A Novel Framework for Locating Software Faults Using Latent Divergences

Shounak Roychowdhury and Sarfraz Khurshid

Department of Electrical and Computer Engineering,
University of Texas at Austin,
Austin, Texas, 78712-0240, USA
{sroychow,khursid}@ece.utexas.edu

Abstract. Fault localization, i.e., identifying erroneous lines of code in a buggy program, is a tedious process, which often requires considerable manual effort and is costly. Recent years have seen much progress in techniques for automated fault localization, specifically using program spectra – executions of failed and passed test runs provide a basis for isolating the faults. Despite the progress, fault localization in large programs remains a challenging problem, because even inspecting a small fraction of the lines of code in a large problem can require substantial manual effort. This paper presents a novel framework for fault localization based on latent divergences – an effective method for feature selection in machine learning. Our insight is that the problem of fault localization can be reduced to the problem of feature selection, where lines of code correspond to features. We also present an experimental evaluation of our framework using the Siemens suite of subject programs, which are a standard benchmark for studying fault localization techniques in software engineering. The results show that our framework enables more accurate fault localization than existing techniques.

1 Introduction

In software engineering, the process of locating the faults, i.e., selecting a set of faulty statements, in a buggy program is called *fault localization*. In machine learning, the process of selecting the most relevant features from a set of possible features is called *feature selection*. Both the processes are about selection. Therefore, the central idea of this paper is to reduce the problem of fault localization to the problem of feature selection, leverage ideas and constructs from the feature selection research to solve the fault localization problem.

There is a growing demand to seek alternatives to the traditional ways of manually debugging large-scale systems by using automated techniques. Some of the promising attempts include algorithmic debugging methods [3], static and dynamic program slicing [5], and most recently to apply data analysis techniques like Statistical Debugging [15]. Program spectrum-based methods [18], which are also known as coverage-based technique, record the execution information of a program. They deal with how statements and branches are executed with

D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part III, LNAI 6913, pp. 49–64, 2011.

respect to a set of successful runs (positive test cases) and unsuccessful runs (negative test cases). The basic insight of the spectrum-based techniques is that similar test cases generate similar execution spectrum while dissimilar test cases generate very different types of execution spectrum. Another effective alternative technique is the predicate-based intrumentation based technique in which boolean predicates are injected within the code that collect the run-time statistics of the program. Statistical Debugging is an example of such a technique [15]. The spectrum-based methods have a two fold advantage over the predicate-based instrumentation methods: firstly it does not overly instrument the source code, and secondly a variety of code coverage tools are easily available in the market that can be used without affecting the runtime performance of source code. This paper presents a novel spectrum-based framework for fault localization based on latent divergences – an effective method for feature selection in machine learning.

The purpose of this paper is to seek a methodology in which the code coverage data is modeled as a probabilistic data source and use tools such as probabilistic divergences and their combinations to extract faulty lines of code. Our proposed method provides an alternative to methods that employ similarity based measures. For doing so, we introduce a new concept called *latent divergence*, which is in fact a product of divergences based on different conditional probabilities. These probabilties are derived from the code coverage data. Through this mechanism we extract potentially hidden information that can effectively be used for selecting faulty lines of code. Furthermore, it should be noted that latent divergence does not use any latent variables. The term "latent" in latent divergence indicates that these measures try to extract hidden information from conditional divergences.

The main contributions of this paper are as follows: 1) It proposes a novel approach of fault localization using latent divergence, a new concept based on conditional probabilistic divergences. 2) It proposes a family of measures based on latent divergence. 3) It gives a framework for using latent divergences in fault localization. 4) The experimental results show that our method performs better than state-of-the-art methods.

1.1 Related Work

In machine learning, feature selection deals with the process of selecting a subset of features without degrading accuracy or classification performance. There are three major feature selection mechanisms: filter, wrapper, and embedded selection. For details, see [11].

In recent days machine learning techniques are being applied to debugging. Neural network based methods have been proposed by Wong et al [21] in the context of fault localization. They have used classic Back-Propagation algorithm and Radial Basis Functions (RBF) based neural networks for fault localization. Jiang and Su [13] proposed Context-Aware Statistical Debugging technique that considers not only individual bug predictors and control flow paths that connect those predictors. They used Support Vector Machines (SVM) and Random Forests (RF) for statistical classification. Use of such intense computational ap-

proach makes the debugging quite slow. The authors have also observed that a single machine learning technique is not be able to properly rank predicates. These methods can be quite sensitive to way the classifiers are trained.

2 Motivation

Usually similarity measures are often used for comparing sets of data. For dichotomous data, similarity measures are based on the combination of four components of the binary contingency matrix. In the pattern recognition literature [17] several different dissimilarity measures have been widely used. The use of similarity based measures in fault localization research was started by Jones et. al [14] and Abreu et. al. [1]. The measure tuple (*suspiciousness, confidence*) was proposed by Jones et. al. in their Tarantula system. Abreu et. al. [1]. used the Ochiai metric as their similarity measure in their study. In fact, this measure was first proposed by Ochiai in a biological study [16] in 1957. While experimenting with some of these measures for our fault localization research, we observed that some of the binary similarity measures like Yule or Kulcsyzski as described in [17], which have range of $[0, \infty)$, were not effective in fault localization problem. Therefore, we were motivated to seek other alternatives to similarity-based measures for our research.

