The 14th International Conference on Advanced Computational Intelligence (ICACI)
July 15-17 2022, Wuhan, China

In Pursuit of the Best Detection of Positive Data Under User's Concern on False Alarm

1st Cong Teng, 2nd Liyan Song*
[1] *Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, China*
[2] *Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation,*
Department of Computer Science and Engineering, Southern University of Science and Technology, China
Shenzhen, China
12132358@mail.sustech.edu.cn, songly@sustech.edu.cn

2022 14th International Conference on Advanced Computational Intelligence (ICACI) | 978-1-6654-7045-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICACI55529.2022.9837710

***Abstract*—Just-In-Time Software Defect Predict (JIT-SDP) has been a popular research topic in the literature of software engineering. In many practical scenarios, software engineers would prefer to pursue the best detection of defect-inducing software changes under the concern of a given false alarm tolerance. However, there have been only two related studies in the Machine Learning (ML) community that are capable of tackling this constraint optimization problem. This paper aims to study how can we utilize the existing ML methods for addressing the research problem in JIT-SDP and how well do they perform on it. Considering the fact that the objective and the constraint are not differentiable, a Differential Evolution (DE) algorithm is by nature suitable for tackling this research problem. Thus, this paper also aims to investigate how can we propose a novel DE algorithm to better address the constraint optimization problem in JIT-SDP. With these aims in mind, this paper adapts the ML methods with a spared validation set to facilitate the constraint learning process, and it also proposes an advanced DE algorithm with an adaptive constraint to pursue the best detection of the positive class under a given false alarm. Experimental results with 10 real-world data sets from the domain of software defect prediction demonstrate that our proposed DE based approach can achieve generally better performance on the constraint optimization problem, deriving better classification models in terms of both objective and the constraint.**

***Index Terms*—differential evolution, the constraint optimization problem, machine learning, false alarm, just-in-time software defect predict, software engineering**

I. Introduction

In the process of software development and maintenance, software defects are inevitable, and once they cannot be detected and fixed in time, software quality would be largely affected, causing poor user experience and even resulting in significant economic losses [1]. For example, a plane's navigation system failure would possibly lead to a catastrophic airplane crash. Research has shown that the cost of defective software detection and repair accounts for 50% $\sim$ 75% of the total development cost [2]. Therefore, identifying software defects as soon as possible is very important for improving the software quality and safety in practice.

Just-In-Time Software Defect Prediction (JIT-SDP) is suitable for these application scenarios [1], [3]. It proceeds on the fine-grained software change level that is submitted by developers as a unit. Software changes that introduce defects in the software system are *defect-inducing*, and those that do not cause future defect are *clean*. This fine-grained predictive technology makes it easier and more instant for software developers to detect software defects and trace them back, improving the software quality and safety in the real-world application scenarios.

JIT-SDP can be modeled as a binary classification problem, where defect-inducing software changes belong to the positive class, and clean ones belong to the negative class [3]. JIT-SDP aims to detect defect-inducing software changes as many as possible while not to induce too high false alarm of the clean software changes. For this, we can set up the objective to pursue the best recall of the positive class under the concern of a given false alarm constraint on the negative class.

We use $\mathscr{D} = \{(X_i, y_i), i = 1,\ldots, m\}$ to denote a data set of software changes, where $X_i \in \mathbb{R}^d$ is a d-dimensional feature vector associating to a software change, $y_i \in \{0, 1\}$ is the class label being "+1" for a defect-inducing software change and "0" for a clean one. Software changes in $\mathscr{D}$ are chronological, meaning that (X_{i+1}, y_{i+1}) comes after (X_i, y_i) at the commit time for any $i \in \{1, \cdots, m\}$. Based on these notations, our research problem can be formulated as

$$\begin{aligned} max \quad & Recall(f) \\ s.t. \quad & False\_alarm(f) \leq \tau, \end{aligned} \tag{1}$$

where $f(\cdot)$ denotes a binary classifier for JIT-SDP that is trained on $\mathscr{D}$, and τ denotes a false alarm tolerance given by (e.g.) project managers or software developers in software development organizations.

Different practical scenarios may have different levels of false alarm tolerance, giving rise to different pressure on the pursuit of positive data detection. For example, the aerospace industry usually prefers a higher recall to detect software defects as many as possible for the safety purpose and has a relatively heavier pressure on the false alarm. Whereas,

* Liyan Song is the corresponding author. This work was supported by the National Natural Science Foundation of China (Grant No. 62002148), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), and Research Institute of Trustworthy Autonomous Systems (RITAS).

as game companies can rely on players' feedback to detect software defects throughout the stage of software development and maintenance, they could set up a lower recall, anticipating the assistance from public players, and establish a lighter false alarm pressure to save human cost. Therefore, this paper aims to investigate how to pursue the best detection of positive class under user's concern on false alarm.

To the best of our knowledge, there have been only two Machine Learning (ML) based approaches [4], [5] that can handle the constraint optimization problem in Eq. (1). Davenport et al. [4] trained a classifier disregarding the constraint, and then adjusted the classification threshold to find the largest false alarm in line with the given constraint. In this paper, we call their method the "shift threshold approach". Broadwater et al. [5] coped with the constraint via the adjustment of the loss function that gradually reduced the false alarm until a given constraint was met. In this paper, we call their method the "dynamic loss approach".

However, there has been no study investigating how well existing ML methods can perform on the constraint optimization problem of JIT-SDP. As will be discussed in Sec. VI-A, directly applying these ML methods to the research problem of Eq. (1) in JIT-SDP cannot obtain good performance in terms of satisfying the given constraint. To tackle this issue, we propose to adapt these two methods by employing a spare validation set based on which we can facilitate their techniques in dealing with the constraint.

Moreover, considering that both the objective and the constraint of the research problem are not differentiable, evolutionary algorithms, such as Differential Evolution (DE) [6], are by nature suitable to tackle this problem. Accordingly, we aim to propose an alternative approach to address the research problem based on the DE framework, which may have the potential to gain better recall while have higher probability for meeting a given false alarm constraint. We name our approach as an "advanced DE algorithm with an adaptive constraint".

