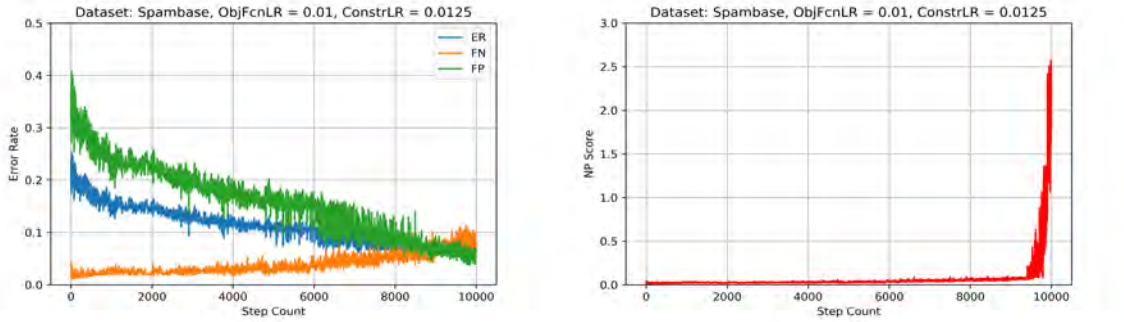


Figure A.24. Spambase dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0125

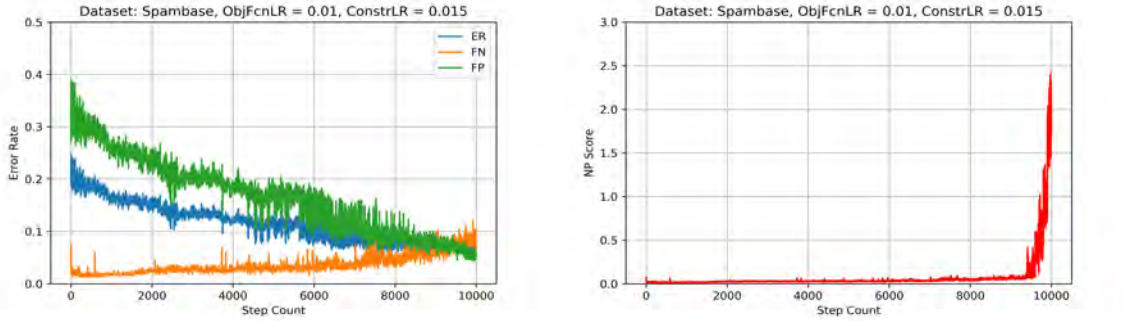


Figure A.25. Spambase dataset results for objective function learning rate = 0.01, constraint learning rate = 0.015

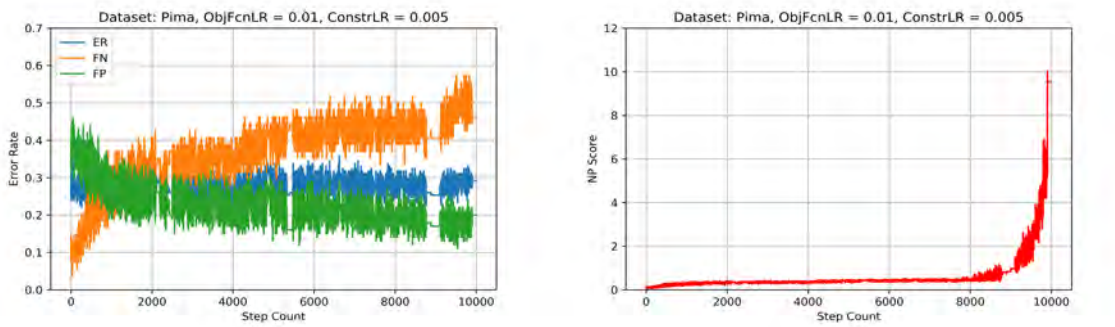


Figure A.26. Pima dataset results for objective function learning rate = 0.01, constraint learning rate = 0.005

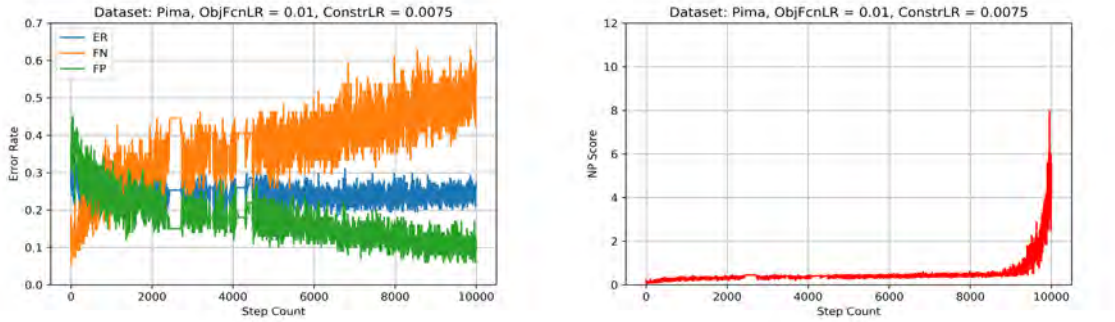


Figure A.27. Pima dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0075

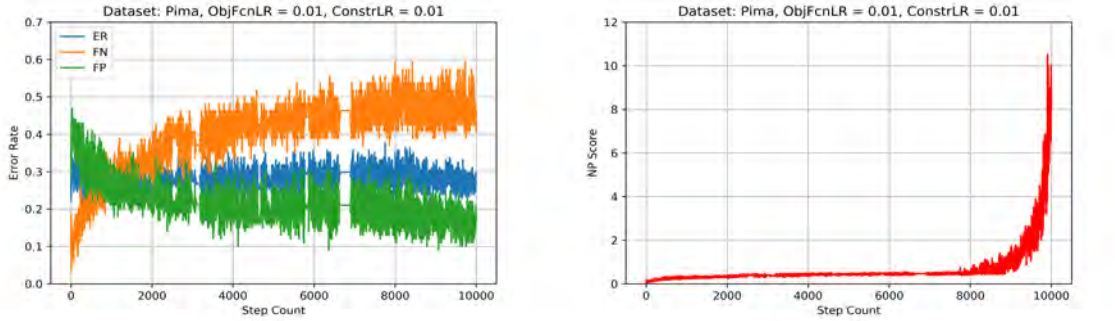


Figure A.28. Pima dataset results for objective function learning rate = 0.01, constraint learning rate = 0.01

