A Distance-Based Dynamic Random Testing Strategy for Natural Language Processing DNN Models

Yuechen Li, Hanyu Pei, Linzhi Huang, and Beibei Yin*
School of Automation Science and Electrical Engineering, Beihang University, Beijing, China
liyuechen@buaa.edu.cn, peihanyu@buaa.edu.cn, huanglinzhi@buaa.edu.cn, yinbeibei@buaa.edu.cn
*corresponding author

***Abstract*—Deep neural networks (DNNs) have achieved tremendous development while they may encounter with incorrect behaviors and result in economic losses. Identifying the most represented data become critical for revealing incorrect behaviours and improving the quality DNN-driven systems. Various testing strategies for DNNs have been proposed. However, DNN testing is still at early stage and existing strategies might not sufficiently effective. Dynamic random testing (DRT) strategy uses the feedback mechanism to guide the test case selection, which has been proved to be effective in fault detection. However, its efficacy for Natural Language Processing (NLP) DNN models has not been thoroughly studied. In this paper, a Distance-based DRT with prioritization (D-DRT-P) is proposed, which combines the priority information and distance information into DRT to guide the selection of test cases and testing profile adjustment. Empirical studies demonstrate that D-DRT-P can improve the fault detecting effectiveness than other test prioritization strategies in most cases.**

Keywords- Dynamic random testing, Test prioritization, Deep neural networks, Natural language processing, Software cybernetics.

I. INTRODUCTION

The technology of Deep Learning (DL), based on Deep Neural Network (DNN), has achieved great process and widely deployed in various domains, including medical image processing, speech recognition, automatic driving, natural language processing (NLP) etc. [1]. NLP is an essential branch of DL and with its rapid development, NLP has made breakthroughs at the application and technology level. Text classification is a classical problem in NLP which aims to assign labels or tags to textual units and includes rich and diverse data from different sources [2]. However, DL suffers from software faults that may cause dangerous accidents. Especially for NLP based on DL, natural language usually contains richer information than image data, and has the characteristics of subjectivity, ambiguity and irregularity, which prompts that extracting insights from text can be challenging and time-consuming.

The technology of software testing is a vital means to ensure software quality and reliability. DNN testing strategies aim at identifying and selecting the most representative data for improving the quality of DNN-driven systems. Inspired by code coverage in traditional software testing, researchers have proposed several neuron coverage criteria to measure the adequacy of DNN testing, such as DeepXplore [3], DeepGauge [4] and DeepCT [5]. They utilize the distribution of neuron activation values and the relationship between adjacent neuron layers. Intuitively, a test suite with higher neuron coverage seems to reveal more faults in DNN-based systems. Thus, it is rational to select a test suite with higher neuron coverages. However, some researchers point out that the resource cost of collecting the activation state of neurons could be too expensive [6]. And some neurons covered criteria can easily achieve maximum coverage, which could limit the effectiveness of neurons coverage as a test case selection strategy [7].

On the other hand, several test case prioritization strategies are also proposed. This technology sorts test cases with specific rules. For example, DeepGini [8] is proposed by Feng et al., which takes the use of the Gini coefficient to measure the likelihood of test case being misclassified. Likelihood-based Surprise Adequacy (LSA) is proposed by Kim et al. [9] to measure the surprise of an input as the difference in DNN system's behavior between the input and the training data. Generally, test case prioritization strategies rank the test cases according to their evaluation scores, and then select test cases in the order from high scores to low scores (or vice versa). Studies have shown that prioritization strategies are effective in DL-based software testing since a good prioritization technique can prioritize test cases that are more likely to reveal errors in classification.

However, one of disadvantages of test prioritization strategies is that the order of test case selection might be overconfident. Usually, test cases with higher scores are more likely to be misclassified by DNN and trigger the failure. Nevertheless, it's a probabilistic rather than an inevitable event, that is, test cases with high scores may not be identified as faults, and vice versa for those with low scores. Especially as the scores drop, the effectiveness of test prioritization guiding the selection of test cases reduces gradually. NLP testing is a relatively emerging field that lack of well-labeled datasets. The accuracy of DNN models might be undesirable in text classification problems, which may affect the effectiveness of testing strategies that merely explore prioritization. Besides, the order of the test prioritization represented by DeepGini etc. is relatively fixed, so adding random factors that obey the probability distribution in the testing process is a necessary attempt to further improve the fault detection ability.

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00089

Dynamic random testing (DRT) strategy utilizes the testing results as feedback information to guide the selection of test cases, which has been proved to be effective in fault detection process [10][11]. More specifically, suppose that the test cases are divided into k subdomains $\{C_1, C_2, \cdots, C_k\}$, and each C_i is allocated with selection probability p_i. DRT selects a subdomain according to the testing profile $\{p_1, p_2, \cdots, p_k\}$, then selects a test case from the chosen subdomain in accordance with uniform probability distribution. If a failure is triggered by a test case in the subdomain C_i the selection probability of C_i will increase from p_i to $p_i + \varepsilon$, and the selection probabilities of other subdomains will decrease $\varepsilon/(k-1)$; otherwise, the selection probability will decrease to $p_i - \delta$, and the selection probabilities of other subdomains will increase $\delta/(k-1)$.

DRT changes the testing profile based on whether faults are detected, which guarantees failure-causing inputs being selected faster and thus improves the fault detection effectiveness. However, if the DRT applied in the traditional software is directly introduced to the DL software, the results from the output layer of DNN will not fully utilized. Thus, taking the advantages of the feedback mechanism of DRT and test prioritization, the effectiveness of DNN testing can be further improved. Therefore, we propose a DNN-DRT strategy, namely Distance-based DRT with test prioritization (D-DRT-P) strategy in this paper, which is referred to the principle of DRT and geared towards DL-based software. Based on the above methods, D-DRT-P, considered as a theoretically feasible optimization strategy, uses the feature of test cases for classification and selection of test cases, and adjusts the testing profile based on distance information among subdomains, along with the results that whether text inputs are misclassified. To validate the fault detection effectiveness of DNN-DRT strategy, we conduct experiments for D-DRT-P and baseline methods with three well-designed DNN models and public datasets for text classification. The experimental results demonstrate that D-DRT-P strategy can achieve better performance in most cases.

The contribution of this paper is as follows:

1) We propose a DNN-DRT strategy, namely D-DRT-P in this paper, which takes advantage of test prioritization method and feedback mechanism of DRT. This can improve the fault detection effectiveness on DL-based NLP software.

2) We introduce a low-dimensional feature vector in regard to the fault detection of NLP software, which can gather faults by clustering and dividing test cases to detect faults faster in the selected subdomain.

3) We conduct a series of experiments to investigate the performance of DNN-DRT strategy with baseline testing strategies. The results show that DNN-DRT can achieve better fault detection performance than other test prioritization methods.

The remainder of this paper is organized as follows: The background is introduced in section II. The DNN-DRT strategy is presented in section III. The experimental setup is described in Section IV. The experimental results are analyzed in Section V. Threats to validity are summarized in Section VI. Related works on testing strategies are presented in Section VII. Conclusions and future works are summarized in Section VIII.

II. BACKGROUND

A. The architecture of DNN testing

A DNN can be explained as an iterative function chain $F_1, F_2, \cdots, F_D$ mapping from input data vector $\mathbf{x}$ to the output vector $\mathbf{y}$. Then, $DNN(\cdot)$ is utilized to stand for the whole DNN in the respective of black-box model, and

$$\mathbf{y} = DNN(\mathbf{x}) = F_D \ \ F_{D-1} \ \cdots \ F_2 \ \ F_1(\mathbf{x}) \tag{1}$$

where D represents the depth of the DNN, which usually refers to the sum of hidden layers and output layers. The function $F_i(\cdot), i \in \{1,2,\cdots,D-1\}$ represents the relation of the hidden layers in the DNN, and $F_D(\cdot)$ represents the relationship with the output layer.