This paper aims to investigate the way to detect more defect-inducing software changes under user's concern on a certain false alarm. We answer the following research questions:

RQ1 How well can existing (adapted) ML methods perform on the constraint optimization problem in JIT-SDP?

RQ2 How can we propose a novel approach based on the DE framework to better address the constraint optimization problem and how well does our approach solve the research problem in JIT-SDP?

Our main contributions are:

- we introduce and adapt the existing ML based approaches to solving the research problem (Eq. (1)) in JIT-SDP by proposing a spare validation set;
- we proposes an advanced DE algorithm with an adaptive constraint, and experimentally justifies its effectiveness in providing good prediction on recall and meeting the constraint of false alarm.

II. Background and Related Work

A. Just-In-Time SDP

Kim et al. firstly presented the idea of JIT-SDP [7], which was later given by this formal name by Kamei et al. [3]. Compared with the traditional software defection prediction that is on the level of coarse-grained files, modules, or packages [8]–[10], JIT-SDP predicts software changes submitted by developers. It has three features: fine-grained, real-time, and easy traceability [1]. JIT-SDP focuses on fine-grained software changes so that developers can spend less time and effort reviewing software changes that are predicted to be defective than coarse-grained SDP. JIT-SDP can predict software changes as soon as they are submitted, while the developer is still impressed with the changes they just submitted, which makes it easier to fix defects. At the same time, software changes contain the information of the submitter, which can be traced back to the submitter during defect repair, facilitating better analysis and repair.

Kamei et al. [3] summarized 14 basic features of software change that were shown to perform well in JIT-SDP. These features can be divided into five groups including diffusion, size, purpose, history and experience. These features of different dimensions reflect the important information of software changes, which are of great significance.

JIT-SDP is chronological, and the generation of samples of software changes is in a chronological order. In the process of prediction, future samples should not be used to predict the past data, otherwise the performance of the model will be optimistic [11]. Meanwhile, the nature of the defect-inducing changes will change over time [12], so new software changes should be predicted using new data generated recently.

A variety of classification models have been applied to JIT-SDP, such as Hoeffding tree [13], random forest [14], and deep learning approaches [15]. Among them, the logistic regression [16] is a simple model that has been widely used in JIT-SDP [3], [17]–[19]. This paper adopt the logistic regression model as the base model to carry out a comparative study between different approaches.

B. Existing Methods in the ML Community

As mentioned in Section I, to the best of our knowledge, there are two classification approaches for false alarm constraint in machine learning, for which we name as the shift threshold approach [4] and the dynamic loss approach [5].

The shift threshold approach manipulates the false alarm of the negative class by adjusting the classification threshold of the model. Classification threshold is an important parameter in the model. A linear model will produce an output value after computing the input data. When the output value is greater than the classification threshold, the data will be classified into the positive class, otherwise it will be classified to the negative class. After the model is trained, the shift threshold approach adjusts the classification threshold of the model to find the classification threshold of the maximized objective function that meets the false alarm constraint. The best shifted

threshold is used as the final classification threshold of the model to predict future software changes.

The dynamic loss approach manipulates the false alarm of the negative class by dynamically adjusting the loss function of the model. Specifically, a false alarm penalty term is added to the loss function of the model. If the trained model does not meet the false alarm constraint, the weight of the false alarm penalty term is increased and the training is conducted again. The process is repeated until the model meets the constraint.

C. Evolutionary Algorithm

The evolutionary algorithm is a global optimization approach. It has high robustness and wide applicability. For the complex problems that are difficult to be solved by traditional optimization algorithms, evolutionary algorithm can effectively deal with them without the limitation of problem nature [6].

There have been some studies that used the evolutionary algorithm in SDP or JIT-SDP. Some work uses evolutionary algorithms for feature selection [20]–[22]. Some work uses evolutionary algorithms to search for model parameters [23], [24], and the models with the selected model parameters are then used for prediction. In this case, the evolution process replaces the traditional training algorithm of classification models. This paper follows the latter approach, considering the weights and biases of the logistic regression model as an individual. Since the representation of an individual is a real number vector, some evolutionary algorithms are not applicable. In this paper, Differential Evolution (DE) [25] is adopted for evolution.

III. The Adaptation of ML-based Approaches

To the best of our knowledge, there have been only two machine learning based approaches for solving the problem of Eq. (1). In this paper, we call the two methods the shift threshold [4] and the dynamic loss [5], respectively. However, as will be discussed in Sec. VI-A, our experimental results showed that adopting the initial setup of these approaches can usually fail the constraint. Our conjecture is that due to the data shift of JIT-SDP, training set can be obsolescent to construct the models that are then used for future prediction on the test set. Therefore, we propose to spare a validation set to facilitate the searching direction of the classification threshold for the shift threshold approach and the regulation term for the dynamic loss approach, leading to their adapted versions.

This paper adapts the existing ML-based approaches for solving the problem in Eq. (1) with the logistic regression model that has been popularly used in JIT-SDP [3], [17], [18]. Particularly, we refer to the Receiver Operating Characteristic (ROC) curve [26] to search the classification threshold of the shift threshold approach, and modify the loss function of logistic regression to set the false alarm penalty for the dynamic loss approach.

A. The Adapted Shift Threshold Approach

Logistic regression performs linear calculation of the input features, then uses Sigmoid function to map the calculated values to between 0 and 1. Ultimately, the classification prediction is conducted by comparing the output against the classification threshold as:

$$\hat{y} = \begin{cases} 0 & pr \leq \mu \\ 1 & pr > \mu \end{cases} \tag{2}$$

where μ is the classification threshold, pr is the probability value calculated by the Sigmoid function, and $\hat{y}$ denotes the predicted label. Specifically, if pr is greater than the classification threshold μ, the data is classified to the positive class; if pr is less than the classification threshold, the data is classified to the negative class. Conventionally, the logistic regression model has a classification threshold of 0.5.