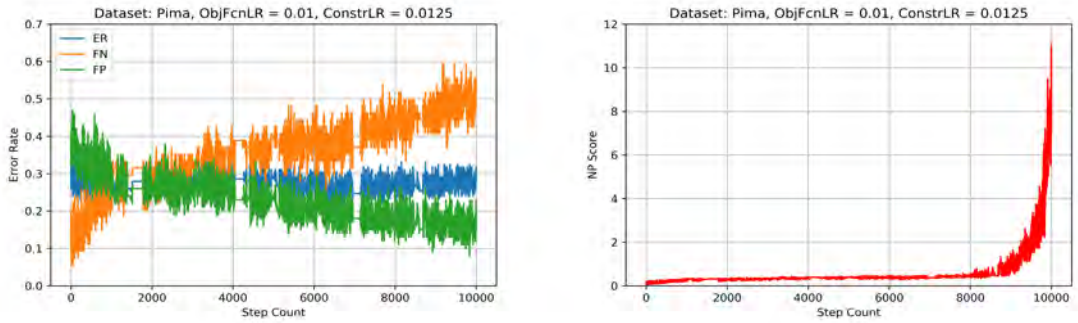


Figure A.29. Pima dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0125

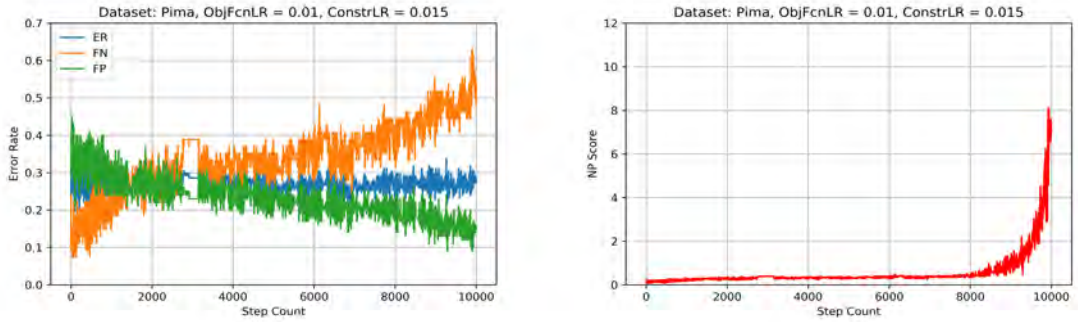


Figure A.30. Pima dataset results for objective function learning rate = 0.01, constraint learning rate = 0.015

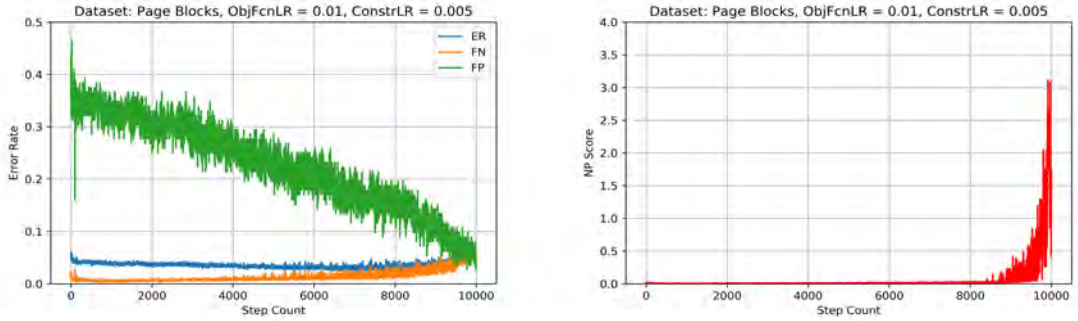


Figure A.31. Page Blocks dataset results for objective function learning rate = 0.01, constraint learning rate = 0.005

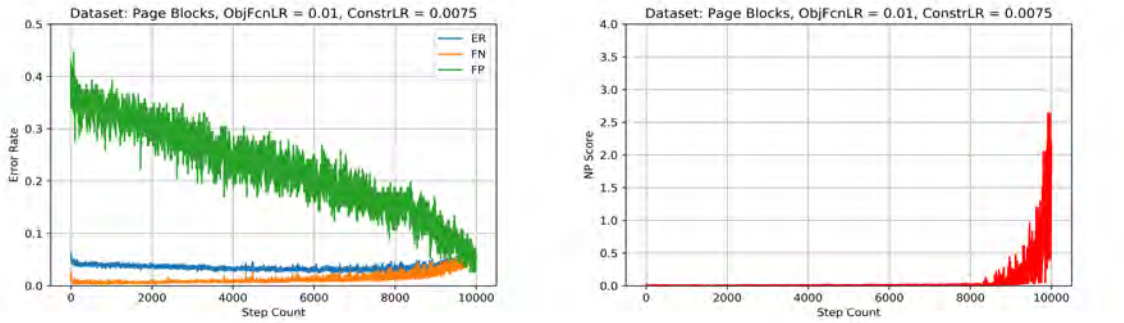


Figure A.32. Page Blocks dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0075

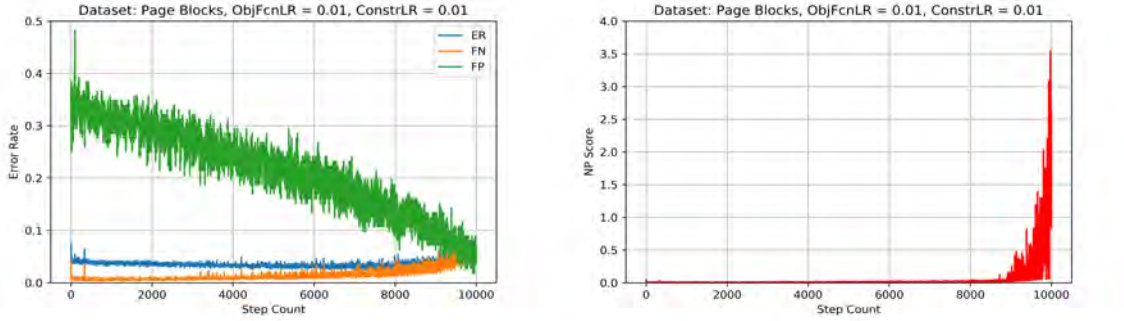


Figure A.33. Page Blocks dataset results for objective function learning rate = 0.01, constraint learning rate = 0.01

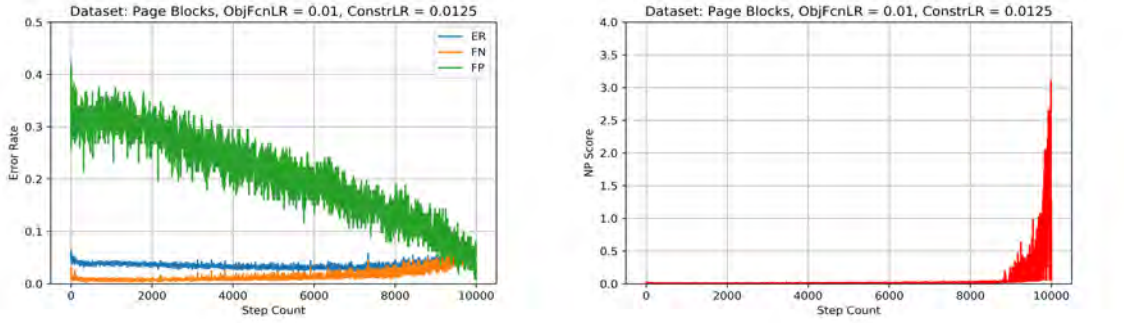


Figure A.34. Page Blocks dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0125