Each hidden layer $F_i(\cdot)$ maps the forward input to a vector, and each element in the vector is an independent, parallel neuron. For a DNN with a fixed width W, the vector composed with neurons in the i-th hidden layer can be represented by $\mathbf{h}_i \in \mathbb{R}^W, i \in \{1,2,\cdots,D-1\}$. If all the hidden layers are considered, then the hidden layers form a matrix $\mathbf{H} \in \mathbb{R}^{W\times(D-1)}$ containing the values of each neuron in the hidden layers. Taking a test case t as the input of DNN, the hidden layer value matrix $\mathbf{H}_t$ can be represented as

$$\mathbf{H}_t = [\mathbf{h}_{t,1}, \mathbf{h}_{t,2}, \cdots, \mathbf{h}_{t,(D-1)}] \in \mathbb{R}^{W\times(D-1)}$$

If we take the text classification problem of m classes as an example to analyze the entire DNN processing, any test case $t \in T$ will be transformed into a high-dimensional semantic vector $\mathbf{x}_t$ as the input of DNN model $DNN(\cdot)$, and then output a m-dimensional vector $\mathbf{y}_t$ according to the categories,

$$\mathbf{y}_t = \left[y_{t,1}, y_{t,2}, \cdots, y_{t,m} \right]^{\mathrm{T}} \in \mathbb{R}^m$$

The goal of a constructed DNN is to approximate the output result $\mathbf{y}_t$ to the theoretical classifier $\mathbf{y}_t^*$, that is, by continuously adjusting the weights in the internal neurons of DNN, so that $\forall i \in \{1,2,\cdots,m\}: y_{t,i} \approx y_{t,i}^*$.

In general, researchers expect a normalized output. The function $softmax$ is to map each element value of the output vector to the range of $(0,1)$ according to its relative proportion, in which the probability vector $\mathbf{Pr}_t$ can represent the probability of each classification,

$$\mathbf{Pr}_t = \left[p_{t,1}, p_{t,2}, \cdots, p_{t,m} \right]^{\mathrm{T}} \in (0,1)^m, \| \mathbf{Pr}_t \|_1 = 1$$

where the mathematical expression of $softmax$ is

$$p_{t,i} = softmax\ \ \mathbf{y}_t \ \ = \frac{e^{y_{t,i}}}{\sum_{j=1}^{m} e^{y_{t,j}}} \tag{2}$$

Since the network output elements are all finite numbers $(-\infty < y_{t,i} < +\infty)$, and only $\lim_{y_{t,i}\to-\infty} p_{t,i} = 0$. Therefore, each

element of the probability vector $\mathbf{Pr}_t$ cannot reach the boundary value in the respective of probability, that is, $p_{t,i} \in [0,1]\backslash\{0,1\} = (0,1)$. Finally, the output of test case t is the classification result based on $\mathbf{Pr}_t$, which is shown as follow,

$$class(t) = \arg\max_{i=\{1,2,\cdots,m\}} p_{t,i} \tag{3}$$

Taking the simple DNN shown in Figure 1 for an example, it's aimed to classify into the reasonable class of the input text among three candidates. If test case t, the text *What a clam shame.* is input in the DNN, it will be changed into a high-dimensional semantic vector as input layer and a 3-dimensional probability vector will be output at last. According to $\mathbf{Pr}_t = [0.02,0.04,0.94]^{\mathrm{T}}$, $class(t) = 3$. In another word, the text is of neither class which excludes hate and offensive factors.

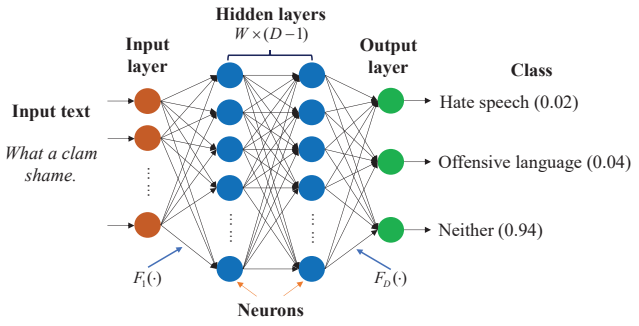

Figure 1 A sample of NLP DNN model

B. Test case prioritization

In the field of software testing, test case prioritization is a classic issue raised by Rothermel et al. [12] as follows. Given a test set $T = \{t_1, t_2, \cdots, t_N\}$, its generating sequence is

$$\{t_{n_k}, k \in \{1,2,\cdots,N\}, t_{n_k} \in T\}$$

When testing by prioritization, the order of test follows above sequence. Let $f: T \rightarrow \mathbb{R}$ as the mapping function from the given test set to real domain and it gives a kind of standard to guide the formation of a suitable sequence. A prioritization sequence means that the top test cases are more important than the bottom ones in some ways, so

$$\forall i \in \{1,2,\cdots,N-1\} : f(t_{n_i}) > f(t_{n_{i+1}})$$

In this paper, Gini impurity and information entropy are introduced to measure the likelihood of test cases to be classified wrongly. Above two metrices can connect test cases and their equivalent numerical values, and then expose test cases that are prone to misclassification in the test set.

1) *Gini impurity*

Given a test case t and a DNN that outputs the vector $\mathbf{Pr}_t = [p_{t,1}, p_{t,2}, \cdots, p_{t,m}]^{\mathrm{T}}$, Gini impurity $\xi(t)$ is defined to measure the likelihood of t being classified differently in two randomized trials,

$$\xi(t) = 1 - \sum_{i=1}^{m} p_{t,i}^2 \tag{4}$$

where $\xi(t)$ ranges in $\left(0, \frac{m-1}{m}\right]$.

2) *Information entropy*

Information entropy can be regarded as the uncertainty of a random variable, which is similar to Gini impurity in the sense of uncertainty. A random variable with higher information entropy means higher uncertainty. For a test case t and corresponding probability vector $\mathbf{Pr}_t$, the information entropy $H(t)$ is calculated as follows,

$$H(t) = -\sum_{i=1}^{m} p_{t,i} \log_2 p_{t,i} \tag{5}$$

where $H(t)$ ranges in $(0, \log_2 m]$. The unit of information entropy defined is binary bit, and the formular means the mathematical expectation of the uncertainty defined by logarithm.

In order to study the effect of combining several metrics, weighted average method can be added and generate a composite metric. Let the metric as $\tau(t; w_1, w_2, \cdots, w_{n-1})$ composed with $n(n \geq 2)$ kinds of mapping functions $f_1(t), f_2(t), \cdots, f_n(t)$, so

$$\tau(t; w_1, w_2, \cdots, w_{n-1}) = \sum_{i=1}^{n-1} \left[w_i f_i^*(t) \right] + \left(1 - \sum_{i=1}^{n-1} w_i \right) f_n^*(t) \tag{6}$$

where $w_1, w_2, \cdots, w_n \in [0,1]$ are adjustable weighted values and let $\sum w_i = 1$. And $f_1^*(t), f_2^*(t), \cdots, f_n^*(t)$ are linear norm-alized function of $f_1(t), f_2(t), \cdots, f_n(t)$, where

$$f_i^*(t) = \frac{f_i(t) - \min_{t \in T} f_i(t)}{\max_{t \in T} f_i(t) - \min_{t \in T} f_i(t)} \quad i \in \{1,2,\cdots,n\} \tag{7}$$

In this paper, assume that $f_1(t) = \xi(t)$, $f_2(t) = H(t)$ and $\tau(t; w_1, w_2, \cdots, w_{n-1}) = \tau(t; w)$. Since Gini impurity and information entropy are proposed based on probability theory and information theory respectively, generating a composite metric using an adjustable parameter w has certain research significance for analyzing the relation between above two.