The core idea of the shift threshold approach is to find the maximum recall classification threshold that meets the false alarm constraint after the model is trained, and take it as the classification threshold of the model. Given a training set $\mathscr{D}_{train}$ and a validation set $\mathscr{D}_{valid}$, we present the training process of the adapted shift threshold approach with the following steps.

Step 1 Train a logistic regression model $M_o(\cdot)$ based on the training set $\mathscr{D}_{train}$.

Step 2 Evaluate the ROC curve of $M_o(\cdot)$ on $\mathscr{D}_{train}$, each point of which corresponds to recall, false alarm, and a classification threshold μ_i.

Step 3 Sort $\{\mu_i\}$ in an ascending order. Find the maximal recall whose false alarm meets the constraint, and adopt the corresponding classification threshold μ_{train}.

Step 4 Repeat Step 2 and Step 3 on the validation set $\mathscr{D}_{valid}$ to get the corresponding classification threshold μ_{valid}.

Step 5 Choose the smaller one between μ_{train} and μ_{valid} as the final classification threshold μ_{use}.

Step 6 Adjust $M_o(\cdot)$ to get the final classification model $M_{used}(\cdot)$ with the classification threshold μ_{use}.

B. The Adapted Dynamic Loss Approach

The dynamic loss approach controls the false alarm by dynamically adjusting the loss function of the classification model. The loss function of a logistic regression model is formulated as:

$$\begin{aligned} J(\theta) = \quad & -\tfrac{1}{m}\sum_{i=1}^{m} y^i \log(\sigma(\theta^T \cdot X^i)) \\ & +(1-y^i)\log(1-(\sigma(\theta^T \cdot X^i))), \end{aligned} \tag{3}$$

where θ represents the parameters of logistic regression model, X denotes input features, y is the label of the input data, and $\sigma(\cdot)$ denotes the Sigmoid function.

We can see that the loss function Eq. (3) can be divided into two parts. The first part contains $y^i \log(\sigma(\theta^T \cdot X^i)$ relating to the positive class, and it is valid for $y = 1$. Conventionally, the higher its value, the higher the recall performed by the model. The second part contains $(1-y^i)\log(1-(\sigma(\theta^T \cdot X^i)))$ relating to the negative class, and it is valid for $y = 0$. Conventionally, the higher its value, the lower the false alarm of the model.

The core idea of the dynamic loss approach is to control the model's false alarm by adding a false alarm penalty term

to the model's loss function dynamically. With this in mind, the second part of the loss function Eq. (3) can be regarded as the false alarm penalty. The heavier the penalty is, the lower the false alarm is. Meanwhile, the remaining part is positively related to recall, which is the objective of the research problem in Eq. (1). Therefore, the loss function of the logistic regression model can be rephrased as:

$$\begin{aligned} J(\theta) = \quad & -\tfrac{1}{m}\sum_{i=1}^{m} y^i \log(\sigma(\theta^T \cdot X^i)) \\ & +\lambda(1-y^i)\log(1-(\sigma(\theta^T \cdot X^i))). \end{aligned} \tag{4}$$

Specifically, we put an extra parameter λ to the second part of the loss function Eq. (3) as the penalty coefficient to pursue the constraint commitment. Given a training set $\mathscr{D}_{train}$ and a validation set $\mathscr{D}_{valid}$, we present the training process of the adapted dynamic loss approach with the following steps.

Step 1 Initialize λ with 0 and the model parameter with $\theta = 0$.
Step 2 Train the logistic regression model from the current parameter θ based on the loss function Eq. (4), for which the new model is denoted as θ'.
Step 3 Set $\theta = \theta'$, $\lambda = \lambda + \Delta\lambda$.
Step 4 If the model's false alarm on $\mathscr{D}_{train}$ and $\mathscr{D}_{valid}$ does not meet the constraint, return Step 2; else, go Step 5.
Step 5 Return the learned model with parameters θ.

Particularly, $\Delta\lambda$ is the changing value of the penalty coefficient λ. A large $\Delta\lambda$ would cause faster drop in both false alarm and recall; whereas a small $\Delta\lambda$ would probably make the algorithm hard to converge. In this paper, $\Delta\lambda$ is set to 0.05 as the trade-off between the two issues. Moreover, the simple random oversampling technique [27] is adopted in this adapted approach to tackle the class imbalance issue.

IV. An Advanced DE with an Adaptive Constraint

This section aims to propose an advanced DE approach for the constraint optimization problem of Eq. (1), for which can adaptively tune the constraint value to pursue a good detection of defect-inducing changes while meet the predefined constraint of false alarm. We will first introduce how to tackle the constraint optimization problem based on the conventional DE framework, and then present the specialized treatment on the constraint in terms of an adaptive constraint algorithm.

A. The Framework of An Advanced DE with An Adaptive Constraint

To solve the optimization with constraint in Eq. (1), we set our searching framework based on the conventional DE algorithm and opt for logistic regression models as individuals of the population to perform the classification task within the problem. A logistic regression model is represented as:

$$\hat{y} = \sigma(b + w_1 x_1 + \cdots + w_n x_n), \tag{5}$$

where x_i is the i-th input feature of a training sample X, w_i is the weight corresponding to the feature x_i, b is the bias of the linear model, and $\sigma(z) = (1+e^{-z})^{-1}$ is the Sigmoid function to make a binary classification. Specifically, the parameter vector of logistic regression $W = (w_1, \ldots, w_n, b)$ is an individual of the population.

As the objective (recall on the positive class) and the constraint (false alarm) of each individual should be measured based on data samples, we need to set up training data set for individuals (i.e., classifiers) to evaluate their predictive performance in terms of recall and false alarm. Given a classifier W, its recall on the training set corresponds to the individual's fitness function, formulated as

$$f(W) = Recall_{train}(W). \tag{6}$$

The false alarm of the classifier is used to justify whether the model can meet the constraint. If the false alarm is smaller than the given constraint tolerance τ of Eq. (1), this individual meets the constraint, being a feasible solution; otherwise, the individual is an infeasible solution.