The inspiration for the proposed method comes from elementary geometry. Specifically, it comes from the theorem of the *Power of a point with respect to a circle*. This is an old and well-known result in the area of inversive geometry. See Fig.1. In simple terms, the theorem states that for a given circle with a center O and radius r, and a point A that lies outside the circle, such that P and Q are the intersections of a line through point A with the circle, then the power (p) of the point A is given by:

$$p = AP \times AQ. \tag{1}$$

This was first described by Steiner in 1826 [20]. Coxeter and Greitzer [8] showed that $p = AO^2 - r^2$. Furthermore, we use Eqn.(1)

$$AO = \sqrt{AP \times AQ + r^2}. \tag{2}$$

The interesting part is when $p = 0$ then the point A is lies on the circle. Moreover, when $p > 0$, the point lies outside the circle, and when $p < 0$ the point lies inside

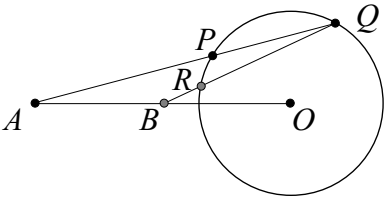

Fig. 1. Power of point with respect to a circle

the circle. Thus it might be quite effective to use this fact to compare relative positions of points.

Without any loss of generality let us consider two points A and B that lie outside the circle. In order to compute the relative distance between point A and point B it might be possible to just compare the lengths of the segments $AO = \sqrt{(AP \times AQ + r^2)}$ and $BO = \sqrt{(BR \times BQ + r^2)}$ where r is the radius the circle. To approximately compute $|AO - BO|$, it might be reasonable to ignore r for a given circle as it is a constant in both of the expressions, thus just compute $AP \times AQ$ and $BR \times BQ$ in order to find relative distance. This formed the basis of our insight to multiply two distances or two probabilistic divergences in our case to find a way of measuring *relative distance* or *separatedness* between the data points. Notice that this technique is quite different when compared to any similarity measure-based techniques. We will further explore this idea in Section 5 from the perspective of conditional probabilistic divergences.

3 An Illustrative Example

In this section we show a simple motivating example that shows the product of conditional divergences are able to accurately identify the faulty lines of code. We show a sample C code in Fig.2 and test cases in Table 1 that were used by Jones et. al. in their paper as an example. The purpose of the code is to find out the middle number. The line numbers followed a colon are shown in the left margin. Apparently a simple looking code is faulty for few inputs. We observe that the fault is in line number 7. The correct code is shown as a comment embedded in that same line.

```
#   mid() {
1:      read("Enter 3 numbers:",x,y,z);
2:      m = z;
3:      if (y<z)
4:          if (x<y)
5:             m = y;
6:          else if (x<z)
7:             m = y;   // fault1. correct: m=x
8:      else
9:         if (x>y)
10:          m = y;
11:       else if (x>z)
12:          m = x;
13:   print("Middle number is:",m);
#   }
```

Fig. 2. This simple C code is taken from Jones et al. [14] as a motivating example

Table 1. This table shows the execution-statement hit for each line using 6 different test-cases. The test cases t1-t5 generates a passed output (denoted by T(1)). The test case t6 generates a failed output (denoted by F(0)).

	1	2	3	4	5	6	7	8	9	10	11	12	13	R
t1	1	1	1	1	0	1	1	0	0	0	0	0	1	T(1)
t2	1	1	1	1	1	0	0	0	0	0	0	0	1	T(1)
t3	1	1	1	0	0	0	0	1	1	1	0	0	1	T(1)
t4	1	1	1	0	0	0	0	1	1	0	1	0	1	T(1)
t5	1	1	1	1	0	1	0	0	0	0	0	0	1	T(1)
t6	1	1	1	1	0	1	1	0	0	0	0	0	1	F(0)

Table 1 shows the execution-statement hits for 6 test cases such that t1 = (3,3,5), t2=(1,2,3), t3=(3,2,1), t4=(5,5,5), t5=(5,3,4), t6=(2,1,6). Rows denote the test cases and the columns denote the line numbers. The last column denotes output of the program with respect to the test cases. We observe that out of the 6 test cases only t6 gives an incorrect result. Fig.3 shows five subplots. Each subplot shows the values of latent divergences with respect to each line number for a set of test cases. The subplot (1) shows the result of two test cases t1 and t6. The test case t1 produces successful output (T(1)). The test case t6 produces an unsuccessful output (F(0)). The result column, which is column 14, records the result of each test case. The values of t1 and t6 are the same for all columns: from column 1 to column 13. Since there is no difference in information between

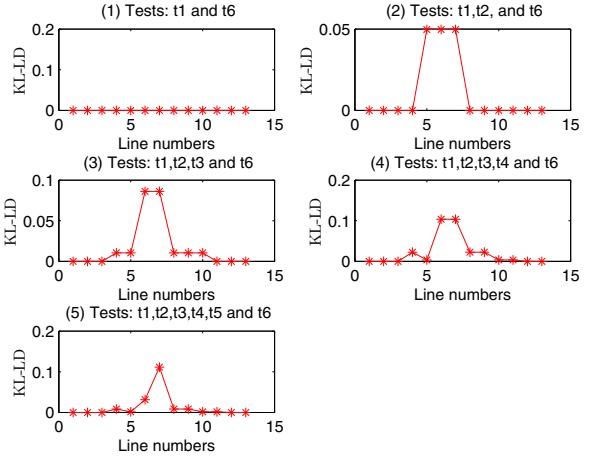

Fig. 3. This figure shows latent divergence using KL-Divergence (KL-$\mathcal{LD}$) for 6 test-cases in 5 subplots. The x-axis shows the line numbers, and y-axis shows the Latent divergence. Subplot (1) shows the latent divergence using 2 tests t1 & t6. Similarly other plots show result for different number of testcases.