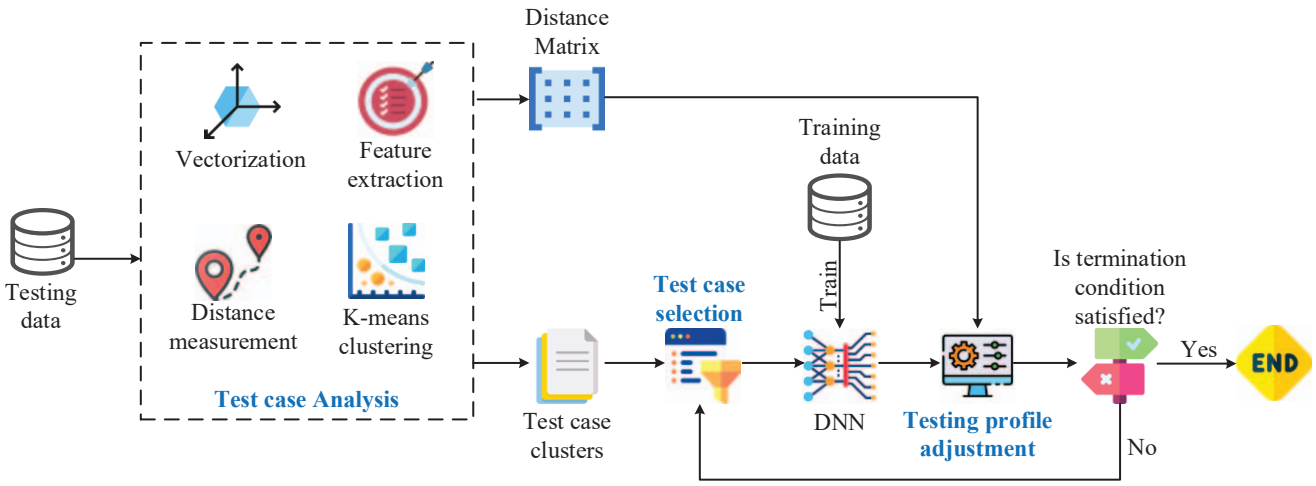

Figure 2 The framework of D-DRT-P strategy

III. METHOD

The proposed framework of D-DRT-P is shown in Figure 2. The testing process is as follows. First, the features of test cases are extracted and then transferred to vectors referred to DNN. Then they are classified to several disjoint subdomains by using k-means clustering method. Second, the distance matrix is calculated based on the classification results. Third, the test case is selected according to the testing profile. Fourth, the testing profile is adjusted based on the testing results and distance information among subdomains and the testing process can be considered as the mapping $Test: T \to \{0,1\}$,

$$Test(t) = \begin{cases} 0, & t \text{ is classified correctly by DNN} \\ 1, & t \text{ is misclassified by DNN} \end{cases} \quad (8)$$

The testing process stops when the terminate condition is satisfied.

A. Feature vectors of test cases

For the part of test case clustering, the reference of clustering is defined as s-dimensional feature vector $\mathbf{z}_t$,

$$\mathbf{z}_t = [z_1(t), z_2(t), \cdots, z_s(t)]^{\mathrm{T}} \in \mathbb{R}^s$$

where $z_i(t)(i \in \{1,2,\cdots,s\})$ is each feature of $\mathbf{z}_t$. Mentioned the mapping function $f: T \to \mathbb{R}$ for test prioritization, $z_i(t)$ and $f_i(t)$ have similarities in quantifying test case t.

For text data, the data type input to DNN is high-dimensional vector, but after some preliminary experiments, it is found that word vector is not suitable as the feature vector for clustering in dynamic random testing because excessive dimensions will dilute the role of more critical elements theoretically. Intuitively, each element in the word vector is not actually related to the test case being easily misclassified.

Therefore, to guide fault detection and expose specific test cases, $\mathbf{z}_t$ can be composed with s_1 kinds of quantified uncertainty information and s_2 kinds of data inherent properties, where $s_1 + s_2 = s$. For the first part, Gini impurity $\xi(t)$ and information entropy $H(t)$ are introduced in the paper and for the other part, $class(t)$ of test case t shows its property output from DNN. Therefore, $\mathbf{z}_t$ can be combined with above three features in the following experiments, where $\mathbf{z}_t = [w_1\xi(t), w_2H(t), w_3class(t)]^{\mathrm{T}}$ and w_1, w_2, w_3 are the weights of each feature.

B. Test Case Classification

1) Clustering of Test Cases

K-means clustering method is efficient and scalable to partition the test cases of large data sets. It first randomly selects k test cases, each of them is set to be the initial center of each cluster. The Euclidean distance between remaining test cases and the centers of clusters are calculated, and test cases are assigned to their closest cluster. Suppose $\{C_1, C_2, \cdots, C_k\}$ represent k clusters, and there are c_i test cases in C_i $(i \in \{1,2,\cdots,k\})$. And $C_i = \{t_{i,1}, t_{i,2}, \cdots, t_{i,j}, \cdots, t_{i,c_i}\}$, where $j \in \{1,2,\cdots,c_i\}$. Then the coordinate of each cluster center $(a_{i,1}, a_{i,2}, \cdots, a_{i,s})$ is updated according to the s-dimensional feature vector $\mathbf{z}_t$ of each test case $t \in C_i$ and it equals to the mean value, which is calculated as

$$a_{i,j} = \sum_{t \in C_i} \frac{z_{t,j}}{c_i} \quad i \in \{1,2,\cdots,k\}, j \in \{1,2,\cdots,s\}$$

For the next round, the Euclidean distance between each test case and the updated center of each cluster is calculated, and the test cases are assigned to the closest clusters individually. This process is repeated until the test cases in each cluster no longer change, or the sum of the squared error (SSE) converges. The SSE of k-means clustering method is calculated with

$$SSE = \sum_{i=1}^{k} \sum_{t \in C_i} [d(t, C_i)]^2$$

where $d(t, C_i)$ represents the distance between test case t of C_i and the mean value of C_i. By above adjusting rounds, when test cases in each cluster remain unchanged compared with the previous round, the current round can be considered as the final state. It should be noted that the clustering results might not be desirable since the selection of initial test cases may have a great effect on the clustering process. Therefore, we

usually conduct k-means clustering method several times, and the result with the minimum SSE is selected as the final classification.

2) *Distance Matrix Generation*

The distance matrix **D** is introduced by Pei et al when proposing Distance-based DRT (D-DRT) [13]. In D-DRT, the adjustment of the testing profile is based on both the testing results and the distance information among subdomains. Assuming that there are $k(k \geq 2)$ testing profiles $\{C_1, C_2, \cdots, C_k\}$ in the test suite and $h(h \in \mathbb{N}^*)$ dimensions in the feature vector of the test case, for the i-th testing profile C_i, the coordinates of the cluster center can be represented as $(a_{i,1}, a_{i,2}, \cdots, a_{i,h})$. Then, the Euclidean distance between any subdomains C_i and C_j is

$$d(C_i, C_j) = \sqrt{\sum_{l=1}^{h} (a_{i,l} - a_{j,l})^2} \quad i, j \in \{1,2,\cdots,k\} \wedge i \neq j \tag{9}$$

From this, the distance matrix is **D** defined as

$$\mathbf{D} \triangleq \begin{bmatrix} 0 & d\ C_1,C_2 & \cdots & d\ C_1,C_k \\ d\ C_2,C_1 & 0 & \cdots & d\ C_2,C_k \\ \vdots & \vdots & \ddots & \vdots \\ d\ C_k,C_1 & d\ C_k,C_2 & \cdots & 0 \end{bmatrix}$$

Since $d(C_i, C_j) = d(C_j, C_i)$, the distance matrix is a real symmetric matrix $\mathbf{D} = \mathbf{D}^{\mathsf{T}}$. During the profile adjustment process, the selected test case will be removed from the testing profile when detected. Then, all the testing profiles will become empty in the end.

C. Test case selection

At first$(n = 0)$, adjusting parameters (ε, δ) are set and test set is divided into $k(k \in \mathbb{N}^* \wedge k \geq 2)$ clustering subdomains $C_1, C_2, \cdots, C_k$ as testing profiles. Each subdomain has $c_1(0), c_2(0), \cdots, c_k(0)$ test cases respectively and the initial selection probability distribution of each clustering is $\{P_1(0), P_2(0), \cdots, P_k(0)\}$. Assuming that the sum of selected features is s, the θ -th feature value $z_\theta(t)$ $(\theta = 1,2,\cdots,s)$ of $\mathbf{z}_t$ can be considered as guideline of test prioritization.

During the testing process $(n > 0)$, based on the probability distribution $\{P_1(n), P_2(n), \cdots, P_k(n)\}$ at the n-th step, select a subdomain C_l, where $l \in \{1,2,\cdots,k\}$. Next, according to the probability distribution $\{p_1, p_2, \cdots, p_{c_l(n)}\}$ of test cases of C_l, select the u -th test case $t_u \in C_l (u \in \{1,2,\cdots,c_l(n)\})$. Then,

1) for original DRT, the selection probability of test cases is

$$p_i = \frac{1}{c_l(n)},\ i = 1,2,\cdots,c_l(n) \tag{10}$$

where the selection mechanism of t_u is chosen based on uniform distribution. Above strategies for selection are similar to traditional DRT and D-DRT without internal information of software execution.