The proposed approach follows the conventional DE framework in searching individuals that can achieve good recall and meet the user's concern on false alarm. Specifically, we adopt the Latin Hypercube Sampling approach [28] to initialize the population, so that the initial population can be dispersed as far as possible in the search space. The DE/rand/1/bin operator is used with the dither scale factor between 0 and 1 [29] to enhance the global search ability.

To facilitate the conventional DE algorithm with the ability of dealing with the constraint, we adopt the feasible rule [30] for the selection of the next generation. The degree an individual W violates the constraint is measured as

$$g(W) = \max(0, False\_alarm_{train}(W) - \tau), \tag{7}$$

where $False\_alarm_{train}(W)$ is the false alarm that an individual W can perform on the training set, and τ is the false alarm tolerance that may be raised by customers or software developers. We refer to the violation degree of Eq. (7) to proceed the selection step of DE between the parent and its child. We can see that $g(W) = 0$ indicates a feasible individual that meets the constraint of Eq. (1). In this case, we can select the individual with higher fitness value; whereas, $g(W) > 0$ indicates that W is an infeasible solution and that the corresponding classifier cannot meet the constraint. In this case, we select the individual with the smaller $g(W)$.

In the measurement of Eq. (7), the constraint τ is static in the evolution process of DE. As will be shown in Section VI-A, data samples of JIT-SDP frequently suffer from the data shift problem and as a result, the resulting models which are evolved by a static τ could often violate the constraints on the future test data set even though they meet the constraint on the training set.

To mitigate this issue, we propose to adaptively adjust the constraint value, forming an adaptive constraint τ^* during the evolution process. By replacing the static constraint τ with the adaptive one τ^*, *an advanced DE algorithm with an adaptive constraint* is proposed. In this case, the degree an individual W violates the constraint can be reformulated as

$$g^*(W) = \max(0, False\_alarm_{train}(W) - \tau^*), \tag{8}$$

Algorithm 1 The framework of the advanced DE algorithm with an adaptive constraint.

Inputs: (1) the training set $\mathscr{D}_{train}$, (2) a specific constraint value τ, (3) population size N, (4) maximal generations T, and (5) the proportion of optimal individuals a.

Outputs: an ensemble model $M_{ensemble}$.

1: Set the adaptive constraint value as $\tau^* = \tau$.
2: Initialize population with N individuals by Latin Hypercube Sampling.
3: **for** generation $t = 1$ until T **do**
4: Mutation, Crossover: for each individual, conduct the DE/rand/1/bin operation to generate an offspring, each of which corresponds to one parent.
5: Evaluation: calculate fitness value as in Eq. (6) and constraint violation as in Eq. (8) for all the individuals.
6: Selection: select the better one from a pair of the parent and its offspring and pass this individual onto the next generation by the feasible rule.
7: Adjust τ^* as in Algorithm 2 with the current population P and the current τ^*.
8: **end for**
9: Select the best $a \times N$ individuals by feasible rule to generate a ensemble model $M_{ensemble}(\cdot)$ with the majority voting rule.

where $False\_alarm_{train}(W)$ is the false alarm that the model W can achieve based on the training set. Especially, when the constraint value τ^* is tuned down, higher pressure is posed to the false alarm that classifiers would perform on future data set, improving the probability these classifiers can possibly meet the constraint in future. The crucial point is how to set the adaptive constraint τ^* during the evolution process, for which we will discuss in Section (IV-B).

Algorithm 1 summarizes the proposed advanced DE algorithm with an adaptive constraint. In this paper, the population size N, the maximum generation T and the crossover rate, are set to 100, 1000, 1.0, respectively. Especially, the algorithm selects the top $a \times N$ individuals to generate an ensemble of classifiers for future prediction, for which N is the population set. We set $a = 10\%$ in this paper, as such amount can retain most good individuals while remove those that cannot perform well. The majority voting is adopted to perform the final prediction, for which the final predictions are determined by the majority classes of the selected individual models.

B. An Adaptive Constraint

This section aims to detail how to automatically set an adaptive constraint τ^* of Eq. (8). The procedure is summarized in Algorithm 2.

Especially, to facilitate the searching directions in view of pursuing individuals that can better meet the constraint, we propose to employ a validation set. That being said, the validation set is independent of the training set in the adjusting process of the adaptive constraint τ^*. It will only be used to evaluate the constraint violation of an individual when determining how to change the adaptive τ^*. The reason for sparing a validation set is to estimate the performance that the corresponding model would possibly perform in an unknown prediction circumstance, based on which the constraint search for τ^* can be facilitated in an adaptive way.

Algorithm 2 The algorithm of the adaptive constraint τ^*.

Inputs: (1) the validation set $\mathscr{D}_{valid}$, (2) a specific constraint value τ, (3) current adaptive constraint value τ^*, (4) population size N, (5) current population P, (6) the proportion of optimal individuals a and (7) the shifting ratio c.

Outputs: adaptive constraint value τ^*.

1: Select the best $a \times N$ individuals from P by feasible rule. Calculate their $False\_alarm_{valid}$ on validation set.
2: **if** there is a $False\_alarm_{valid} > \tau$ **then**
3: $\tau^* = (1 - c)\tau^*$
4: **else if** there are more than half $False\_alarm_{valid} \leq (1 - a)\tau$ **then**
5: $\tau^* = \min(\tau, (1 + c)\tau^*)$
6: **else**
7: $\tau^* = \tau^*$
8: **end if**

We can see from Step 1 of Algorithm 2 that, the optimal $a \times N$ individuals are selected to decide a better searching direction of constraint values based on the validation set, with the same reason for the model ensemble at Step 9 of Algorithm 1, where a is the proportion of the optimal individuals. Given the chosen individuals, if there is any one violating the given constraint τ based on the validation set, the adaptive constraint value τ^* would be reduced to $(1 - c)\tau^*$ (Step 3) for which $c \in (0, 1)$ denotes a shifting ratio. In this way, we can pursue a searching direction that can probably meet the constraint in future. Whereas, if half of the chosen individuals have their false alarms being no larger than $(1-a)\tau$ based on the validation set, the adaptive constraint value τ^* is updated to $\min(\tau, (1 + c)\tau^*)$ (Step 5), slightly relaxing the adaptive constraint. This hints the improvement room in terms of pursuing a higher recall.