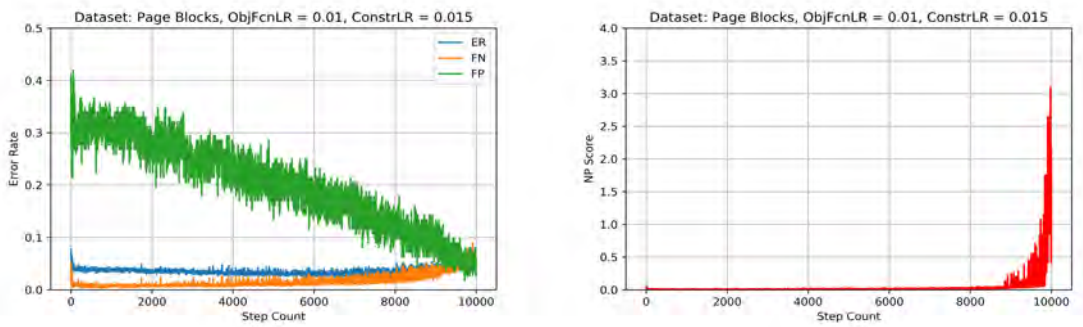


Figure A.35. Page Blocks dataset results for objective function learning rate = 0.01, constraint learning rate = 0.015

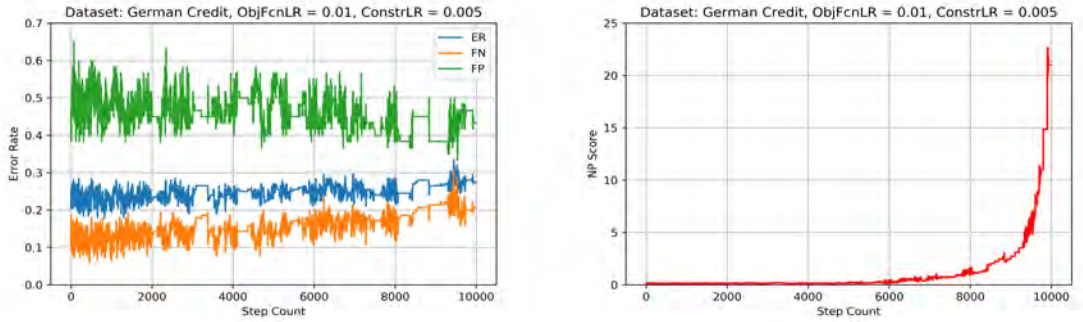


Figure A.36. German Credit dataset results for objective function learning rate = 0.01, constraint learning rate = 0.005

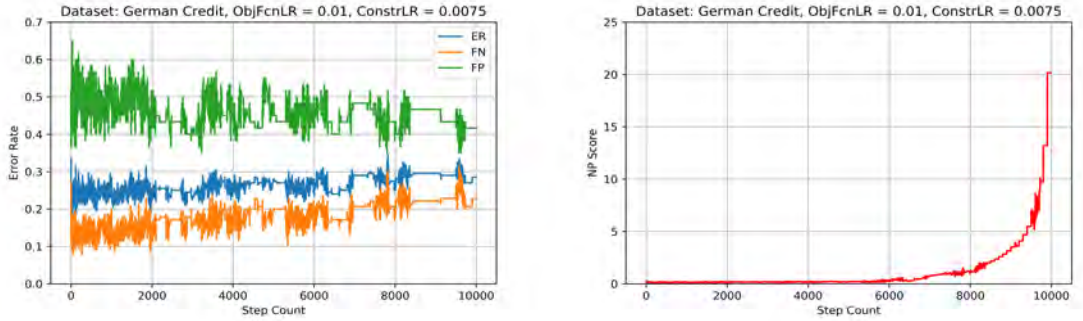


Figure A.37. German Credit dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0075

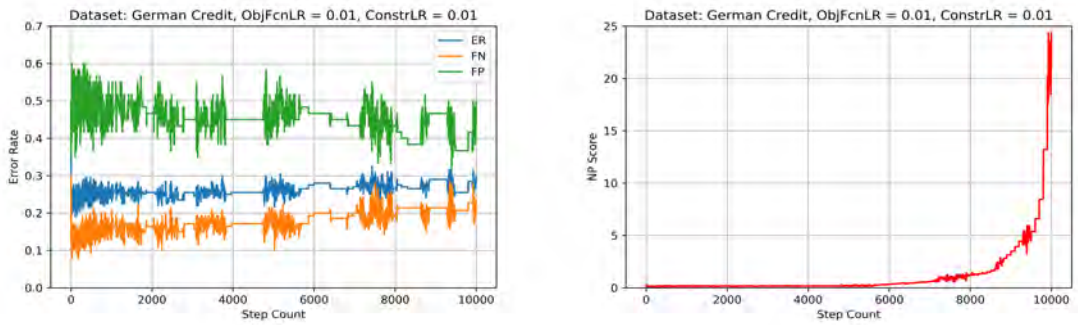


Figure A.38. German Credit dataset results for objective function learning rate = 0.01, constraint learning rate = 0.01

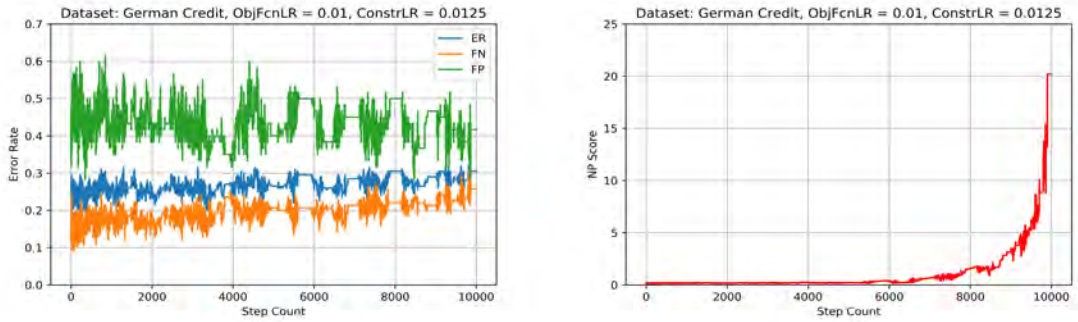


Figure A.39. German Credit dataset results for objective function learning rate = 0.01, constraint learning rate = 0.0125

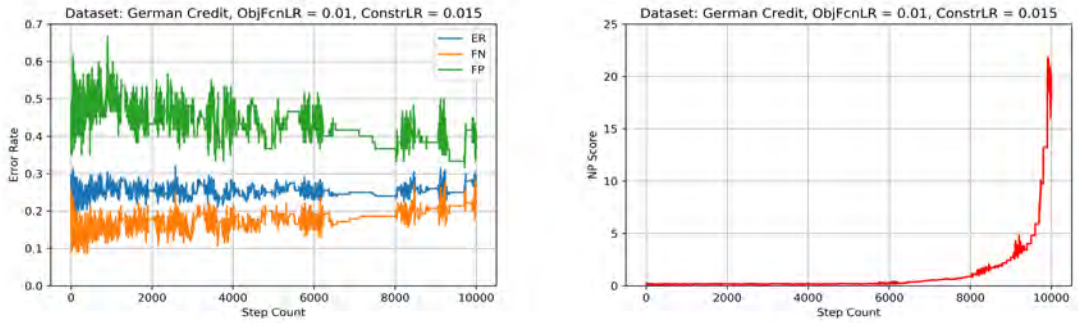


Figure A.40. German Credit dataset results for objective function learning rate = 0.01, constraint learning rate = 0.015

From Figures A.36 to A.20, constraint function learning rates that are equal with the objective function learning rate achieve the best results for most of the datasets. Therefore, we use a constraint function learning rate of 0.01 for the rest of the analyses.