2) for D-DRT-P, the selection mechanism is related to the feature value $z_\theta(t)$,

$$p_i = \frac{z_\theta(t)}{\sum_{t \in C_l} z_\theta(t)},\ i = 1,2,\cdots,c_l(n) \tag{11}$$

To reduce the randomness in the testing process, the process is to directly select the test case with the largest probability value, where u satisfies

$$u = \underset{i \in \{1,2,\cdots,c_l(n)\}}{\arg\max}\ p_i = \underset{i \in \{1,2,\cdots,c_l(n)\}}{\arg\max}\ z_\theta(t_i) \tag{12}$$

D. Testing profile adjustment

When t_u is input into DNN, and the result(pass/fail) of classification is recorded, so as to judge whether the fault is detected. If $t_u \in C_l$ is misclassified, increase the probability of C_l being selected and update the selection probability of the testing profile from the n-th to the $(n+1)$-th step,

$$P_l(n+1) = \begin{cases} P_l(n) + \varepsilon, & P_l(n) \leq 1 - \varepsilon \\ 1, & P_l(n) > 1 - \varepsilon \end{cases} \tag{13}$$

$$P_j(n+1) = \begin{cases} P_j(n) - \varepsilon_j, & P_l(n) \leq 1 - \varepsilon \\ 0, & P_l(n) > 1 - \varepsilon \end{cases}, j \neq l \tag{14}$$

If $t_u \in C_l$ is not misclassified, reduce the probability of C_l, and update the selection probability as well,

$$P_l(n+1) = \begin{cases} P_l(n) - \delta, & P_l(n) \geq \delta \\ 0, & P_l(n) < \delta \end{cases} \tag{15}$$

$$P_j(n+1) = \begin{cases} P_j(n) + \delta_j, & P_l(n) \geq \delta \\ \frac{\delta_j}{\delta}, & P_l(n) < \delta \end{cases}, j \neq l \tag{16}$$

Noticed that the parameters (ε, δ) and $(\varepsilon_j, \delta_j)$ determine the degree of probability change, the adjustment parameters $(\varepsilon_j, \delta_j)$ of the $C_j (j \neq l \wedge j = 1,2,\cdots,k \wedge k \geq 2)$ de-pend on the following situations:

(1) for original DRT, the probability changes of the remaining subdomains C_j keep consistent,

$$\begin{aligned} \varepsilon_j &= \frac{\varepsilon}{k-1} \\ \delta_j &= \frac{\delta}{k-1} \end{aligned} \tag{17}$$

(2) for D-DRT-P, the probability changes of the remaining subdomains C_j are related to the distance between C_l,

$$\begin{aligned} \varepsilon_j &= \frac{d(C_j, C_l)}{\sum_{i \in \{1,2,\cdots,k\} \setminus \{l\}} d\ C_i,C_l} \cdot \varepsilon \\ \delta_j &= \frac{d(C_j, C_l)}{\sum_{i \in \{1,2,\cdots,k\} \setminus \{l\}} d\ C_i,C_l} \cdot \delta \end{aligned} \tag{18}$$

After testing current t_u, remove it from the selected subdomain. If the subdomain remains no test cases, its probability is evenly distributed to other non-empty subdomains.

E. Algorithm

The algorithm of D-DRT-P strategy is as follows.

Algorithm 1: D-DRT-P

Input: T: the test set; k: the number of subdomains; $\mathcal{X}$: the set of feature vectors of test cases; θ: characteristic of test case; $\mathcal{M}$: DNN model under test; ε, δ: adjusting parameters; $label$: the set of ground labels.

Output: $results$: the testing results.

1 $(Class, CC) \leftarrow ClusterMeans(k, \mathcal{X})$;
2 Classify T into subdomains $\{C_1, C_2, \cdots, C_k\}$;
3 Set initial testing profile $\{P_1, P_2, \cdots, P_k\}$;
4 $\mathbf{D} \leftarrow DistanceCalculation(CC, k)$;
5 $results \leftarrow \varnothing$
6 **while** the testing stop criteria is not satisfied **do**
7 Select a subdomain $C_l \neq \varnothing$ according to $\{P_1, P_2, \cdots, P_k\}$;
8 Select a test case $t_u \in C_l$ with highest prioritization;
9 $(\varepsilon_j, \delta_j) \leftarrow Parameters(\varepsilon, \delta, \boldsymbol{D})$;
10 $prediction_u \leftarrow DNN(\mathcal{M}, t_u)$
11 **if** $prediction_u \neq label_u$ **then**
12 $\{P_1, P_2, \cdots, P_k\} \leftarrow ProfileAdjustment1(k, \varepsilon_j, \delta_j)$;
13 $results \leftarrow results \cup \{(t_u, 1)\}$;
14 **else**
15 $\{P_1, P_2, \cdots, P_k\} \leftarrow ProfileAdjustment2(k, \varepsilon_j, \delta_j)$;
16 $results \leftarrow results \cup \{(t_u, 0)\}$;
17 **end**
18 $C_l \leftarrow C_l \backslash \{t_u\}$
19 **end**

Remarks:

The algorithm of D-DRT-P aims at detecting faults of input text test set T, and finally judging whether the test case is a fault. All the test results can be recorded as the set $results = \{(t, Test(t)) | t \in T\}$. $ClusterMeans()$ in line 1 shows clustering process by k-means according to sum of clustering k and the set of feature vector $\mathcal{X} = \{\boldsymbol{z}_t | t \in T\}$ and output the classes of clustering *Class* as well as the clustering centers *CC*. Then, line 2-5 initialize testing profile and distance matrix to prepare for testing processing. Noticed that compared to traditional D-DRT with initial probability following uniform distribution, DNN-DRT can more clearly assign weights with the feature value $z_\theta(t)$, such as

$$P_i(0) = \frac{\sum_{t \in C_i} \left[z_\theta(t) \,/\, c_i \right]}{\sum_{j=1}^{k} \sum_{t \in C_j} \left[z_\theta(t) \,/\, c_j \right]} \tag{19}$$

which means that the probability may be determined as the proportion of the feature value density $\sum_{t \in C_i} [z_\theta(t)/c_i]$ in each testing profile C_i. Next, line 6-23 are the finite feedback process of fault detection and profile adjustment in which test case t_u is selected referred to prioritization of uncertainty and the profile is changed by $ProfileAdjustment1()$ or $ProfileAdjustment2()$ according to the detection result of t_u whether the ground value $label_u$ equals the output class $prediction_u$. The entire loop structure ends after reaching the stop criteria.

In a word, D-DRT-P continuously adjusts the testing profiles by real-time feedback to find faults earlier. However, the number of subdomains k affects the quality of defined testing profiles, and (ε, δ) impacts the sensitivity of profile adjustment. Therefore, adopting an appropriate group of parameters (k, ε, δ) is critical to the effectiveness of the algorithm.

IV. Experimental Setup

Some experimental studies are conducted in this paper to evaluate the performance of the proposed D-DRT-P strategy. We aim to answer the following questions.

RQ1：How do DNN-DRT strategies perform in terms of fault detection effectiveness compared with other test case prioritization strategies?

RQ2: How does the parameter tuple (k, ε, δ) have effect on the performance of DNN-DRT?

RQ3：How do DNN-DRT strategies perform under different datasets?

A. Subjects

The DNN-based software is constructed by DNN models and it comes to be done after model training. The selected DNN models in this paper are TextCNN, TextRNN and Bert. There is introduction of above well-known models:

1) TextCNN: It was proposed by Yoon Kim in 2014 [14], and utilizes Convolutional Neural Network (CNN) to extract key information similar to n-grams in sentences. The model consists of the following process: input the preprocessed text data into the input layer, then perform text feature extraction in the embedding layer, and then obtain filtered multiple feature maps in the convolutional layer. It is further reduced in the transformation layer, and finally the probability distribution of each category in the classification task will be obtained in the softmax layer.

2) TextRNN: It was proposed by Liu et al. in 2016 in order to solve the problem that long sequence information cannot be modeled in CNN and the hyperparameter adjustment of filter size is too cumbersome [15]. Compared with CNN, Recurrent Neural Network (RNN) sacrifices running speed to capture long-range dependencies in the sequence

3) Bert: It's short for Bidirectional Encoder Representations from Transformers and was represented by Devlin et al. [16] in 2018 by Transformers' bidirectional encoder. The obvious advantage of Bert is that the pre-trained model can be fine-tuned with an additional output layer to create state-of-the-art models for various tasks without requiring extensive modifications to the task-specific architecture.