Ultimately, we can set the adaptive constraint value τ^* throughout the evolution process of the proposed advanced DE algorithm with adaptive constraint. In this paper, the shifting ratio c is set to 1%. Too large c would cause a severe change of the adaptive constraint value τ^*, causing too many good solutions becoming infeasible and to be eliminated in the search space; too small c would result in the change of the adaptive constraint τ^* being too small, requiring more generations to evolve. In this paper, it is set to 1%, which can reach the desired optimal constraint value within a proper number of generations without losing too much information about optimal solutions, so as to carry out deeper space exploration.

V. Experimental Setup

A. Dataset

Our studies use 10 real-world data sets to investigate the effectiveness of our approach in addressing the constraint optimization problem in JIT-SDP, as shown in Table I. They were chosen among projects with more than 2 years of duration, rich history (>15k commits) and a wide range of defect-inducing

TABLE I: An overview of the datasets. The last 10,000 software changes (commits) are used in our experiments.

Project	Total Changes	Defect-inducing	%Defect-inducing	Time Period
Ansible	44,817	16,113	35.95	02/2012 - 07/2020
VScode	63,246	1,309	2.07	11/2015 - 07/2020
wp-Calypso	36,057	9,913	27.49	01/2014 - 07/2020
Elasticsearch	49,128	17,262	35.14	02/2010 - 07/2020
Googleflutter	17,842	227	1.27	10/2014 - 07/2020
Pytorch	27,838	11,150	40.05	01/2012 - 07/2020
Rails	59,825	14,240	23.80	11/2004 - 07/2020
Rust	91,623	1,469	1.60	06/2010 - 07/2020
Tensorflow	55,656	15,041	27.02	11/2015 - 07/2019
Homebrew	44,044	950	2.16	09/2017 - 03/2019

change ratio ($1\% \sim 40\%$). The `Commit Guru` [19] tool was used to collect the data with 14 features. The labels are defect-inducing (positive class) or clean (negative class). We then follow Kamei et al.'s [3] to conduct the data pre-process. For a uniform investigation, we use the last 10k software changes of each project in the experiments, which are ordered chronologically.

Our studies involve the justification of the requirement of a validation set. Thus, the total 10k software changes are divided into the training and test sets with the ratio of 8:2 or into the training, validation, and test sets with the ratio of 6:2:2 in the chronological order.

B. Performance Evaluation

To the best of our knowledge, there has been no existing method in the domain of JIT-SDP targeting on solving the constraint optimization problem of Eq. (1). In the ML community, there have been only two classification models that were adapted to cope with this problem as discussed in Sec. III. Therefore, experimental studies are carried out on the two approaches and the proposed advanced DE algorithm with adaptive constraint. In this paper, we investigate the performance of the three approaches with the constraint tolerance values $[0.2, 0.3, 0.4]$ (τ in Eq. (1)), individually.

Performance of the investigated methods is evaluated from three aspects: the recall on the positive class, whether the prediction meets the constraint of false alarm, and the overall performance in terms of G-mean. Comparisons are conducted based on the mean performance across 20 runs. Specifically, recall of the positive class (corresponding to the defect-inducing software changes in JIT-SDP) is the objective in Eq. (1). The higher the recall is, the better predictive performance a prediction model can achieve. We use tp to denote the number of positive samples that are correctly predicted, and use p to denote the total number of positive samples. The objective can be formulated as $Recall = tp/p$.

The constraint in terms of false alarm is formulated as fp/n, where fp denotes the number of negative samples that are misclassified to a positive class, and n denotes the total number of negative samples. We can see that false alarm is associated with the performance of classifiers on the negative class. Given a classifier that performs 20 times on the spared test set, if we cannot detect significant difference between the achieved false alarm and the given tolerance τ across multiple data sets and 20 runs, it is concluded that the model can meet the constraint. The overall performance of the model is measured in G-mean and calculated as $\sqrt{Recall \times (1 - False\_alarm)}$.

VI. Experimental Results

A. A Spared Validation Set Is Required

As previously mentioned in Sec. IV-A, this section aims to demonstrate the data shift issue suffered by chronological software changes, motivating the validation set for solving the constraint optimization problem in JIT-SDP.

Table IIa shows the false alarm values of the shift threshold approach for each individual tolerance τ. The last row presents the maximal constraint violation across data sets. We can see that the investigated ML approach usually violate the constraint τ in a relatively large magnitude. For example, the shift threshold approach always fails the constraint of a given false alarm tolerance in wp-Calypso. Specifically, though the approach can meet the constraint of $\tau = 0.2$ in the training set, the shift threshold approach can only achieve the false alarm 0.3046 in the spared test set.

A potential reason for this phenomenon is that the training set that consists of software changes committed at earlier time slot may become obsolescent to construct classification models that are used for predicting the labels of software changes in the future test set, i.e., the samples in JIT-SDP probably suffers from the data shift issue [31]. To justify this conjecture, we adopt the stratified sampling technique that evenly divides the entire data set into the training and test sets, so that the data shift issue can be alleviated on the data level. Table IIb presents the false alarm of the shift threshold approach performing on the stratified data sets. We can see that the false alarm values achieved by the same approach can be largely improved. For example, in wp-Calypso, this approach can gain the false alarm that is very close to its corresponding tolerance. This observation can confirm our conjecture that software changes in JIT-SDP suffers from the data shift issue.