A.3 Constraint Momentum

We test the algorithm using different constraint function momentums to determine if this plays an important factor in the analysis.

Figures A.61 to A.45 show the error rates and NP scores for all five UCI machine learning benchmark data (German Credit, Page Blocks, Pima, Spambase and Yeast), using objective function momentum of 0.01, linear decrement schedule down to the α_{fp} threshold of 0.01, and varying constraint momentums of 0.1, 0.2, 0.3, 0.4 and 0.5.Note that these plots are tracking performance on the test set during training.

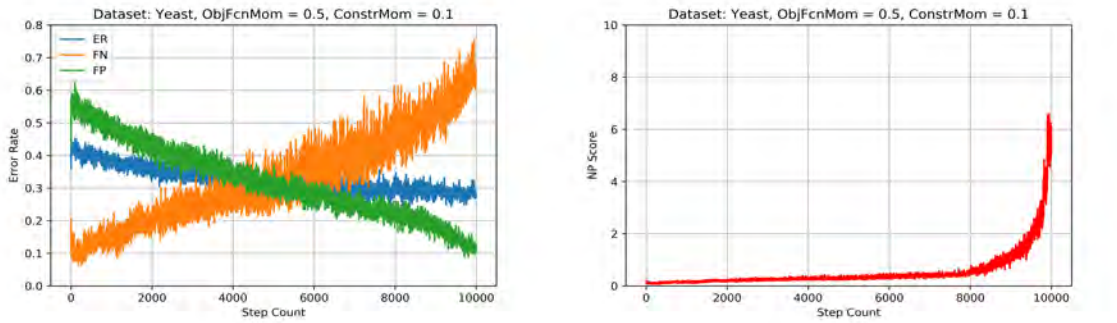


Figure A.41. Yeast dataset results for objective function momentum = 0.5, constraint momentum = 0.1

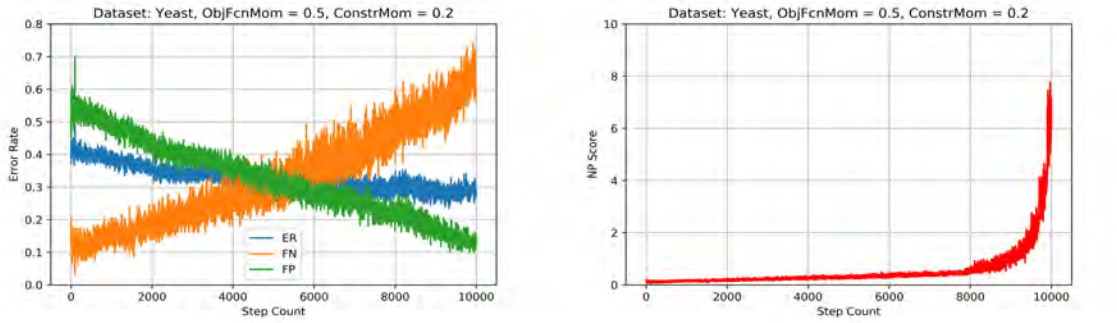


Figure A.42. Yeast dataset results for objective function momentum = 0.5, constraint momentum = 0.2

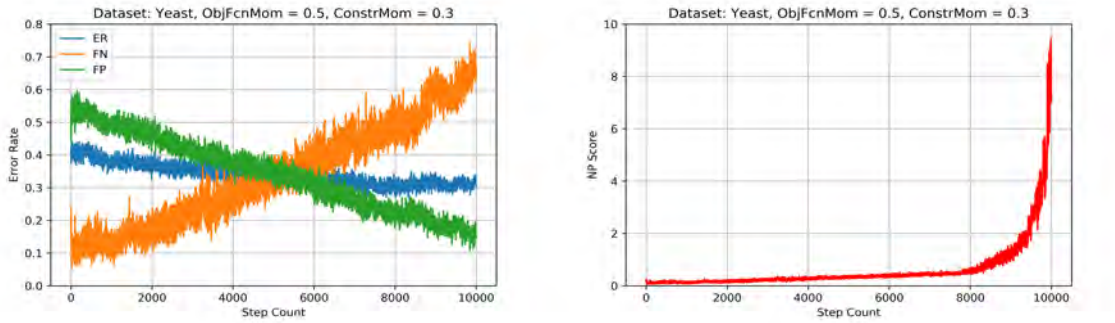


Figure A.43. Yeast dataset results for objective function momentum = 0.5, constraint momentum = 0.3

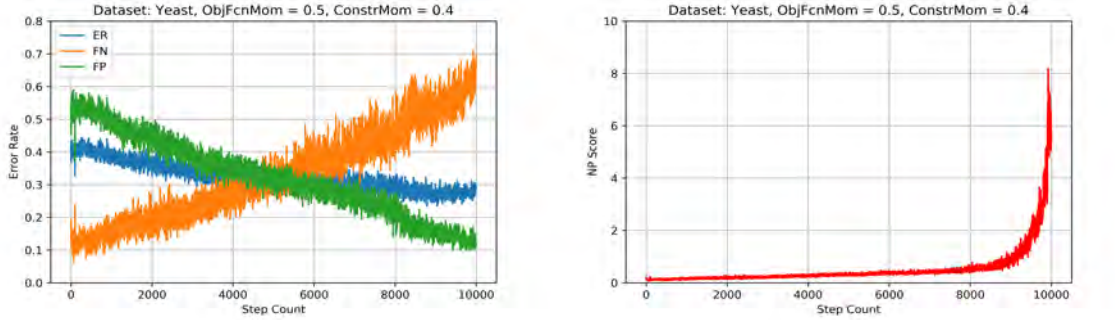


Figure A.44. Yeast dataset results for objective function momentum = 0.5, constraint momentum = 0.4