In our experiments, three kinds of datasets in the NLP domain for text classification, are used to train, verify and test models. Table 1 shows basic information of selected open-source datasets and fault distribution after testing each DNN. As to the part of dataset, *Hate speech and offensive language* (short for *Tweets*) is Davidson et al.'s collection of tweets containing terms from the Hatebase.org dictionary, and samples are labeled as hate speech, offensive speech and non-offensive speech [17]. *Topic labeled news dataset* (short for

News) is provided by NewsCatcher team and 108,774 news articles are labelled with 8 topics such as business, entertainment, health, nation, science, sports, technology and world [18]. *IMDB movie reviews* (short for *Reviews*) is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets and provides 50k movie reviews labelled with positive or negative ones [19]. With regard to the fault distribution, the model accuracy varies with datasets and DNNs, which further reveals the necessity of introducing and analyzing multiple datasets and DNNs.

Table 1 Datasets and DNN models

Dataset	# of classes	DNN models	# of faults	# of training data	# of test cases
Tweets	3	TextCNN	593	24,783	12,007
		TextRNN	264		
		Bert	1,533		
News	2	TextCNN	536	76,142	2,000
		TextRNN	487		
		Bert	384		
Reviews	8	TextCNN	2,154	50,000	7,500
		TextRNN	2,093		
		Bert	1,735		

For above datasets, different methods are used to divide the data. For example, the size of tweet dataset is relatively small, so the training set can be larger to ensure higher accuracy of DNN model. The news dataset and movie review dataset have large-scale samples, of which 70% of data are assigned to the training set and the 15% are divided into the validation set, but the former dataset only samples a small amount of data as test set, and the latter take the remaining 15% as test set. The above data segmentation method shows the diversity of the dataset, and then reflect the general effectiveness of the proposed strategy in subsequent experiments.

B. Variables and Measures

1) *Independent variable*

The independent variable of the experiment is the testing strategy. To evaluate the performance of proposed strategies, RT, three types of test prioritization strategies Gini, Entropy, GE-0.5, and four types of DNN-DRT strategies DRT-R, DRT-P, D-DRT-R, D-DRT-P are included in the following experiments.

RT: RT selects and executes test cases from the entire test suite randomly.

Gini: Gini selects and executes test cases according to the order of $\xi(t)$ decline.

Entropy: Entropy selects and executes test cases according to the order of $H(t)$ decline.

GE-0.5: **G**ini-**E**ntropy with w=**0.5** (GE-0.5) selects and executes test cases according to the order of $\tau(t; 0.5)$ decline where $f_1(t) = \xi(t)$, $f_2(t) = H(t)$.

In addition, four types of DNN-DRT strategies are explored on the basis of previous research, considering whether to apply prioritization and distance-based information.

DRT-R: **D**ynamic **R**andom **T**esting with **R**andom selection **(**DRT-R) selects and executes test cases in the chosen testing profiles randomly.

DRT-P: **D**ynamic **R**andom **T**esting with test **P**rioritization **(**DRT-P) selects and executes test cases in the chosen testing profiles randomly.

D-DRT-R: **D**istance-based **D**ynamic **R**andom **T**esting with **R**andom selection (D-DRT-R) selects and executes test cases in the chosen testing profiles according to the prioritization of $z_\theta(t)$ decline.

D-DRT-P: **D**istance-based **D**ynamic **R**andom **T**esting with test **P**rioritization (D-DRT-P) selects and executes test cases in the chosen testing profiles according to the prioritization of $z_\theta(t)$ decline.

2) *Dependent variables*

The dependent variable for RQ1 to RQ3 is the metric for characterizing the fault detection effectiveness of the candidate strategies. Since there are faults misclassified by DNN in each version of software program and the following experiments intend to detect all seeded faults in a round of testing faster. APFD (Average Percentage of Faults Detection) is used in this study to evaluate the performance and measures the prioritization effectiveness in terms of the rate of faults detection of a test set [20]. Let the test set as T with N test cases and $N_F(N_F \in \{1,2,\cdots,N\})$ faults, and $o_i(o_i \in \{1,2,\cdots,N_F\})$ as the order of the i-th fault to be detected in the testing processing. Then *APFD* can be defined as

$$APFD = 1 - \frac{\sum_{i=1}^{N_F} o_i}{N_F \cdot N} + \frac{1}{2N} \tag{20}$$

where the larger the value of *APFD*, the faster the fault detection of the selected order. It is easy to prove that the range of $APFD$ is $R_{APFD} = \left[\frac{N_F}{2N}, 1 - \frac{N_F}{2N}\right]$, that is, its value relates to the defect proportion N_F/N. Therefore, the evaluation of the performance for testing strategies also needs to consider the characteristics of the test set and the original dataset itself. In the follow-up experiments, in order to eliminate the intuitive influence of fault distribution, it is advisable to linearly normalize $APFD$ as $APFD^*$ so that $R_{APFD^*} = [0,1]$.

Table 2 Average fault detection effectiveness for strategies

Fault Detect ($APFD^*$)		RT	Test Prioritization			DNN-DRT			
Dataset	Model		Gini	Entropy	GE-0.5	DRT-R	DRT-P	D-DRT-R	D-DRT-P
Tweets	TextCNN	0.4989	0.8978	0.8942	0.8962	0.8708	**0.9099**	0.8705	0.9080
	TextRNN	0.5019	0.9220	0.9214	0.9219	0.8316	**0.9252**	0.8307	0.9231
	Bert	0.5001	0.8792	0.8759	0.8777	0.8436	0.8796	0.8438	**0.8800**
Reviews	TextCNN	0.4988	0.8109	0.8109	0.8109	0.5426	0.8121	0.5428	**0.8122**
	TextRNN	0.5000	**0.8448**	**0.8448**	**0.8448**	0.5298	**0.8448**	0.5299	0.8447
	Bert	0.4999	0.8697	0.8697	0.8697	0.5318	**0.8806**	0.5318	0.8805
News	TextCNN	0.4976	0.8219	0.8153	0.8191	0.7621	**0.8225**	0.7621	**0.8225**
	TextRNN	0.5005	0.8124	0.8120	0.8128	0.7546	**0.8129**	0.7546	**0.8129**
	Bert	0.4998	0.8334	0.8308	0.8324	0.7445	**0.8335**	0.7445	0.8334

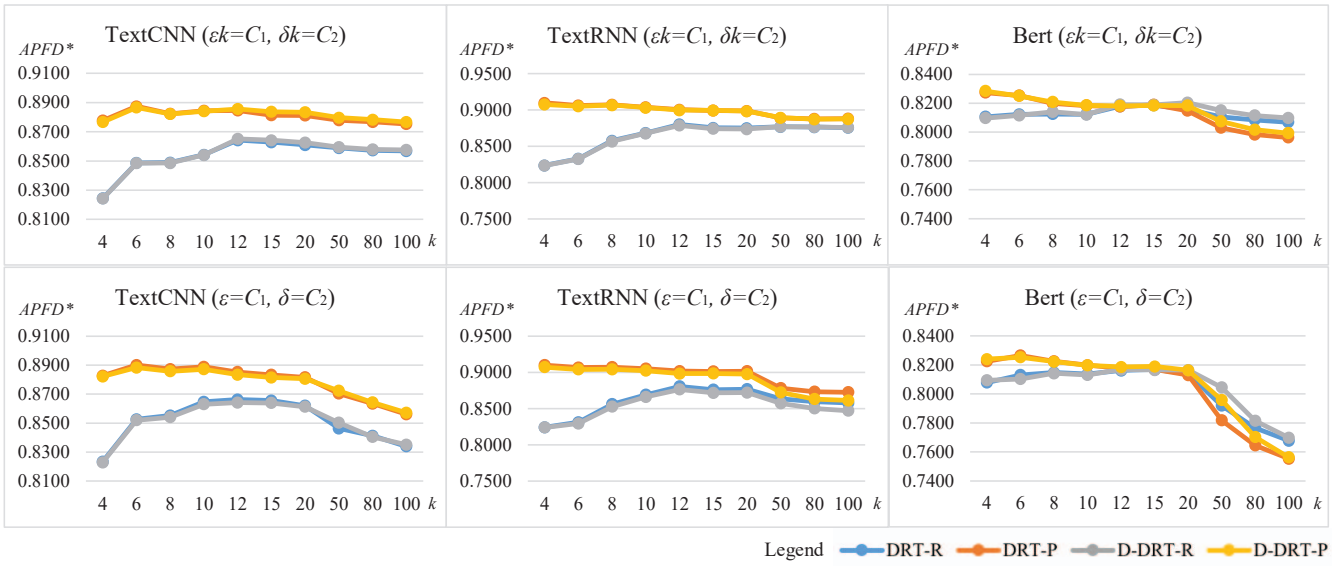

Figure 3 Effect of parameter tuple (k, ε and δ) on performance of DNN-DRT

C. Experimental Settings and Process

Three series of experiments are conducted in this paper to answer the proposed research questions. For RQ1, we compare DNN-DRT strategies with RT, Gini, Entropy and GE-0.5 in terms of fault detection effectiveness. For RQ2, we conduct experiments on four DNN-DRT strategies under different parameter tuples. For RQ3, the performances of DNN-DRT are examined under different datasets. Besides, all the mentioned datasets are applied in the experiment of RQ1 and only the *Tweets* is adopted in RQ2 and RQ3 for further research.