two test cases, the latent divergence is zero for all the lines. The subplot (2) shows the results of adding another test case t2 to the existing set of test cases. In this scenario, we observe some changes in the values of the latent divergences at line numbers 5, 6, and 7. When we add more test cases as seen in remaining subplots, we notice that the values of latent divergence start to converge. In subplot (5), we use all the six test cases, and where we notice that, the latent divergence peaks at the line number 7 where the fault resides.

Table 2 compares the value of the metrics for Tarantula, Ochiai, and other proposed measures. Herein, we see that Tarantula and Ochiai measures have values 0.5 and 0.7071 at line 1 which is a little difficult to interpret. On careful observations, we further notice that the nonzero values of both these metrics are greater or equal to 0.5. For Tarantula, the base computed value is 0.5 ($= \frac{1}{1+1}$). For Ochiai metric, the base value is 0.7071. The computed values for the other lines are relatively close to the base value for these metrics. In this context, we believe that the power of the metric lies in its ability to differentiate the incorrect lines of code from the correct ones by significant margins. As this table shows, it is possible to localize bugs by using latent divergence.

Table 2. Comparison of Tarantula and Ochiai metrics different types of probabilistic divergences used in Definition 1 of Latent Divergence. KL-$\mathcal{LD}$ is the latent divergence uses KL-divergence, JS-$\mathcal{LD}$ uses Jensen-Shannon Divergence, R(α)-$\mathcal{LD}$ uses Renyi-Divergence, and IS-$\mathcal{LD}$ uses Itakuro-Saito divergence.

	1	2	3	4	5	6	7	8	9	10	11	12	13
Tarantula	0.50	0.50	0.50	0.625	0	0.71	0.83	0	0	0	0	0	0.5
Ochiai	0.7071	0.7071	0.7071	0.7906	0	0.8452	0.9129	0	0	0	0	0	0.7071
KL-$\mathcal{LD}$	0	0	0	0.0095	0.0037	0.0444	0.2110	0.0160	0.0160	0.0037	0.0037	0	0
JS-$\mathcal{LD}$	0	0	0	0.0366	0.0064	0.1312	0.4607	0.0366	0.0366	0.0064	0.0064	0	0
R(2)-$\mathcal{LD}$	0	0	0	0.0089	0.0015	0.0332	0.1072	0.0089	0.0089	0.0015	0.0015	0	0
R(0.5)-$\mathcal{LD}$	0	0	0	0.0019	0.0003	0.0069	0.0244	0.0019	0.0019	0.0003	0.0003	0	0
IS-$\mathcal{LD}$	0	0	0	0.8730	0.1617	2.8838	8.8456	0.8730	0.8730	0.1617	0.1617	0	0

4 Probabilistic Divergences

Probabilistic divergence is a well-known concept like dissimilarity measure but in the probabilistic space. The divergence actually measures the distance between two probability mass functions $p(x)$ and $q(x)$ such that $x \in \mathcal{X}$, and $\mathcal{X}$ is the alphabet. The standard notation for divergence is $D(p||q)$. It plays a significant role in the areas of pattern recoginition, learning and inference, and optimization.

4.1 KL-Divergence

Information and coding theory extensively use Kullback-Liebler divergence also known as KL-Divergence or Relative Entropy. Kullback-Liebler Divergence (KLD) is given by:

$$D_{KL}(p||q) = \sum_{x \in \mathcal{X}} p(x)\ log\left(\frac{p(x)}{q(x)}\right). \tag{3}$$

KL-divergence measures the distortion between two probability mass functions $p(x)$ and $q(x)$. In terms of entropies the Eqn.(3) can be written as follows: $D_{KL}(p||q) = H(p,q)$ - $H(p)$, where $H(p,q)$ is the cross-entropy between p and q and $H(p)$ is the entropy of p. It is not a true metric as it does not satisfy the symmetric and triangular properties of a metric. Jensen-Shannon Divergence (JSD) is given by the mean of two Kl-divergences:

$$D_{JS}(p||q) = \frac{1}{2}\bigg(D_{KL}(p,q) + D_{KL}(q,p)\bigg) \tag{4}$$

where $D_{KL}(p||q) = \sum_{x\in\mathcal{X}} p(x)\ log(\frac{p(x)}{q(x)})$ and $D_{KL}(q||p) = \sum_{x\in\mathcal{X}} q_(x)\ log(\frac{q(x)}{p(x)})$. JS-divergence satisfies three properties of being a metric.