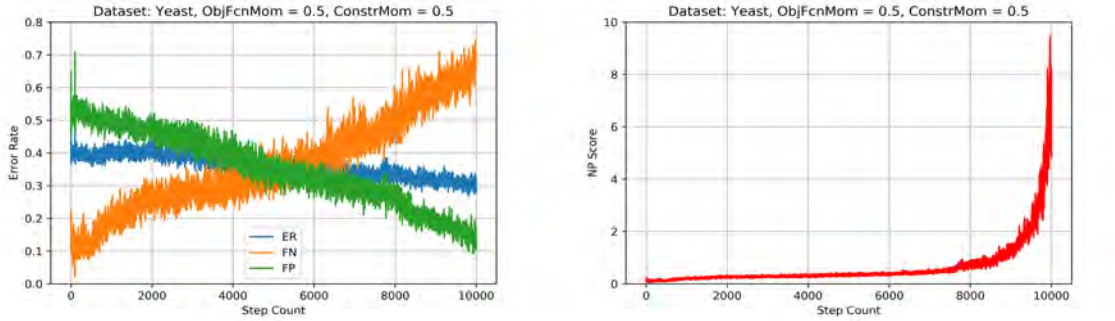


Figure A.45. Yeast dataset results for objective function momentum = 0.5, constraint momentum = 0.5

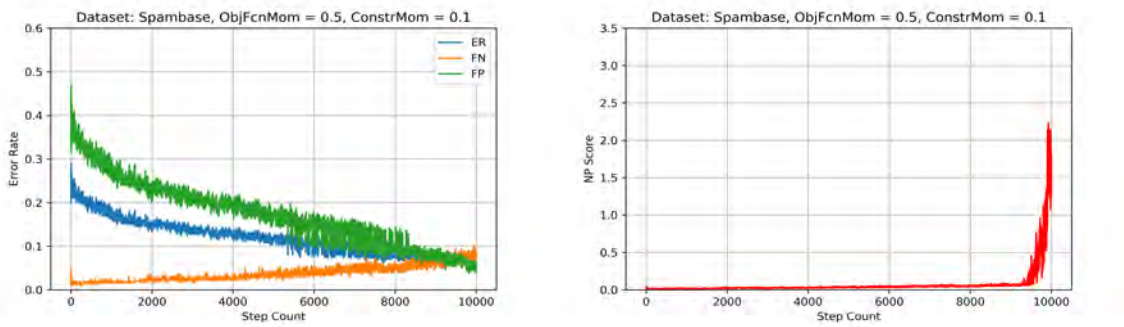


Figure A.46. Spambase dataset results for objective function momentum = 0.5, constraint momentum = 0.1

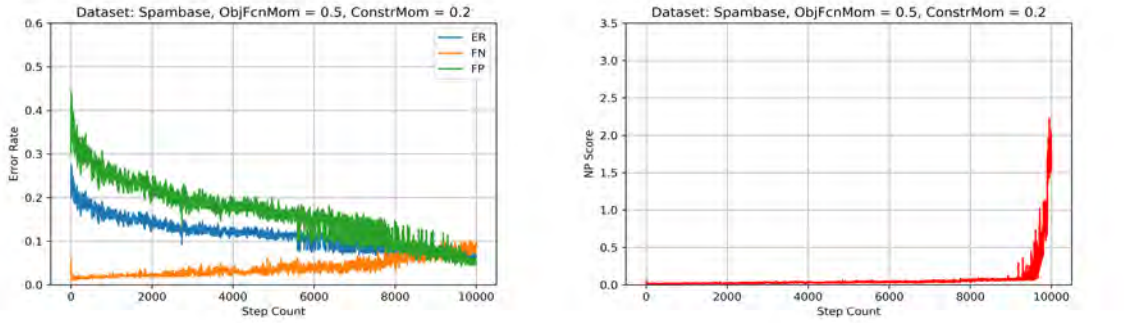


Figure A.47. Spambase dataset results for objective function momentum = 0.5, constraint momentum = 0.2

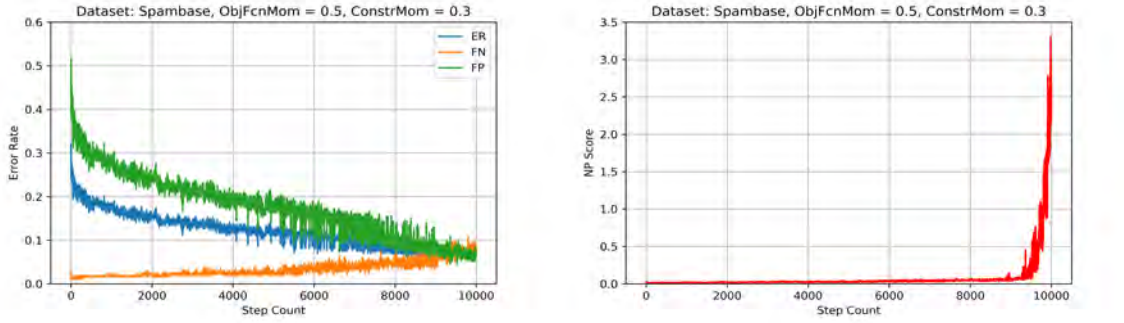


Figure A.48. Spambase dataset results for objective function momentum = 0.5, constraint momentum = 0.3

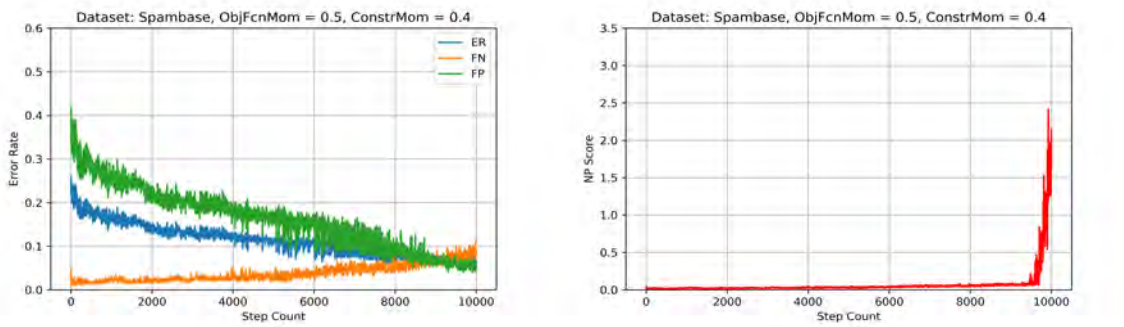


Figure A.49. Spambase dataset results for objective function momentum = 0.5, constraint momentum = 0.4

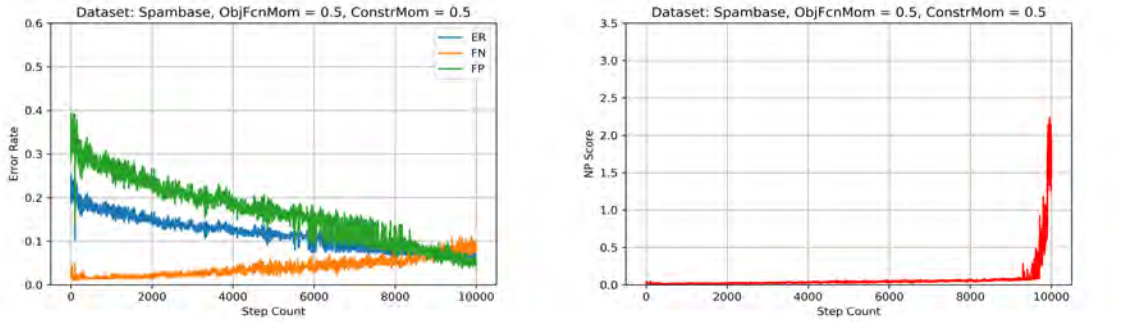


Figure A.50. Spambase dataset results for objective function momentum = 0.5, constraint momentum = 0.5