The experiment is executed on a PC, which is powered by a 1.80GHz Intel (R) Core (TM) i7-8550U CPU Quad processor and has 16.0 GB RAM. A testing platform is developed to conduct the experiment automatically and an automated test oracle is utilized to detect failure. A failure is triggered if the output class of DNN are inconsistent with the ground label.

The experiment is conducted on three types of text datasets and DNN models that is nine versions of DL software need to be tested. Test cases are selected according to the specific testing strategies and the stopping criterion for testing is that the entire test set is executed. Each testing process with randomness is repeated for 100 times to avoid deviation as these strategies have certain stochasticity.

V. Experimental Results

A. RQ1: Fault Detection Effectiveness

As can be seen from Table 2, in terms of effectiveness, Gini performs best in most cases compared with other test prioritization strategies while DRT-P and D-DRT-P with the guidance of test prioritization significantly outperform DRT-R and D-DRT-R. And all the $APFD^*$s of DRT-P and D-DRT-P with the feedback mechanism in the *Tweets*, *Reviews* and *News* are not inferior to those of Gini, Entropy and GE-0.5 merely with test prioritization. Especially for *Tweets* in the datasets and TextCNN in the DNN models, the improvement of DRT-P and D-DRT-P over Gini for effectiveness is more obvious, where the maximum and minimum of optimized degree are respectively 0.0121(*Tweets*, TextCNN) and 0.0006 (*News*, TextCNN).

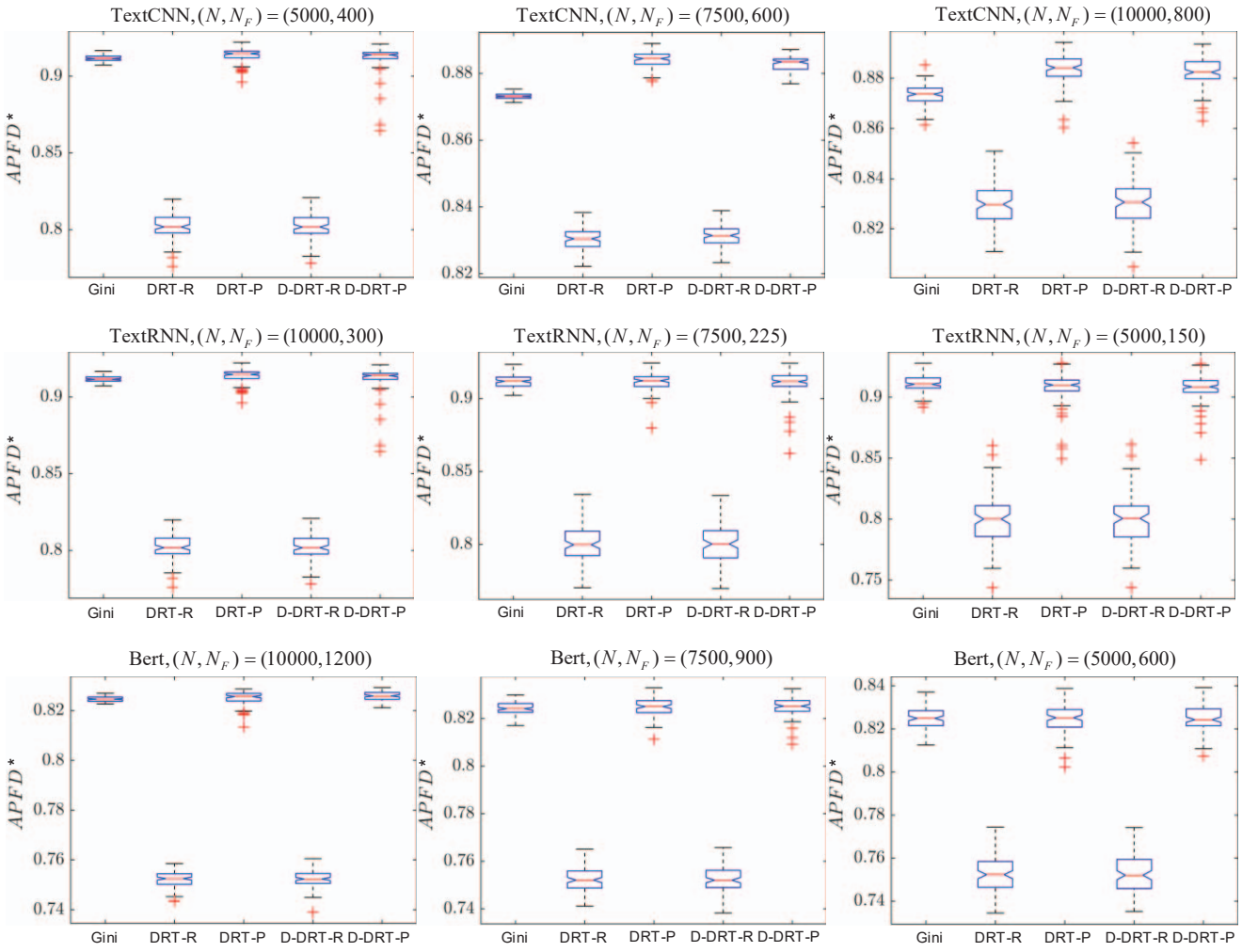

Figure 4 Fault detection effectiveness of DNN-DRT under different datasets

B. RQ2: The effect of parameter tuple (k, ε, δ) on DNN-DRT

In this section, we investigate the effect of parameter tuple (k, ε, δ) on performance of DNN-DRT strategies, in which 10 different ks (from 4 to 100 and unequally spaced) are examined. Previous experiments on DRT usually ignore the relationship between adjusting parameters (ε and δ) and the number of subdomains k. Intuitively, if parameters ε and δ remain unchanged, then the testing profile adjustment process will be sensitive when the number of k is large. It may lead to drastic fluctuation of the testing profile and thus decrease the fault detection effectiveness. Therefore, we set two kinds of settings of adjusting parameters in this section, which is as follows.

1) The value of the parameters (ε, δ) is fixed, that is $\varepsilon = C_1, \delta = C_2$.

2) The value of the parameters (ε, δ) is related to k that is $\varepsilon k = C_1, \delta k = C_2$. Namely, the increment and decrement of the selected probability of the subdomain are set as ε/k and δ/k.

The experimental results are shown in Figure 3 and C_1, C_2 are set as 0.05 and 0.01. We can conclude that k has major effect on effectiveness. As k increases, the performa-nce of DNN-DRT increases first and then decreases. Besides, when the values of the parameters (ε, δ) are related to k, all the four DNN-DRT strategies under three DNN models fluctuates in a small range, and the test effectiveness does not show an obvious downward trend even k is large. When the values of the parameters (ε, δ) are fixed, the testing effectiveness shows a significant downward trend when k is large. It can be verified that the parameters (ε, δ) have effect on performance of DNN-DRT, and associating (ε, δ) with k might be a better approach when applying DRT in practice.

C. RQ3: Comparison on fault detection effectiveness of DNN-DRT under different datasets

Datasets in NLP are characterized by the scarcity of valid data and the high cost of manual labeling. However, a well-performing testing strategy should have low dependencies on specific datasets, that is, its test results don't fluctuate greatly with the sizes of test sets and influence of specific test cases, so it is necessary to study the impact of test sets on test results. In the following experiment, a series of test sets are generated with different N and N_F/N in which test cases and inserted

faults are sampled from the original data set randomly in the following experiment.