4.2 α-Divergences

Renyi [19] proposed a following class of divergence (R(α)), which is given by:

$$D_\alpha(p||q) = \frac{1}{1-\alpha} log\bigg(\sum_{x\in\mathcal{X}} \frac{p(x)^\alpha}{q(x)^{\alpha-1}}\bigg). \tag{5}$$

It is a generalization of KL-divergence when $\alpha \geq 0$. When $\alpha = 0.5$, $D_{0.5}(p||q) = -2log\sum_{x\in\mathcal{X}} \sqrt{p(x)q(x)}$ which is related to Bhattacharyya coefficient. When $\alpha = 2$, $D_2(p||q) = log\ E(p(x)/q(x))$ which is related to log of expected value of ratios of probabilities.

4.3 *f*-Divergences

Another broader class of divergence called $f - divergence$ has been studied by Ali-Silvey [2] and Csiszar [9]. It is given by:

$$D_f(p||q) = \sum_{x\in\mathcal{X}} q(x)\ f\bigg(\frac{p(x)}{q(x)}\bigg), \tag{6}$$

where f is a real valued convex function over the domain of $(0,\infty)$. This is a popular family as it can generate a variety of well-known distances like l_1 norm ($D_f(p||q) = \sum_{x\in\mathcal{X}} |p(x) - q(x)|$), squared-Hellinger distance ($D_f(p||q) = \sum_{x\in\mathcal{X}}(\sqrt{p(x)} - \sqrt{q(x)})$) etc.

4.4 Bregman-Divergences

There is another larger class of divergence called Bregman-divergence [7]. It encapsulates many well-known divergences. Like f-divergence, this class also uses convex functions to generate its members. However, the general formulation of Bregman divergence is $D_{B(f)}(p||q) = f(p) - f(q) - \langle\nabla f(q), p-q\rangle$, where f is convex function and ∇ is the gradient operator, quite different from f-divergence. The equation has flavor of first order Taylor expansion. The Euclidean distance,

Mahalanobis distance, and KL Divergence are special cases of Bregman divergence. Here we will use a non-obvious member of the Bregman family in our study. Itakura-Saito Divergence (IS) [10] belongs to the class of Bregman divergence, and it is given below:

$$D_{B(f)}(p||q) = \sum_{x \in \mathcal{X}} \left(\frac{p(x)}{q(x)} - log \frac{p(x)}{q(x)} - 1 \right). \tag{7}$$

Here the convex function is $f(p) = \sum_{x \in \mathcal{X}} log(p(x))$. This divergence is quite popular in the area of speech and signal processing.

5 Latent Divergence

In this subsection, we introduce a new measure called Latent Divergence, and also propose its general family.

Let X_l denote a Bernoulli random variable for the coverage data of a particular line l such that $p_{X_l}(x) = Pr(X_l = x)$ and p_{X_l} is the probability mass function. Similarly the result is also modeled as Bernoulli random variable R and is denoted by $p_R(x) = Pr(R = x)$ and $p_R(x)$ is the probability mass function. The result of the tests passed and failed are denoted by $(T(1))$ and $(F(0))$ respectively. Since the values of X_l effect the values of R; we note that there is a joint distribution between X_l and R, and it is given by $p(x, r) = Pr(X_l = x, R = r)$. We can calculate the marginals of X_l and R as $P(X_l)$ and $P(R)$. Let us compute the conditional from the perspective of $X_l = 1$ on R, that is $p_{\{R|X_l=1\}}(x)$ $= Pr(\{R|X_l = 1\} = x)$ which is again a conditional Bernoulli pmf. For $X_l = 0$ compute the conditional $p_{\{R|X_l=0\}}(x) = Pr(\{R|X_l = 0\} = x)$, which is again a conditional Bernoulli pmf. Using these conditional random variables we can find a certain amount of information divergence that is hidden within them when measured against random variable R. The idea is to find the amount of divergence with respect to the random variable R. That implies that we are trying to extract information like distance between $\{R|X_l = 0\}$ and R, and $\{R|X_l = 1\}$ and R respectively.

Definition 1. The latent divergence measure $(\mathcal{LD})$ between two Bernoulli random variables X_l and R is defined as follows:

$$\mathcal{LD}(X_l : R) = D(p_{\{R|X_l=1\}}||p_R)D(p_{\{R|X_l=0\}}||p_R). \tag{8}$$

Definition 2. The family of latent divergence measure is denoted by $(\mathcal{FLD})$ between two Bernoulli random variables X_l and R is defined as follows:

$$\mathcal{FLD}(X_l : R|f) = f(\mathcal{LD}(X_l : R)), \tag{9}$$

where $f : (0, \infty) \rightarrow \mathtt{R}^+$ is convex function.

Example 1. When $f_1(x) = x$, then

$$\mathcal{FLD}(X_l : R|f_1) = \mathcal{LD}(X_l : R). \tag{10}$$

Example 2. When $f_2(x) = e^x - 1$, then

$$\mathcal{FLD}_2(X_l : R|f_2) = e^{\mathcal{LD}(X_l:R)} - 1. \tag{11}$$

We show two members of the family and others can be easily be constructed. For this paper, we show only the results using the simplest latent divergence which is given by Eqn.(8).