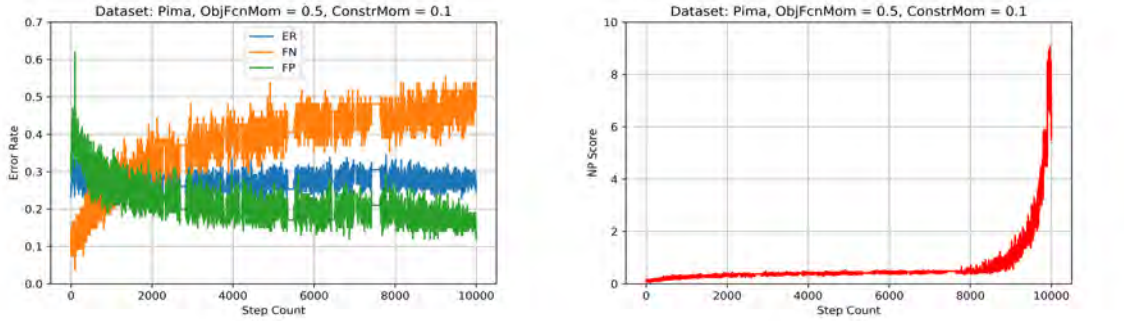


Figure A.51. Pima dataset results for objective function momentum = 0.5, constraint momentum = 0.1

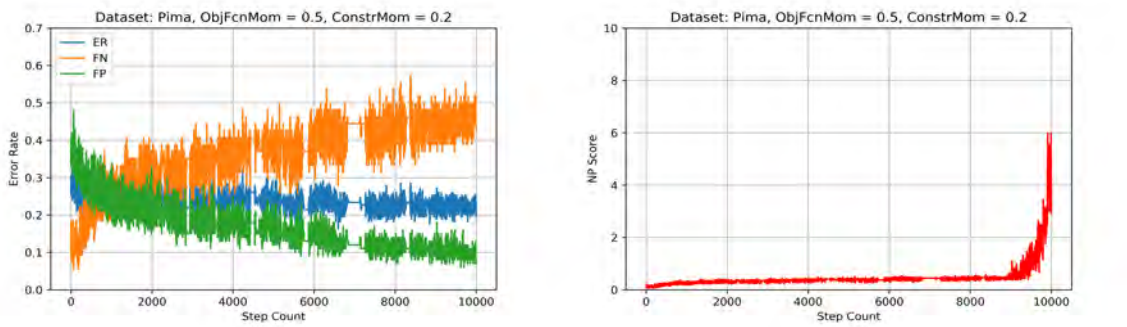


Figure A.52. Pima dataset results for objective function momentum = 0.5, constraint momentum = 0.2

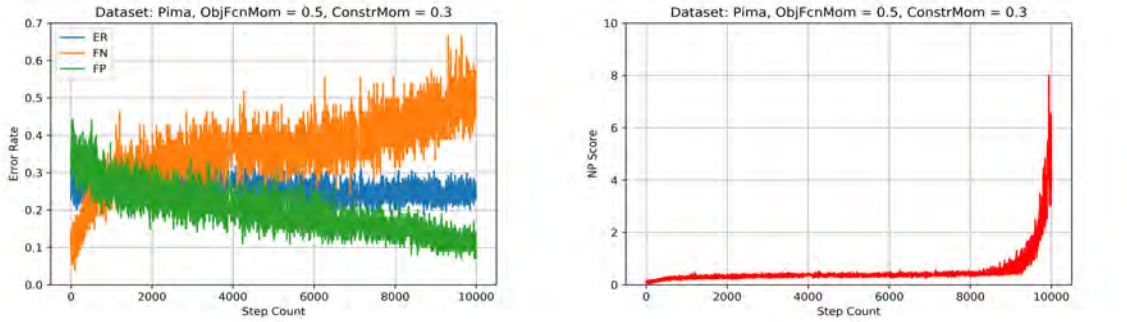


Figure A.53. Pima dataset results for objective function momentum = 0.5, constraint momentum = 0.3

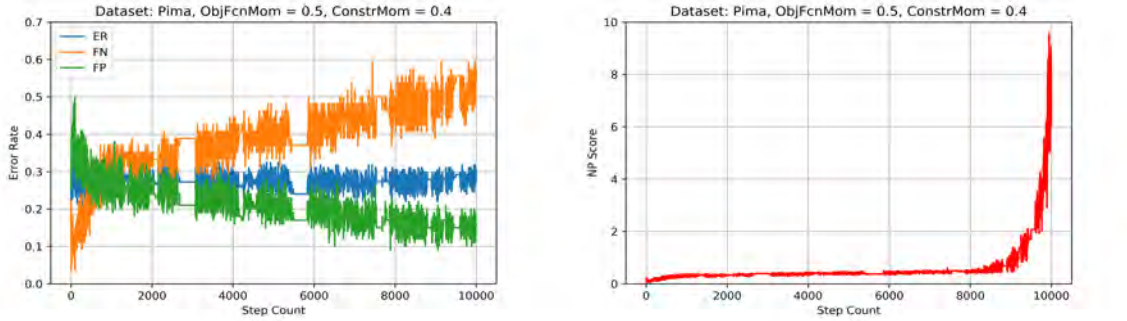


Figure A.54. Pima dataset results for objective function momentum = 0.5, constraint momentum = 0.4

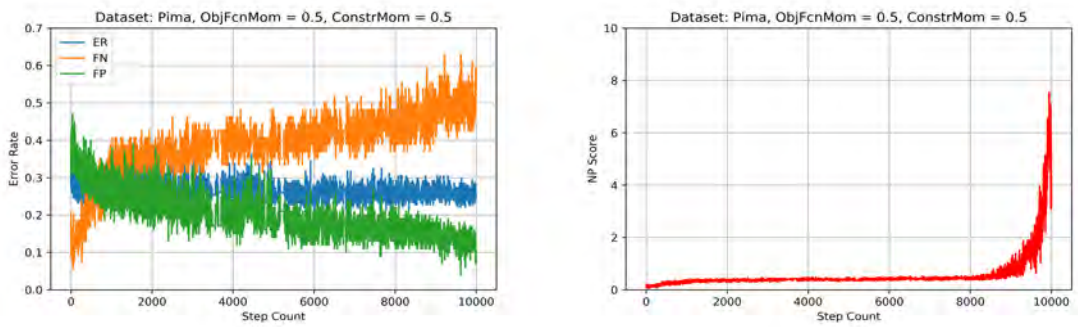


Figure A.55. Pima dataset results for objective function momentum = 0.5, constraint momentum = 0.5