As explained in Sec. III, to cater for this issue, we propose to spare a validation set to facilitate the search for the classification threshold of the shift threshold approach. Table IIc presents the false alarm the shift threshold approach can perform when adopting the validation set. We can see that, though the adapted approach cannot perform as good as in Table IIb, it can better meet the constraints compared with the case that does not spare the validation set.

Dynamic loss function approach has similar results in terms of the false alarm constraints and thus is omitted in this paper. Therefore, the adapted ML approaches are used to investigate the answers to RQ1 and RQ2 from here onward.

B. How Well Do the Adapted ML Methods Solve the Constraint Optimization of JIT-SDP?

This section aims for answering part of RQ1 by investigating the performance of the adapted ML approaches in solving the constraint optimization problem in JIT-SDP. Predictive performance of the two methods in terms of recall (the

TABLE II: Experimental results of false alarm that the original and adapted shift threshold methods perform across data sets.

(a) Training and test sets in the chronological order of software changes in JIT-SDP.

Dataset	false alarm tolerance τ		
	0.2	0.3	0.4
Ansible	0.2971	0.4181	0.5114
wp-Calypso	0.3046	0.4220	0.5228
Elasticsearch	0.2867	0.3977	0.5107
Googleflutter	0.1666	0.2627	0.3608
Homebrew	0.1949	0.3187	0.3993
Pytorch	0.2096	0.3308	0.4616
Rails	0.2358	0.3618	0.4744
Rust	0.2190	0.3158	0.3980
Tensorflow	0.2200	0.3238	0.4158
VScode	0.1870	0.3153	0.4222
max violation	0.1046	0.1220	0.1228

(b) Training and test sets being re-sampled with the stratified technique.

Dataset	false alarm tolerance τ		
	0.2	0.3	0.4
Ansible	0.2065	0.3040	0.4032
wp-Calypso	0.2049	0.3039	0.4116
Elasticsearch	0.1974	0.2989	0.3963
Googleflutter	0.1974	0.2973	0.3934
Homebrew	0.1939	0.2927	0.3963
Pytorch	0.2027	0.2924	0.3931
Rails	0.1964	0.2962	0.4075
Rust	0.2056	0.3034	0.4006
Tensorflow	0.1942	0.2982	0.4018
VScode	0.2071	0.3059	0.4004
max violation	0.0071	0.0059	0.0116

(c) Training and test sets in the chronological order along with a spare validation set.

Dataset	false alarm tolerance τ		
	0.2	0.3	0.4
Ansible	0.1505	0.3337	0.4375
wp-Calypso	0.2052	0.3096	0.4148
Elasticsearch	0.2610	0.3783	0.4837
Googleflutter	0.1072	0.2144	0.2144
Homebrew	0.1475	0.1475	0.4094
Pytorch	0.1945	0.2952	0.4288
Rails	0.2021	0.2938	0.4082
Rust	0.1183	0.3013	0.3509
Tensorflow	0.1868	0.3060	0.3932
VScode	0.1617	0.3097	0.3685
max violation	0.0610	0.0783	0.0837

objective), false alarm (the constraint), and G-mean across data sets is presented in Table III. The reported performance is a single value for the adapted shift threshold approach, and is a mean value across 20 runs for the adapted dynamic loss approach.

Specifically, as the adapted shift threshold approach is a deterministic algorithm that has no randomness, the performance is a single value (not mean). The false alarm failing the given constraint τ is highlighted in red font (gray). As the adapted dynamic loss approach has randomness, we conduct the one sample t-test [32], a statistical test, to investigate whether the false alarm can meet the given constraint τ at the significance level 0.05. The false alarm significantly failing the given constraint is also highlighted in red font (gray).

We can see that the adapted shift threshold approach fails the constraint on 15 data sets out of the total 30; the adapted dynamic loss approach fails the constraint on 11 sets, being slightly better. In terms of the recall, the adapted shift threshold approach can achieve better recall compared with the adapted dynamic loss approach in 23 data sets out of the total 30. Overall, the adapted shift threshold approach performs relatively well in terms of recall but it usually fails the constraint; the adapted dynamic loss approach can obtain meet the constraint better but it cannot achieve good recall.

Therefore, it is concluded that existing adapted ML methods can solve the constraint optimization problem of JIT-SDP to some extent, but there is still large improvement room.

C. How Well Does Our Approach Solve the Constraint Optimization of JIT-SDP?

This section aims for answering RQ2 by comparing our proposed method against existing adapted ML approaches across the investigated data sets. Predictive performance of our approach in terms of recall, false alarm, and G-mean across data sets is reported in the last column part of Table III. The reported performance is the mean value across 20 runs.

We perform the one sample t-test at the significance level 0.05 to statistically justify whether the false alarm of a classifier can meet the given constraint τ. Significant failure in the constraint is highlighted in red font. We perform the Wilcoxon rank sum test [33] at the significance level 0.05 to judge whether our proposed method significantly outperforms or is significantly inferior to the adapted shift threshold approach and the adapted dynamic loss approach, respectively, on each data set. The parentheses associated to our method record this comparison results, for which the first and the second signs correspond to the adapted shifted threshold approach and the adapted dynamic loss approach, respectively. The sign "$+$" means that our approach performs significantly better than its competitor; "$-$" means that our approach performs significantly worse than its competitor; and "$\sim$" means that there is no significant difference between two approaches.

We can see from Table III that our method only fails the constraint on 6 data sets out of the total 30, performing the best among the three competing methods in terms of the constraint of false alarm. In terms of the objective (recall), we can see that our method performs similar to or significantly better than the adapted shifted threshold approach in 17 data sets out of the total 30, and the adapted dynamic loss approach in 24 data sets, showing the superiority of our method in achieving generally good recall over the existing adapted ML approaches. In terms of the overall performance in G-means, we can see that our method performs similar to or significantly better than the adapted shifted threshold approach in 22 data sets out of the total 30, and the adapted dynamic loss approach in 25 data sets, showing the superiority of our method in achieving generally good G-mean over the existing adapted ML approaches. Therefore, it is concluded that our method outperform its competitors with respect to all the three performance aspects, demonstrating the effectiveness of the proposed method.