The motivation section showed that it is possible to compare distances AO and BO using multiplicative factors. We extend that idea to compute distance like measure (separatedness) between two binary vectors with repect to a reference binary vector using probabilistic divergences. A small illustration will clarify the concept. It will also render a geometric flavor to the problem. Let there be two binary vectors such as $X_1 = (1, 0, 1, 1, 0, 1, 1)^t$ and $X_2 = (1, 0, 1, 0, 0, 0, 0)^t$ along with a reference binary vector given by $R = (1, 1, 1, 1, 0, 0, 0)^t$. After all necessary probability transformations, we plot X_1, X_2 and R in terms of their probability components p_0 and p_1 in a Cartesian space. Let p_0 be on y-axis and p_1 be on x-axis. Now see Fig. 4. Therefore, in that space we can represent X_1 as filled black circle and X_2 as filled black square. R is presented by filled red circle. The shaded circle Z_1 and shaded square Z_2 represent $Pr(\{R|X_1 = 0\} = x)$ and $Pr(\{R|X_2 = 0\} = x)$ respectively. Similarly, the unfilled circle Y_1 and unfilled square Y_2 represent $Pr(\{R|X_1 = 1\} = x)$ and $Pr(\{R|X_2 = 1\} = x)$ respectively.

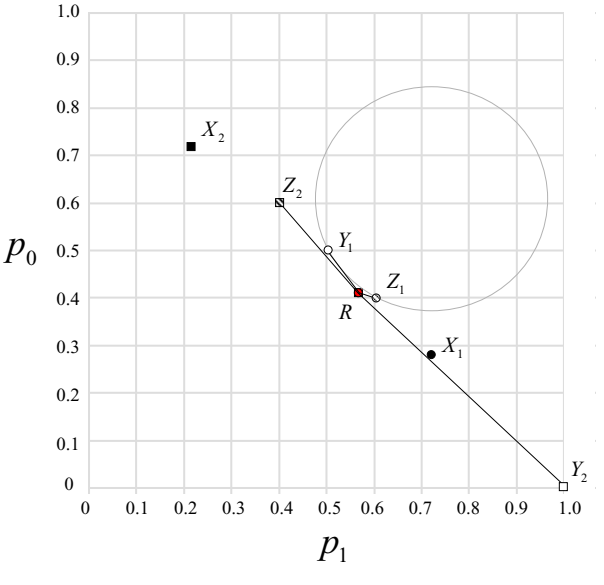

Fig. 4. A Bernoulli random variable is represented in terms components of its probabilities p_0 and p_1. The distances between points like Z_1 and R are actually divergences and should not be confused with Euclidean distance. The intention of the figure is to render a geometric idea.

We find that the latent divergence $\mathcal{LD}(X_1 : R) = 2.2262e - 005$ using KL-Divergence as a base divergence. Next for the other point, the latent divergence is $\mathcal{LD}(X_2 : R) = 0.0690$. Therefore, X_2 has more discrimination than X_1 with respect to R. It should be noted that the Fig. 4 is given to provide a geometry flavor for visualization.

The first property of the metric holds. The latent divergence $\mathcal{LD}(X_l : R) \geq 0$ because component divergences are greater than 0. The symmetric property does not hold as component divergences are not symmetric. The triangular property does not hold as divergences do not satisfy triangular property.

Theorem 1. *$\mathcal{LD}(X_l : R)$ is convex.*

Proof. We know that divergence is a convex function for all classes of divergences. Therefore,

$$D(\alpha p_1 + \beta p_2||\alpha r_1 + \beta r_2) \leq \alpha D(p_1||r_1) + \beta D(p_2||r_2) \tag{12}$$
$$D(\alpha q_1 + \beta q_2||\alpha r_1 + \beta r_2) \leq \alpha D(q_1||q_1) + \beta D(p_2||r_2) \tag{13}$$

Multiply Eqn.(12) and Eqn.(13), and we get the following: $D(\alpha p_1 + \beta p_2||\alpha r_1 + \beta r_2) \times D(\alpha q_1 + \beta q_2||\alpha r_1 + \beta r_2)$

$$\begin{aligned} &\leq (\alpha D(p_1||r_1) + \beta D(p_2||r_2)) \times (\alpha D(q_1||q_1) + \beta D(p_2||r_2)) \\ &= \alpha^2 D(p_1||r_1)D(q_1||r_1) + \\ &\quad \alpha\beta(D(p_2||r_2)D(q_1||r_1) + D(p_1||r_1)D(q_2||r_2)) \\ &\quad \beta^2 D(p_2||r_2)D(q_2||r_2). \end{aligned} \tag{14}$$

From the knowledge of p_1 and q_1, and along with p_2 and q_2, we find that following inequality holds true.

$$[D(\alpha p_1||\alpha r_1) - D(\alpha p_2||\alpha r_2)] \times [D(\alpha q_2||\alpha r_2) - D(\alpha q_1||\alpha r_1)] \leq 0. \tag{15}$$

Rearrange the terms of the above inequality to get

$$D(\alpha p_1||\alpha r_1)D(\alpha q_2||\alpha r_2) + D(\alpha p_2||\alpha r_2)D(\alpha q_1||\alpha r_1) \tag{16}$$
$$\leq D(\alpha p_1||\alpha r_1)D(\alpha q_1||\alpha r_1) + D(\alpha p_2||\alpha r_2)D(\alpha q_1||\alpha r_1) \tag{17}$$