Assume that the faults in the original data set are uniformly distributed, that is fault proportionN_F/N of generated test sets by random sampling keeps similar to the original. So, fault proportion settings of test sets in the same DNN are set equivalent and those in the different DNNs are related to the estimated prediction accuracies after the initial testing.

In the experiment of RQ1, Gini is generally better than the other two strategies based on test prioritization in effectiveness, so this experiment only compares Gini with four types of DNN-DRT strategies. Since the generation and testing process of the test sets under the same parameter tuple (N, N_F) have certain randomness, the test method may be carried out by repeating 100 times. The following boxplot may be used to represent the test effectiveness with $APFD$ as the metric.

It can be found from Figure 4 that for different test sets under the same dataset and model, the median values of the testing strategies are approximately constant. As N decreases of a certain dataset and model, the results become more divergent and random. On the contrary, the introduction of test prioritization reduces the statistical uncertainty. For DRT-P and D-DRT-P strategies, the ranges of $APFD$s are wider than those of Gini and narrower than those of DRT-R and D-DRT-R, but they usually achieve larger non-outlier maximum values while their non-outlier minimum values don't drop too much.

D. Summary

It can be observed that the proposed DRT-P and D-DRT-P strategies outperform RT, Gini, Entropy, GE-0.5, DRT-R and D-DRT-R in terms of APFD in the nine versions. Test prioritization greatly improves the effectiveness of testing strategies by introducing reasonable reference from the DNN output. The closed-loop feedback mechanism optimizes the original strategy, and the effect is obvious in certain versions. Therefore, DRT-P and D-DRT-P strategies can be considered as better strategies and optimizations in general.

The initial experimental results show that DRT-P and D-DRT-P are slightly better than Gini under part of circumstances. As the matter of fact, the parameter tuple (k, ε, δ) has greatly influence on the test effectiveness of DNN-DRT. The results of comparison indicate that if the strategies maintain effective, the adjustment parameters (ε, δ) should decrease accordingly, when k increases.

How different test sets influence the fault detection of testing strategies is also investigated and the results reveal that the average effectiveness of both test prioritization and DNN-DRT strategies isn't affected by sizes of test sets and specific test cases. In particular, APFDs of DRT-P and D-DRT-P strategies tend to achieve higher upper limit while ensuring that the lower limit is not too low, that is, it's more likely to obtain a more satisfactory result than that of test prioritization in a single run.

VI. Threats to Validity

Some potential threats to the validity of our experimental study are discussed in this section.

First, one obvious threat to validity is the selection of subject programs. In our experiment, the testing strategies are conducted on three NLP datasets and three DNN models. They cannot represent all test objects since the number of datasets and models are limited. It is hard to guarantee that our strategies will exhibit similar results on other programs. Nevertheless, these datasets and DNN models have been widely used in many practical studies to investigate testing performance. They can reflect the data characteristics and fault distribution in real situations to certain extent. The principle of DNN-DRT can also be applied in many other forms. We look forward to the application of our techniques in more areas.

Second, a possible threat is related to the measures of fault detection effectiveness. In our study, we use APFD to evaluate the performance of testing strategies. Other widely used metrics, such as F-measure (expected number of test case executions required to detect the first failure in a specific test run) and E-measure (expected number of faults being detected by executing a certain number of test cases) are not applied in experiment. These metrics can be included in our future study to evaluate testing strategies from more perspectives. Nevertheless, to increase the credibility of experimental results, each trial is repeated 100 times to avoid bias, and the conclusion of our experiment can be adequately supported.

In addition, the training of DNN models requires a relatively long time, and the setting of epochs and iteration times might have effect on accuracy of models. Therefore, it is difficult to guarantee that our methods and models can be fully reproduced.

VII. Related Works

In this section, some of the major state-of-the-art works on software testing strategies and DNN testing strategies are introduced.

A. Software testing strategies

Random Testing (RT) is a well-known software testing technique [21]. In RT, the test cases are selected randomly from the input domain according to any given distribution [22]. The simplicity of RT makes it widely used in many testing fields. However, RT ignores the structural information concerning software and historical data during the testing process, which may lead to several difficulties in improving their effectiveness through internal or external guidance. Some testing strategies were proposed to enhance the fault detection effectiveness, including Adaptive Testing (AT) [23], Adaptive Random Testing [24] and DRT [10].

AT is in the context of software cybernetics which emphasizes the interplay of control science and software engineering [23]. In AT, the software testing process is modeled as an optimal and adaptive control problem. And the test case selection process can be adjusted online to make an optimal testing decision [25]. Experimental results show

that AT outperforms RT in terms of fault detection effectiveness, however, it requires additional execution time in real applications. Therefore, some AT-based strategies have been proposed to improve fault detection effectiveness and efficiency, including AT-RT hybrid approach [26], AT strategy based on moment estimation [27] and AT strategy based on the coverage spectrum and operational profile [28]. Another widely studied testing strategy is ART [24], which aims at using fewer test cases to detect the first failure. The intuition of ART is that the failure-causing inputs are clustered, and selecting a test case far away from previously executed non-failure-causing test cases will be more likely to detect a failure. In ART, the test case is not selected randomly but rather selected by calculating the total distance from all previously selected test cases such that the total distance is maximized [29]. Many ART-based algorithms were proposed due to the inception of ART, such as Mirror ART [30], Randomized Random Testing (RQRT) [31], and Random Border Centroidal Voronoi Tessellations (RBCVT) [32]. Both AT and ART can significantly improve fault detection effectiveness, however, the sophisticated algorithm behind them hinders the enhancement of testing efficiency. Besides, the intuitions of ART and AT are similar to our approach since they are based on the same hypothesis of similar behavior.

DRT is also in the context of software cybernetics [23], which dynamically changes the testing profile so that the test cases with higher failure detection rates will have higher selection probabilities. Some improved DRT strategies have been proposed, i.e., history-based DRT (DRT-h) [33], adaptive DRT (A-DRT) [34] optimization DRT (O-DRT) [35], Adaptive Partition Testing (APT) [36] and D-DRT [13], aiming at enhancing the fault detection effectiveness. The experimental results have shown that these DRT-based strategies can improve the effectiveness under certain circumstances. However, the setting of optimal parameters may vary with different types of software. Our study focusses on combining the principle of DRT into DNN testing to improve the testing process, The ideas behind these methods can also be applied in the D-DRT-P to enhance the fault detection effectiveness in future works.

B. DNN testing strategies

Inspired by traditional test coverage metrics, a few researchers believe that the distribution of neuron activation values plays an important role in DL software. They propose a series of structured test coverage metrics based on neuron activation values by counting and tracking the distribution of neuron activation values or the changing relationship between adjacent neuron layers.

Coverage criteria is a type of testing strategies borrowed from traditional software testing in the early days. Some researchers believe that the distribution of neuron activation values plays a vital role in intelligent software testing. In 2017, Pei et al. proposed DeepXplore [3], the first white-box testing framework based on neuron coverage in real DL systems. And DeepCover [37] introduces more coverage criteria such as symbol-symbol, distance-symbol, symbol-value, and distance-value coverage. Later, DeepGauge [4], DeepCT [5], DeepPath [34], etc. have done further research on coverage-guided testing criteria by extracting the internal information of the DNNs. Compared with the black-box model, the coverage criteria based on the white-box model requires deep analysis of the neuron states in the DNN, which greatly increases the complexity and has great limitations in its testing effectiveness.

Test case prioritization, another type of DL software-oriented testing mechanism, has the advantage of being relatively simple and effective. It prioritizes test cases by calculating the weight or test significance in the candidate set, aiming to detect the test cases with higher defect detection rates or easier misclassification faster. Kim et al. propose LSA [9] to measure how close an input is to the class boundary. Li et al. propose a test selection strategy of Cross Entropy-based Sampling (CES)[39] to evaluate the accuracy of the operating environment. Feng et al. propose DeepGini based on the statistical perspective of DNN [8], which adopts Gini impurity to estimate the probability of test cases being classified differently in a simple and effective way. Besides, $maxp$ and Geometric diversity-based prioritization (GD) [40], etc. are also used as the metrics to point the suitable order of selection. However, the above prioritization strategies focus on image data more compared with text data. Most of the effective comparisons are based on experimental results, and it's not clear to analyze theoretically and intuitively.