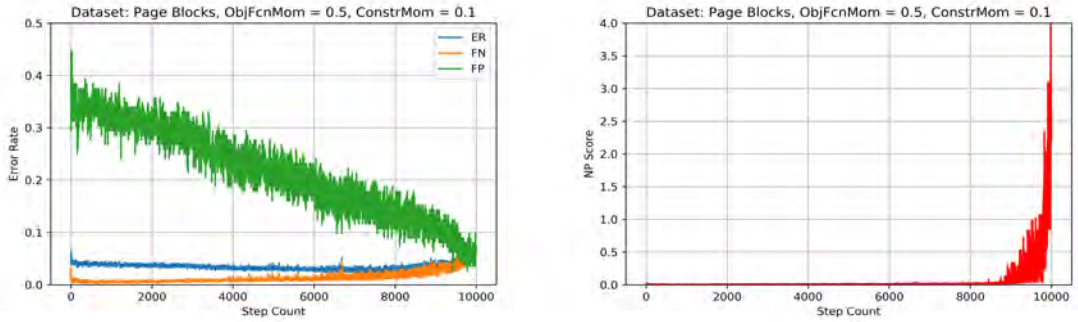


Figure A.56. Page Blocks dataset results for objective function momentum = 0.5, constraint momentum = 0.1

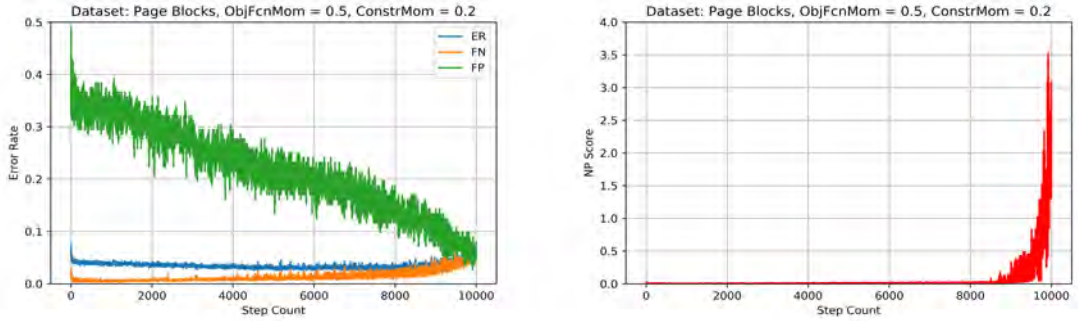
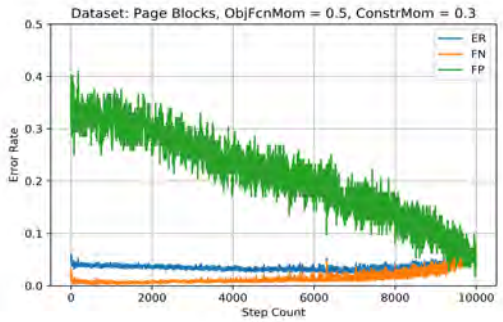


Figure A.57. Page Blocks dataset results for objective function momentum = 0.5, constraint momentum = 0.2



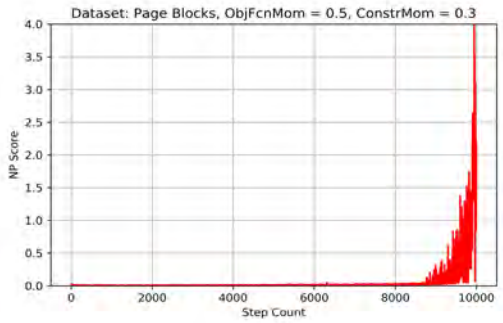
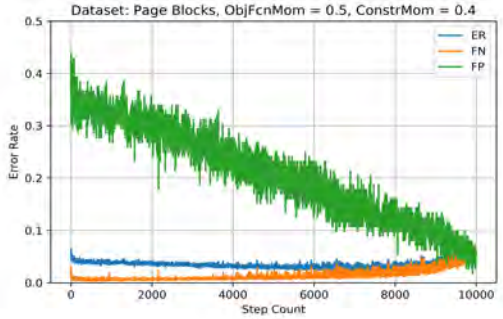


Figure A.58. Page Blocks dataset results for objective function momentum = 0.5, constraint momentum = 0.3



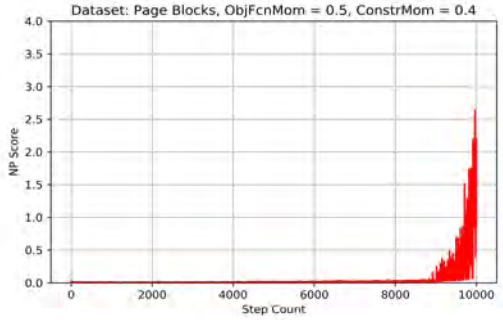
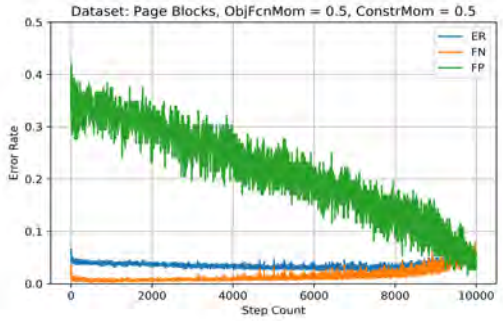


Figure A.59. Page Blocks dataset results for objective function momentum = 0.5, constraint momentum = 0.4



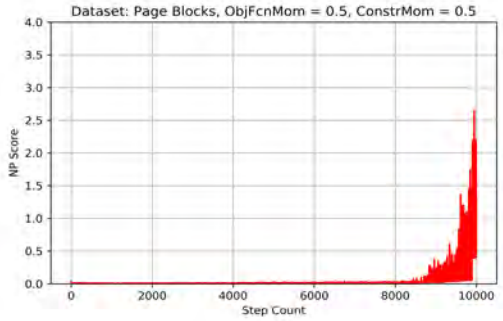


Figure A.60. Page Blocks dataset results for objective function momentum = 0.5, constraint momentum = 0.5

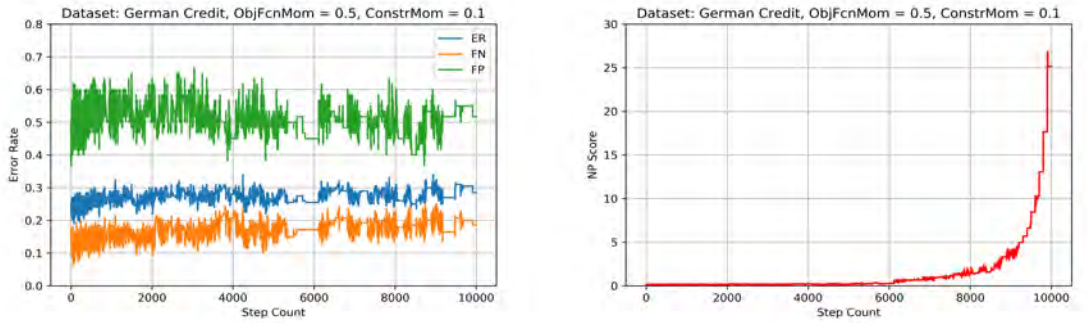


Figure A.61. German Credit dataset results for objective function momentum = 0.5, constraint momentum = 0.1