Recall $\alpha + \beta = 1$. Use the above inequality in Eqn.(14) to get the following:

$$\begin{aligned} &\leq \alpha^2 D(p_1||r_1)D(q_1||r_1) + \alpha\beta\big(D(\alpha p_1||\alpha r_1)D(\alpha q_1||\alpha r_1) + \\ &\qquad D(\alpha p_2||\alpha r_2)D(\alpha q_1||\alpha r_1)\big) + \beta^2 D(p_2||r_2)D(q_2||r_2) \qquad (18) \\ &= \alpha D(p_1||r_1)D(q_1||r_1) + \beta D(p_2||r_2)D(q_2||r_2). \qquad (19) \end{aligned}$$

Thus the convexity is preserved for product of two conditional probabilistic divergences. ∎

Algorithm 1. Ranking Algorithm

Require: CodeCoverageMatrix: X, ResultVector: R
Ensure: Lines are ranked.
$M \leftarrow$ GetNumberOfColumns(X)
for $l = 1$ to M **do**
$Y[l] \leftarrow \mathcal{LD}(X_l : R)$
end for
{Normalize the array Y to capture the ranks; the lines having with maximum latent divergence will be ranked 1.}
for $i = 1$ to $|Y|$ **do**
$Z[i] \leftarrow \frac{Y[i]}{max(Y)}$
end for
for $i = 1$ to $|Z|$ **do**
$Q[i] \leftarrow \lfloor Z[i] + M \times (1 - Z[i]) \rfloor$
end for

Theorem 2. *The measures generated by latent divergence family $\mathcal{FLD}(X_l : R|f)$ are convex.*

Proof. Given that $f : \mathbb{R} \to (0, \infty)$ and is a convex function and increasing, and $g : \mathbb{R}^n \to \mathbb{R}$ and g is a convex function, then the composite function $h(x) = f \circ g$ is convex. (See [6]). Using this fact and from Theorem 1 we know $\mathcal{LD}(X_l : R)$ is convex. Thus, $\mathcal{FLD}(X_l : R|f)$=$f(\mathcal{LD}(X_l : R))$ is convex. ■

6 Latent Divergences and Fault Localization

6.1 Algorithm

Given a program P having M set of executable lines, and a set of N test cases. Firstly it is necessary to collect code coverage data X by running both successful test cases and unsuccessful test cases. The result data is collect in result vector R. Table 1 shows a sample of X matrix which has binary values and the result vector R consists of has T or F values.

After the data collection phase of X and R is over, we use the ranking algorithm described in Algorithm1. Let the l^{th} column of the code coverage matrix X be denoted $X_l = X[l]$. The ranking algorithm first computes latent divergence for each column X_l of the coverage matrix with the result vector R. It should be noted that the user selects the type of base divergence like KL-divergence, Renyi etc.

The list of latent divergence of each column is normalized to capture the rank. With respect to the algorithm the normalized data is stored in Z list. We easily observe that the list Z only contains values between 0 and 1. Furthermore, Now use the following formula in which any value of x in interval between a and b can be written as $\lambda a + (1 - \lambda)b$, a formula quite trite in the theory of convex functions. Finally the ranks of the lines are computed by using the following formula:

$$\lfloor Z[i] + M \times (1 - Z[i]) \rfloor ,$$

where M is also the number of columns of X.

Table 3. Summary of the Siemens Test Suite

Program	Description	Versions	LOC	Executable	Testcases
print_tokens	Lexical Analyser	7	565	175	4130
print_tokens2	Lexical Analyser	10	510	178	4115
replace	Pattern replacement	32	563	216	5542
schedule	Priority Scheduler	9	412	121	2650
schedule2	Priority Scheduler	10	307	112	2710
tcas	Altitude Separation	41	173	55	1608
tot-info	Information Measure	23	406	113	1052

The score measure has been used in the literature quite extensively and it is as follows:

$$score_i = \left(1 - \frac{Q[i]}{M}\right) \times 100 \tag{20}$$

We see that the score is computed for each line. Since we are only interested in the lines that are ranked number 1, unlike feature selection where the number of features is usually bigger than 1, we collect the scores reported by those lines. In other words we want the maximum value from the list of scores. On little more careful observation we may easily notice that there is a little drawback with the above equation 20 because it is highly possible that there may be more than one line (or statement) that can have the maximum suspiciousness measure at the same time. In that case the score values for all the lines that are ranked 1 will be same. However, we think that it is quite important and necessary to differentiate between the scenarios when there are multiple lines having rank 1. Thus, we introduce a weight factor W to the score value.

$$W = \frac{M + 1 - |Q^{\#1}|}{M}. \tag{21}$$

Moreover, let us denote $Q^{\#1}$ as a subset of Q whose elements are lines that are ranked 1. $|Q^{\#1}|$ is the cardinality of that subset. Thus, the score of each line then becomes:

$$score_i = \left(\left(1 - \frac{Q[i]}{M}\right) \times W\right) \times 100 \tag{22}$$

It is clear from the above equation that for when $|Q^{\#1}| = 1$, we get Eqn.(22) is equal to Eqn.(20). As only the maximum score value is a signnificant number, so the overall score is given by $score{=}max\{score_1, ..., score_M\}$.