VIII. CONCLUSION

DNN has achieved great process and widely deployed in many domains and suffer from software faults that may cause economic losses. NLP contains richer information and has the characteristics of subjectivity, ambiguity and irregularity, which prompts that extracting insights from text can be challenging and time-consuming. It is essential to ensure the quality and reliability of NLP-DNN models. However, DNN testing is still at early stage and existing strategies might not sufficiently effective.

We propose a D-DRT-P strategy in this paper. It utilizes the priority information and distance information to enhance the DRT strategy. The priority information is used to classify the test suite and guide the test case selection inside the subdomains, and distance information is adopted to adjust the testing profile, along with testing results.

We conducted experiments to compare the four kinds of DNN-DRT with other test prioritization strategies on three NLP datasets and three well-known DNN models. The experimental results demonstrate that DNN-DRT strategies can achieve better performance than RT, Gini, entropy and GE-0.5 in most cases. And we find that both priority information and distance-based adjustment can enhance the effectiveness of DRT, and priority information might be more effective in improving the fault detection effectiveness. Besides, the parameter tuple (k, ε, δ) has greatly influence on the test effectiveness of DNN-DRT, and associating (ε, δ) with k might be a better choice when applying DNN-DRT in practice. In addition, the effectiveness of DNN-DRT strategies is little affected by sizes of test sets and specific test cases.

Future works include combining more feature analysis methods and uncertainty measurement methods into D-DRT-

P strategy, so that the fault detection capacity of test cases can be evaluated more comprehensively. Besides, we tend to conduct experiments on wider range of datasets and models, and verify the effectiveness and efficiency of our strategies through more metrics.

ACKNOWLEDGMENT

This work is supported in part by National Key R&D Program of China under Grant 2021YFB1600601, in part by the National Natural Science Foundation of China under Grant 61772055 and Grant 61872169.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." Nature, 2015, 521(7553):436.

[2] M. Shervin, et al. "Deep learning--based text classification: a comprehensive review." ACM Computing Surveys (CSUR) 2021, 54.3: 1-40.

[3] K. Pei, Y. Cao, J. Yang, et al. "Deepxplore: Automated whitebox testing of deep learning systems". Mobile Computing and Communications Review, 2018, 22(3):36-38.

[4] L. Ma, Y. Liu, J. Zhao, et al. "DeepGauge: Multi-granularity testing criteria for deep learning systems." ACM/IEEE International Conference. ACM, 2018:120-131.

[5] L. Ma, J. Xu, F, M. Xue, et al. "DeepCT: Tomographic combinatorial testing for deep learning systems." 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019: 614−618.

[6] Z. Li, X. Ma, C. Xu and C. Cao. "Structural coverage criteria for neural networks could be misleading." Proc. of the 41st Int'l Conf on Software Engineering: New Ideas and Emerging Results. 2019: 89−92.

[7] J. Chen, M. Yan, Z. Wang, Y. Kang, and Z. Wu, "Deep neural network test coverage: How far are we?" 2020. arXiv preprint arXiv:2010,04946.

[8] Y. Feng, Q. Shi, X. Gao, et al. "DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks." Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020:177–188.

[9] J. Kim, R. Feldt and S. Yoo. "Guiding deep learning system testing using surprise adequacy." 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) 2019.

[10] K.Y. Cai,, H. Hu, C.H Jiang, and F. Ye. "Random testing with dynamically updated testing profile." Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE 2009), 2009: 1-2.

[11] H. Pei, K.Y. Cai, B. Yin, A.P. Mathur, and M. Xie, "Dynamic random testing: Technique and experimental evaluation." IEEE Transactions on Reliability, 2019, 68(3):872-892.

[12] Rothermel, Gregg, et al. "Prioritizing test cases for regression testing." IEEE Transactions on software engineering. 2001. 27.10: 929-948.

[13] H. Pei, B. Yin , M. Xie and K.Y. Cai. "Dynamic random testing with test case clustering and distance-based parameter adjustment." Information and Software Technology, 2021, 131(12):106470.

[14] K.Yoon. "Convolutional neural networks for sentence classification [OL]." arXiv Preprint. 2014.

[15] P. Liu, X. Qiu and X. Huang. "Recurrent neural network for text classification with multi-task learning." arXiv preprint arXiv:1605.05101. 2016.

[16] D. Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805. 2018.

[17] D.Thomas, et al. "Automated hate speech detection and the problem of offensive language." Proceedings of the international AAAI conference on web and social media. 2017, 11(1).

[18] https://www.kaggle.com/datasets/kotartemiy/topic-labeled-news-dataset.

[19] https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

[20] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G.Rothermel, "A static approach to prioritizing junit test cases." IEEE Transactions on Software Engineering, 2012, 38(6):1258-1275.

[21] W.J. Gutjahr. "Partition testing vs. random testing: The influence of uncertainty." IEEE Transactions on Software Engineering, 1999, 25(5): 661-674.

[22] T.Y. Chen, and Y.T. Yu, "On the relationship between partition and random testing." IEEE Transactions on Software Engineering, 1994, 20(12), 977-980.

[23] K.Y. Cai. "Optimal software testing and adaptive software testing in the context of software cybernetics." Information and Software Technology, 2002, 44(14):841-855.

[24] T.Y. Chen, T. Tse, and Y. Yu. "Proportional sampling strategy: a compendium and some insights." Journal of Systems and Software, 2001, 58(1): 65-81.

[25] K.Y. Cai, B. Gu, H. Hu, and Y. Li, "Adaptive software testing with fixed-memory feedback." Journal of Systems and Software, 2007, 80(8):1328-1348.

[26] J. Lv, H. Hu, K.Y. Cai. and T.Y. Chen, "Adaptive and random partition software testing.". IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2014, 44(12):1649-1664.

[27] P. Xiao, Y. Yin, Liu, B. Jiang, and Y.K. Malaiya, "Adaptive testing based on moment estimation." IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017 50(3):911-922.

[28] A. Bertolino, B. Miranda, R. Pietrantuono, and S. Russo. "Adaptive test case allocation, selection and generation using coverage spectrum and operational profile." IEEE Transactions on Software Engineering, 2019:1-18.

[29] J. Chen, L. Zhu, T.Y. Chen, D. Towey, F.C. Kuo, R. Huang, and Y. Guo. "Test case prioritization for object-oriented software: An adaptive random sequence approach based on clustering." Journal of Systems and Software, 2018, 135:107-125.

[30] R. Huang, H. Liu, X. Xie, and J. Chen. "Enhancing mirror adaptive random testing through dynamic partitioning." Information and Software Technology.2015, 67:13-29.

[31] H. Liu, and T.Y. Chen, "Randomized quasi-random testing." IEEE Transactions on Computers, 2015, 65(6):1896-1909.

[32] A.Shahbazi, A.F. Tappenden, and J. Miller, "Centroidal voronoi tessellations-a new approach to random testing." IEEE Transactions on Software Engineering. 2012, 39(2):163-183.

[33] L. Zhang, B. Yin, J. Lv, K.Y. Cai, S.S. Yau and J. Yu. "A history-based dynamic random software testing." Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International, 2014:31-36.

[34] Z. Yang, B. Yin, J. Lv, K.Y. Cai, S.S.Yau, and J. Yu. "Dynamic random testing with parameter adjustment." Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International, 2014:37-42.

[35] Y.,Li, B. Yin, J. Lv, and K.Y. Cai, "Approach for testing profile optimization in dynamic random testing." Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. 2015, (3): 466-471.

[36] C.A. Sun, H. Dai, H. Liu, T.Y. Chen, and Cai, K.Y. "Adaptive partition testing." IEEE Transactions on Computers, 2018, 68(2), pp.157-169.

[37] Y. Sun, X. Huang and D. Kroening. "Testing deep neural networks." arXiv preprint arXiv:1803.04792, 2019.

[38] J. Sekhon, C. Fleming. "Towards improved testing for deep learning." Proc. of the 41st Int'l Conf. on Software Engineering: New Ideas and Emerging Results. 2019: 85−88.

[39] Z. Li, X.Ma, C. Xu, C. Cao, J. Xu, and J. Lü. "Boosting operational DNN testing efficiency through conditioning." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 499–509.

[40] Z. Aghababaeyan, M. Abdellatif, , L. Briand and M. Bagherzadeh, "Black-box testing of deep neural networks through test case diversity." 2021, arXiv preprint arXiv:2112.1259.