Nevertheless, the proposed advanced DE algorithm with an adaptive constraint is search-based, so that it has higher computational cost and needs more time to get the prediction model in comparison to the existing adapted ML approaches. The adapted shift threshold method has the lowest computational cost. As the adapted dynamic loss approach need to train models many times with varying loss functions, this approach would have slighted higher computational cost than the adapted shift threshold method.

Therefore, the proposed advanced DE algorithm with an adaptive constraint can achieve generally better performance

TABLE III: Predictive performance with respect to varying constraint tolerance. Significant violation of constraint is highlighted in red font (gray). The sign "+" associated to our approach means that our approach performs significantly better than its competitor; "−" means that our approach is significantly inferior to its competitor; and "∼" means that no significant difference is detected.

(a) The constraint tolerance 0.2.

Dataset	Adapted shift threshold approach			Adapted dynamic loss approach			The proposed DE algorithm with an adaptive constraint		
	Recall	False alarm	G-mean	Recall	False alarm	G-mean	Recall	False alarm	G-mean
Ansible	0.4472	0.1505	0.6164	0.3970	0.1476	0.5816	0.2945 (−, −)	0.1413	0.5028 (−, −)
wp-Calypso	0.6409	0.2052	0.7137	0.5436	0.2110	0.6549	0.6449 (∼, +)	0.2022	0.7172 (∼, +)
Elasticsearch	0.6478	0.2610	0.6919	0.5423	0.2092	0.6548	0.6608 (+, +)	0.2539	0.7021 (+, +)
Googleflutter	0.6923	0.1072	0.7862	0.6154	0.1464	0.7248	0.6192 (−, ∼)	0.1003	0.7458 (−, +)
Homebrew	0.4286	0.1475	0.6044	0.4286	0.1739	0.5950	0.4643 (+, −)	0.1760	0.6168 (∼, ∼)
Pytorch	0.5074	0.1945	0.6393	0.4217	0.1725	0.5907	0.4731 (−, +)	0.1758	0.6244 (−, +)
Rails	0.5360	0.2021	0.6540	0.4011	0.1760	0.5749	0.5459 (+, +)	0.1860	0.6665 (+, +)
Rust	0.4000	0.1183	0.5939	0.6000	0.2094	0.6887	0.4200 (∼, −)	0.1757	0.5854 (∼, −)
Tensorflow	0.4586	0.1868	0.6107	0.3651	0.1860	0.5451	0.4766 (+, +)	0.2077	0.6145 (+, +)
VScode	0.4444	0.1617	0.6104	0.4815	0.1941	0.6229	0.5259 (+, +)	0.1971	0.6496 (+, +)

(b) The constraint tolerance 0.3.

Dataset	Adapted shift threshold approach			Adapted dynamic loss approach			The proposed DE algorithm with an adaptive constraint		
	Recall	False alarm	G-mean	Recall	False alarm	G-mean	Recall	False alarm	G-mean
Ansible	0.6080	0.3337	0.6365	0.6043	0.2656	0.6661	0.5304 (−, −)	0.2393	0.6352 (∼, −)
wp-Calypso	0.7810	0.3096	0.7343	0.6952	0.3079	0.6937	0.7541 (−, +)	0.2914	0.7310 (−, +)
Elasticsearch	0.7783	0.3783	0.6956	0.6294	0.3047	0.6615	0.7531 (−, +)	0.3593	0.6946 (∼, +)
Googleflutter	0.6923	0.2144	0.7375	0.6231	0.2307	0.6922	0.6885 (∼, +)	0.2004	0.7418 (∼, +)
Homebrew	0.4286	0.1475	0.6044	0.7179	0.3169	0.6995	0.7357 (+, ∼)	0.2994	0.7172 (+, +)
Pytorch	0.6556	0.2952	0.6797	0.5752	0.3001	0.6343	0.6325 (−, +)	0.2807	0.6745 (−, +)
Rails	0.6439	0.2938	0.6743	0.5540	0.2767	0.6330	0.6732 (+, +)	0.3006	0.6862 (+, +)
Rust	0.8000	0.3013	0.7477	0.6000	0.3148	0.6412	0.6000 (−, ∼)	0.2762	0.6581 (−, +)
Tensorflow	0.5955	0.3060	0.6429	0.5244	0.2870	0.6114	0.6185 (+, +)	0.2991	0.6583 (+, +)
VScode	0.5926	0.3097	0.6396	0.5556	0.2918	0.6272	0.6556 (+, +)	0.3187	0.6682 (+, +)

(c) The constraint tolerance 0.4.

Dataset	Adapted shift threshold approach			Adapted dynamic loss approach			The proposed DE algorithm with an adaptive constraint		
	Recall	False alarm	G-mean	Recall	False alarm	G-mean	Recall	False alarm	G-mean
Ansible	0.7437	0.4375	0.6468	0.6892	0.3732	0.6572	0.7231 (−, +)	0.3834	0.6675 (+, +)
wp-Calypso	0.8422	0.4148	0.7020	0.8108	0.4022	0.6962	0.8323 (−, +)	0.3966	0.7085 (+, +)
Elasticsearch	0.8103	0.4837	0.6468	0.7624	0.4246	0.6623	0.8050 (−, +)	0.4379	0.6726 (+, +)
Googleflutter	0.6923	0.2144	0.7275	0.8846	0.3666	0.7483	0.7308 (+, −)	0.3167	0.7063 (−, −)
Homebrew	0.7857	0.4094	0.6812	0.8821	0.4127	0.7195	0.8464 (+, −)	0.4102	0.7065 (+, −)
Pytorch	0.7796	0.4288	0.6673	0.6996	0.4151	0.6396	0.7524 (−, +)	0.3981	0.6729 (+, +)
Rails	0.7446	0.4082	0.6638	0.7174	0.4003	0.6559	0.7496 (∼, +)	0.3923	0.6748 (+, +)
Rust	0.8000	0.3509	0.7206	0.6000	0.4224	0.5887	0.7700 (∼, +)	0.3789	0.6900 (−, +)
Tensorflow	0.7357	0.3932	0.6681	0.6629	0.3886	0.6366	0.7223 (−, +)	0.3752	0.6717 (+, +)
VScode	0.6296	0.3685	0.6306	0.6593	0.3920	0.6328	0.6833 (+, ∼)	0.4016	0.6393 (+, ∼)