Next we introduce another measure called Metric-Quality and it is denoted by ϕ. It actually measures a quality of a metric that is able to rank least important lines to lower ranks and most important lines to high ranks. It is defined as:

$$\phi = \sum_{i=1}^{M} \frac{1}{Q[i] \times M} \tag{23}$$

We can easily show that ϕ is bounded between $1/M$ and 1 as $Q[i]$ can vary from 1 to M. When the ϕ is small and closer to $1/M$ that implies that the metric is

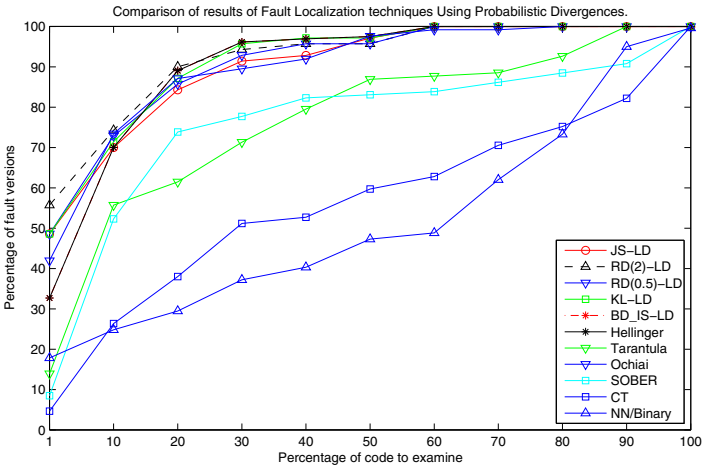

Fig. 5. The Score is plotted between percentage of code to examine and percentage of fault versions

able to suppress less important lines effectively and provide a better quality by ranking important lines much higher.

Refer to Table 2, we observe that when we use metrics like Ochiai, it generates suspiciousness values greater than to 0.9 even though the line may not be part of the bug and that is happens primarily because of the presence of a square root in Ochiai's denominator that increases its value. Such high values may confuse the programmer who might be inclined to look for statements having high suspiciousness values. Therefore, the metric's higher values might not provide the right direction for debugging.

7 Experiments

7.1 Siemens Test Suite

It is a standard practice to use the seven programs of the classic Siemens Test Suite [12] for testing feasiblities of fault localization techniques and compare them. Each program has a correct version as well as incorrect versions. All the programs in this test suite are written in C. Similar to other studies we also downloaded all the test cases from the web site [4]. Some of the test cases like version 10 of "print-tokens" and version 32 of "replace", version 9 of "Schedule2" as they were the same as the original version and therfore had no failed test cases, so they were not the part of our experiments. Similarly we also discarded some of the other test cases like version 4 and version 6 of "print_tokens" where the faults were in the header files instead of the C files. Similar observations were are mentioned in other studies as well [14]. Table 3 shows the summary of the programs. For each subject it includes the name of the program, a brief

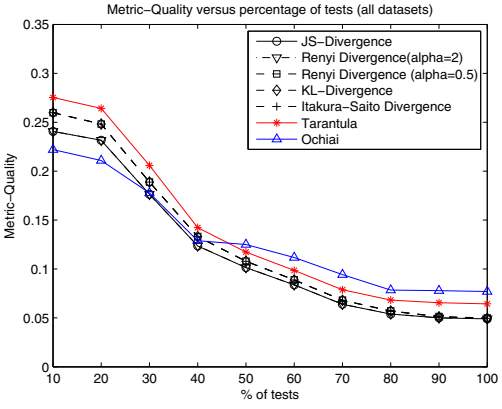

Fig. 6. The metric quality shows results for different metrics

description, the number of faulty versions, Lines of code (LOC), number of executable lines (statements), and the number of test-cases.

7.2 Results

Fig. 5 shows the plot between percentage of faulty versions versus percentage of code that needs to be examined. We find that the latent divergence performs better than other methods in general. We observe that NN/Binary [18] performs the worst while results of Tarantula and SOBER [22] are comparable. Out of the existing methods the Ochiai method seem to reasonably well. When we compare our results to the existing methods we find our method does much better Tarantula, SOBER, and CT. Even though Ochiai performs well compared to our method for only 42% of the fault versions while our method does it 33%, but just by examining 10% more code our method performs better that Ochiai. We are able to pin-point faults for 90% of the fault versions just by examining 20% of the code. We can practically cover more than 95% of the fault versions by just examining only 30% of the code the Ochiai method can only match upto 88% of the fault versions. The results obtained by using latent divergence using KL-divergence is more or less similar to all other latent divergences using different types of Renyi entropies at $\alpha = 2$ and $\alpha = 0.5$ and others.

Refer back to Eqn.(23). From the equations it is clear that the value of the ϕ should be less when most of the lines are ranked low and only few lines line are ranked high. We also observe the measure Metric-Quality (ϕ) in Fig.6 average for all programs (we also sometimes call it datasets). As shown in the plot as the number of tests are increased from 10% to 100% we can see that ϕ approaches a lower value closer to 0. The closer it is to 0, it implies that ranks of non related lines of statements are much lower and can be discard from further examination. We note latent divergence methods do better in ranking only few of the relevant lines or statements. Note that both Ochiai and Tarantula metrics