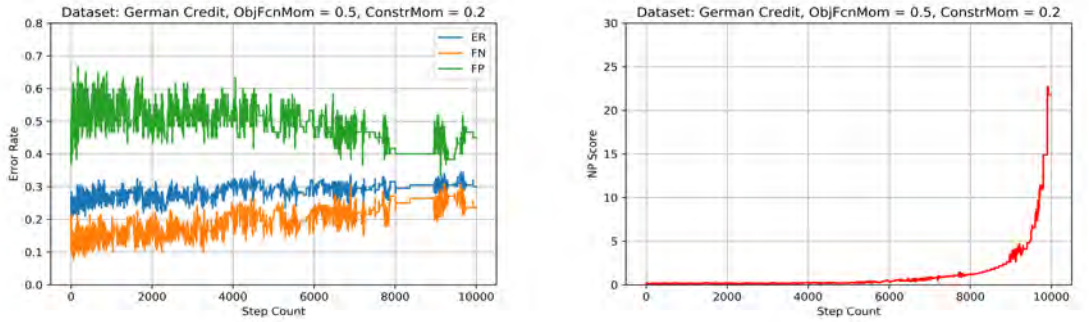


Figure A.62. German Credit dataset results for objective function momentum = 0.5, constraint momentum = 0.2

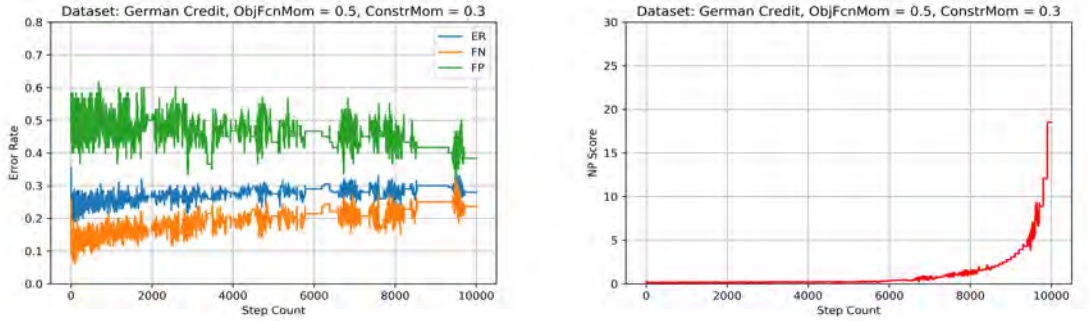


Figure A.63. German Credit dataset results for objective function momentum = 0.5, constraint momentum = 0.3

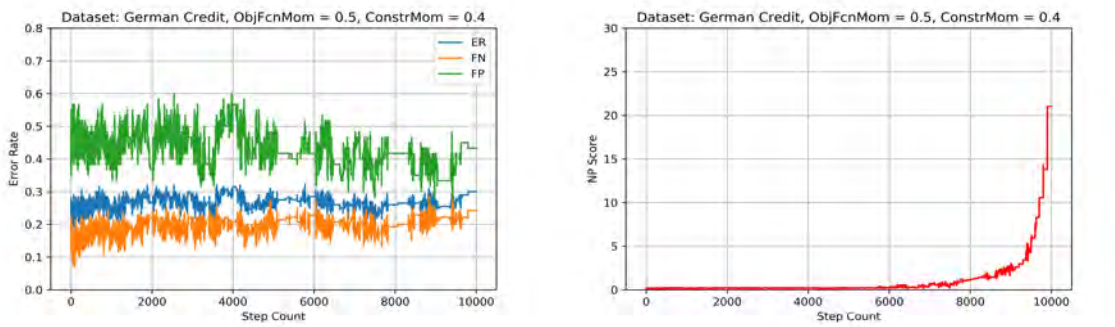


Figure A.64. German Credit dataset results for objective function momentum = 0.5, constraint momentum = 0.4

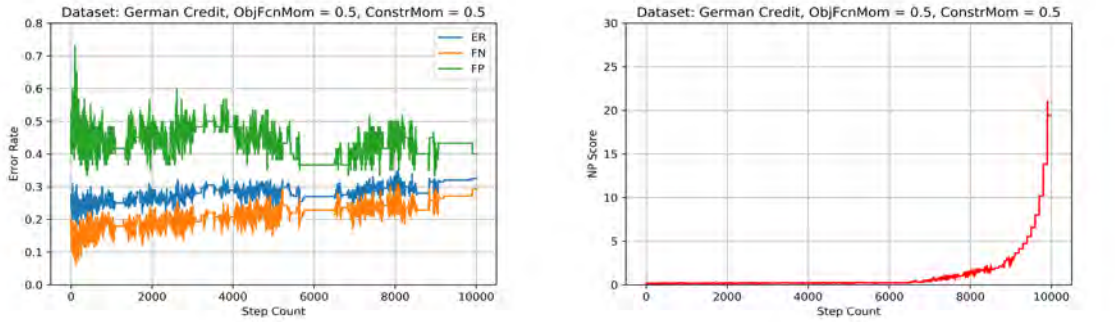


Figure A.65. German Credit dataset results for objective function momentum = 0.5, constraint momentum = 0.5

From Figures A.61 to A.45, there is negligible improvement in using constraint function momentums that are smaller than the objective function momentum. Therefore, we use a constraint function momentum of 0.5 for the rest of the analyses.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, United Kingdom).

Cull KM (2018) *Classifying Vessels Operating in the South China Sea by Origin with the Automatic Identification System*. Master's thesis, Naval Postgraduate School, Monterey, CA, https://calhoun.nps.edu/bitstream/handle/10945/58289/18Mar_Cull_Kimberly.pdf?sequence=1&isAllowed=y.

Davenport MA, Baraniuk RG, Scott CD (2010) Tuning support vector machines for minimax and neyman-pearson classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32(10):1888–1898.

Ghadimi S, Lan G, Zhang H (2016) Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming* 155(1-2):267–305.

Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning* (MIT Press, Cambridge, MA).

James G, Witten D, Hastie T, Tibshirani R (2013) *An Introduction to Statistical Learning*, volume 112 (Springer, New York, NY).

Norton M, Uryasev S (2017) Error control and Neyman-Pearson classification with buffered probability and support vectors. Accessed May 18, 2019, https://www.mnortonlab.com/?page_id=38.

PyTorch (2019) Introduction to PyTorch. Accessed April 24, 2019, https://pytorch.org/tutorials/beginner/nlp/pytorch_tutorial.html#sphx-glr-beginner-nlp-pytorch-tutorial-py.

Scott C (2005) Comparison and design of Neyman-Pearson classifiers. Accessed April 24, 2019, http://www.stat.rice.edu/~cscott/pubs/npdesign.pdf.

Scott C (2007) Performance measures for Neyman-Pearson classification. *IEEE Transactions on Information Theory* 53(8):2852–2863.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California