on the constraint optimization problem (Eq. (1)) in JIT-SDP, deriving better models in terms of recall, constraint, and G-mean. However, the proposed method is of higher computational cost compared with the adapted ML methods, and if practitioners prefer a low cost on computing resources, the adapted dynamic loss approach is recommended.

VII. Conclusion

This paper studies the research problem of pursuing the best detection of positive data under the concern on a given false alarm in the application domain of JIT-SDP, which can be formulated as an constraint optimization problem. We proposed an advance DE algorithm with an adaptive constraint to tackle this research problem, and evaluated it by answering the two research questions as follows.

Answer to RQ1: Applying existing ML methods directly to address the research problem can usually fail the constraint, so we propose to spare a validation set to facilitate the search for the classification threshold of the shift threshold approach and for the regulation term of the dynamic loss approach, leading to the adapted versions of these two methods. Experimental results showed that the adapted ML methods can solve the constraint optimization problem of JIT-SDP to some extent, but there is still large improvement room.

Answer to RQ2: We proposed an advanced DE algorithm with an adaptive constraint, as a third approach, to tackle the constraint optimization problem in JIT-SDP. Experimental results with 10 real-world data sets from the domain of JIT-SDP demonstrated that our proposed method can achieve generally better performance in terms of achieving good detection of positive data and satisfying the given constraint tolerance. However, the proposed method is of higher computational cost compared with the adapted ML methods, and if practitioners prefer a low cost on computing resources, the adapted dynamic loss approach is recommended.

REFERENCES

[1] L. Cai, Y. Fan, M. Yan, and X. Xia, "Just-in-time software defect prediction: Literature review," *Journal of Software*, vol. 30, no. 5, pp. 1288–1307, 2019.

[2] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 492–501.

[3] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[4] M. A. Davenport, R. G. Baraniuk, and C. D. Scott, "Controlling false alarms with support vector machines," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, 2006, pp. 589–592.

[5] J. Broadwater, C. Carmen, and A. Llorens, "False alarm constrained classification," in *4th International Conference and Exhibition on Underwater Acoustic Measurements: Technologies and Results*, 2011, pp. 347–354.

[6] X. Yao and Y. Xu, "Recent advances in evolutionary computation," *Journal of Computer Science and Technology*, vol. 21, no. 1, pp. 1–18, 2006.

[7] S. Kim, E. J. Whitehead, and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.

[8] X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, "Predicting crashing releases of mobile applications," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1–10.

[9] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[10] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[11] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 99–108.

[12] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 5, pp. 412–428, 2018.

[13] G. G. Cabral, L. L. Minku, E. Shihab, and S. Mujahid, "Class imbalance evolution and verification latency in just-in-time software defect prediction," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 666–676.

[14] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 172–181.

[15] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 17–26.

[16] R. E. Wright, *Logistic regression.* American Psychological Association, 1995.

[17] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[18] Y. Cho, J.-H. Kwon, and I.-Y. Ko, "Cross-sub-project just-in-time defect prediction on multi-repo projects," in *6th International Workshop on Quantitative Approaches to Software Quality.*, 2018, pp. 2–9.

[19] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: analytics and risk prediction of software commits," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 966–969.

[20] P. Wang and C. Jin, "The application of genetic algorithm support vector machine in software defect prediction," *Electronic Measurement Technology*, vol. 35, no. 2, p. 4, 2012.

[21] R. Malhotra and A. Khurana, "Analysis of evolutionary algorithms to improve software defect prediction," in *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, 2017, pp. 301–305.

[22] X. Chen, Y. Shen, S. Meng, Z. Cui, X. Ju, and Z. Wang, "Multi-objective optimization based feature selection method for software defect prediction," *Journal of Frontiers of Computer Science and Technology*, vol. 12, no. 9, p. 14, 2018.

[23] X. Yang, H. Yu, G. Fan, and K. Yang, "Dejit: a differential evolution algorithm for effort-aware just-in-time software defect prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 31, no. 03, pp. 289–310, 2021.

[24] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Defect prediction as a multiobjective optimization problem," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 426–459, 2015.

[25] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[26] V. Bewick, L. Cheek, and J. Ball, "Statistics review 13: receiver operating characteristic curves," *Critical care*, vol. 8, no. 6, pp. 1–5, 2004.

[27] S. Kotsiantis, D. Kanellopoulos, P. Pintelas *et al.*, "Handling imbalanced datasets: A review," *GESTS international transactions on computer science and engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[28] W.-L. Loh, "On latin hypercube sampling," *The annals of statistics*, vol. 24, no. 5, pp. 2058–2080, 1996.

[29] D. Dawar and S. A. Ludwig, "Differential evolution with dither and annealed scale factor," in *2014 IEEE Symposium on Differential Evolution (SDE)*, 2014, pp. 1–8.

[30] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, no. 2-4, pp. 311–338, 2000.

[31] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. Mit Press, 2008.

[32] B. Gerald, "A brief review of independent, dependent and one sample t-test," *International journal of applied mathematics and theoretical physics*, vol. 4, no. 2, pp. 50–54, 2018.

[33] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*, 1992, pp. 196–202.