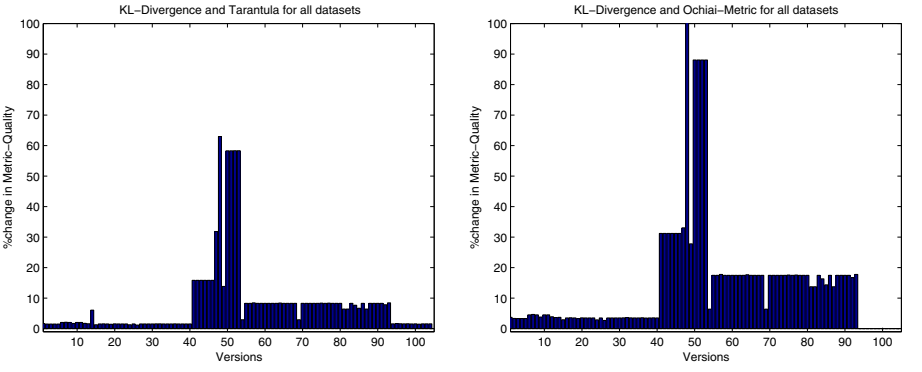

Fig. 7. (1) The percentage of the metric quality is compared between latent divergence based on KL-Divergence and Tarantula for all different program versions. (2)The percentage of the metric quality is compared between latent divergence based on KL-Divergence and Ochiai for all different program versions.

ranks of lines (statements) much higher compared to the ranks assigned by the latent divergence method on the same lines, and that may lead the programmer to wrong direction. Therefore the value of ϕ for latent divergence is naturally quite less than ϕs for the Ochiai method and the Tarantula method. Furthermore, Fig.7 shows percentage change in metric quality for each program separately. For some program versions the both Ochiai metric and Tarantula perform quite poorly. The %-change of metric-quality values shown in the Fig.7(1) and Fig.7(2) confirm trend observed in Fig.6.

8 Conclusions

In this paper we introduced a novel framework for spectrum-based fault localization using latent divergence, a novel concept based on probabilistic divergences. We show that it is feasible to use a family of latent divergences to accurately identify the lines of code that are faulty. Our experimental results show that our technique performs better than existing methods.

Acknowledgements. Khurshid's work was funded in part by the NSF under Grant Nos. IIS-0438967, and CCF-0845628, and AFOSR grant FA9550-09-1-0351.

References

1. Abreu, R., Zoeteweij, P., Golsteijn, R., van Gemund, A.J.C.: A practical evaluation of spectrum-based fault localization. Journal of Systems and Software 82(11), 1780–1792 (2009)
2. Ali, S.M., Silvey, S.D.: A General Class of Coefficients of Divergence of One Distribution from Another. J. Roy. Statist. Soc. Ser. B 28, 131–142 (1966)

3. Agrawal, H., Horgan, J.R.: Dynamic program slicing. In: Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation, PLDI 1990, pp. 246–256. ACM, New York (1990)
4. Aristole, http://www-static.cc.gatech.edu/aristole/tools/subjects
5. Binkley, D., Harman, M.: A survey of empirical results on program slicing. Advances in Computers 62, 106–179 (2004)
6. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (March 2004)
7. Bregman, L.M.: The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. USSR Computational Mathematics and Mathematical Physics 7(1-2), 200–217 (1967)
8. Coxeter, H.S.M., Greitzer, S.L.: Geometry Revisited (Mathematical Association of America Textbooks, 1st edn. The Mathematical Association of America (1967)
9. Csiszar, I.: On the computation of rate distortion functions. IEEE Transactions of Information Theory IT-20(1), 122–124 (1974)
10. Itakura, F., Saito, S.: An analysis-synthesis telephony based on maximum likelihood method. In: Proc. Int. Cong. Acoust., vol. c-5-5, pp. c17–c20 (1968)
11. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182 (2003)
12. Hutchins, M., Foster, H., Goradia, T., Ostrand, T.: Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In: Proceedings of the 16th International Conference on Software Engineering, ICSE 1994, pp. 191–200. IEEE Computer Society Press, Los Alamitos (1994)
13. Jiang, L., Su, Z.: Context-aware statistical debugging: from bug predictors to faulty control flow paths. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, ASE 2007, pp. 184–193. ACM, New York (2007)
14. Jones, J.A., Harrold, M.J.: Empirical evaluation of the tarantula automatic fault-localization technique. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005, pp. 273–282. ACM, New York (2005)
15. Liblit, B., Naik, M., Zheng, A.X., Aiken, A., Jordan, M.I.: Scalable statistical bug isolation. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2005, pp. 15–26. ACM, New York (2005)
16. Ochiai, A.: Zoogeographic studies on the soleoid fishes found in japan and its neighbouring regions. Bull. Jpn. Soc. Sci. Fish 22, 526–530 (1957)
17. Pekalska, E., Duin, R.P.W.: The Dissimilarity Representation for Pattern Recognition: Foundations And Applications. World Scientific Publishing Co., Inc., River Edge (2005)
18. Renieris, M., Reiss, S.P.: Fault localization with nearest neighbor queries. In: International Conference on Automated Software Engineering, p. 30 (2003)
19. Renyi, A.: On measures of information and entropy. In: Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability, pp. 547–561 (1961)
20. Steiner, J.: Einige geometrische Betrachtungen. Journal für die reine und angewandte Mathematik 1, 161–184 (1826)
21. Eric Wong, W., Debroy, V., Choi, B.: A family of code coverage-based heuristics for effective fault localization. J. Syst. Softw. 83, 188–208 (2010)
22. Liu, C., Yan, X., Fei, L., Han, J., Midkiff, S.P.: SOBER: statistical model-based bug localization. In: ESEC/FSE-13: Proceedings of the 10th European Software Engineering, Lisbon, Portugal, pp. 286–295 (2005)