Assessing Test Adequacy for Black-Box Systems without Specifications

Neil Walkinshaw

Department of Computer Science,
The University of Leicester
nw91@le.ac.uk

Abstract. Testing a black-box system without recourse to a specification is difficult, because there is no basis for estimating how many tests will be required, or to assess how complete a given test set is. Several researchers have noted that there is a duality between these testing problems and the problem of inductive inference (learning a model of a hidden system from a given set of examples). It is impossible to tell how many examples will be required to infer an accurate model, and there is no basis for telling how complete a given set of examples is. These issues have been addressed in the domain of inductive inference by developing statistical techniques, where the accuracy of an inferred model is subject to a tolerable degree of error. This paper explores the application of these techniques to assess test sets of black-box systems. It shows how they can be used to reason in a statistically justified manner about the number of tests required to fully exercise a system without a specification, and how to provide a valid adequacy measure for black-box test sets in an applied context.

1 Introduction

When do we know that a test set is adequate? How do we know that it is sufficiently rigorous for its execution to highlight the presence of any faults? If it is not adequate, how many more tests will we need to generate to achieve a requisite level of adequacy? These questions are fundamental to software testing.

Although numerous approaches are routinely used to assess test adequacy (e.g. code or model coverage), these have significant drawbacks. Code-based coverage has been shown to be an unconvincing fault predictor (c.f. work by Nagappan *et al.* [16]). Model-based coverage on the other hand makes the restrictive assumption that there exists a complete and up-to-date model of the system in question.

Over the past 30 years, a different approach to test adequacy has emerged that attempts to circumvent the weaknesses of traditional techniques. This approach exploits an intuitive relationship between the seemingly unrelated fields of inductive inference and software testing. The idea is to treat the two approaches as two sides of the same coin; both are dealing with a system that is unknown; testing elicits behaviour, and inductive inference reasons about its

B. Wolff and F. Zaidi (Eds.): ICTSS 2011, LNCS 7019, pp. 209–224, 2011.

behaviour by inferring models. From the perspective of test adequacy, there is a direct link between the accuracy of an inferred model and the adequacy of the test set that was used to infer it [32,31]. If a model can be shown to be accurate, the underlying test set evidently exercises the system in a sufficiently extensive manner.

There has been a recent resurgence in techniques that expolit this relationship [3,5,6,8,12,17,19,22,23,26,28,29,30] by inferring models from test sets, and in some cases using these models to elicit further test cases. However, these techniques tend to suffer from two problems: (1) there is no means of predicting how many tests would be required to arrive at an adequate test set and (2) given a partial test set, there is no basis for gauging how close it is to being adequate.

Problems that are analogous to these have been the subject of much research in the context of inductive inference [4,9,24]. These techniques, which are largely based on probabilistic reasoning, are especially interesting from a testing perspective because they offer potential solutions to these testing problems. This was the subject of a reasonably concentrated amount of research in the eighties and nineties [6,20,21,30,32,31], but has not been revisited in the light of the aforementioned surge in popularity of learning-based testing techniques.

This paper investigates the application of these techniques in a realistic testing context. The key contributions are as follows:

1. An implementation of Valiant's PAC framework [24] in a testing context. This enables the probabilistic specification of what would be considered to be an adequate test set in terms of the accuracy of the model that is inferred from it.
2. The application of PAC-based probabilistic techniques [9,4] to estimate lower bounds on the number of tests required for a test set of a black-box SUT to be adequate.
   - An applied demonstration of how to apply these approaches to SUTs that may be modelled by Finite State Machines.
3. A practical demonstration of the use of the PAC framework in an applied setting to quantify the adequacy of test sets with respect to a small black-box simulator of an SSH client. The entire infrastructure used for experimentation have been made openly available.

Section 2 will present the background to combining inductive inference with testing. Section 3 will show how the PAC framework can be reinterpreted in a testing context. Section 4 will show how this can be used to estimate the required size of an adequate test set. Section 5 shows how the PAC framework can be used in a practical context to estimate the adequacy of existing test sets. Section 6 will discuss related work, and section 7 will present the conclusions and discuss future work.

2 Background

2.1 The Setting

This paper considers a setting where the SUT is a black-box, but where there is no usable specification to generate tests from. In this context, a test case is simply an input to the SUT without an expected output. An *adequate* test set [30,31,32] will exercise every essential element of functionality in the system, and in doing so trigger any obvious faults such as a crash or an uncaught exception. This setting is realistic. The source code of a system, even if it is available, is only effective to a limited extent when as a basis for test set generation[16]. Although there are several sophisticated model-based testing techniques [13], developers rarely produce and maintain models that are sufficiently accurate and up-to-date to serve as a suitable basis for test generation.

The task of generating an adequate test set in this setting is seemingly impossible. Without a specification or source code there is no means by which to assess how complete the test set is. There is also no coverage-like metric to serve as a basis for homing-in on an adequate test set.

2.2 Testing with Inductive Inference

Over the past thirty years one approach has emerged that can (at least in principle) assess test sets in the above setting. Instead of generating a test set in a single step and subsequently executing it, the idea is to generate test sets by experimentation. The outputs produced by an initial test set are observed and are used to infer a hypothetical model of system behaviour. Depending on the approach, this may then be used to drive the generation of further test sets, or to assess the adequacy of the original test set by somehow comparing the model with the SUT.

Inductive inference is a means of reasoning about a black-box SUT in terms of its observable behaviour. If a test set is comprehensive enough to enable the inference of an accurate model, then it can be deemed to be adequate [30,31]. The relationship between inductive inference and software testing was first explored by Weyuker in 1983 [30]. Since then a large number of techniques have been developed that adopt different types of model inference. Initially, Weyuker's work and subsequent work by Bergadano *et al.* [3,30] focussed on synthesised programs. Since then however, similar approaches have been based upon Artificial Neural Nets [12,22], invariants [8], decision trees [5] and deterministic finite state automata [2,19,23,26,28,29].

2.3 Practical Problems in Establishing Test Adequacy

The use of inductive inference provides a plausible method for assessing the adequacy of test sets in a meaningful way (i.e. with respect to the behaviour they elicit). However, from a practical point of view, there remain two important barriers to its widespread use:

1. **Predicting expense:** There is no reliable basis for estimating how expensive the testing process will be, i.e. how many tests will be required to produce an adequate test set. This is a fundamental testing problem and is not restricted to testing techniques that incorporate inductive inference. In the context of testing techniques that use inductive inference, it is akin to stating that it is not known how many examples will be required to infer an accurate model.
2. **Quantifying adequacy:** Current testing approaches that revolve around inductive inference implicitly assume that an inferred model must be accurate before the test set can be considered adequate. Their feedback is binary: adequate or inadequate. This is impractical for two reasons. Firstly, there is no feedback to provide any insights about how close the test set is to being adequate, or determining whether one test set is better than an other. Secondly, most inductive inference algorithms are prone to making mistakes and can at best infer a model that is approximate even if the test set itself is adequate. However, there is no way to account for this by allowing for a given degree of error.

3 Inductive Inference and Testing in a Probably Approximately Correct Setting

In the context of machine learning, the area that seeks to address such problems is generally referred to as *Computational Learning Theory* (also *Statistical Learning Theory*). Given the widely acknowledged link between inductive inference and testing, it seems intuitive that some of the Computational Learning Theory principles that have been successfully applied in inductive inference should be readily applicable in a testing context. This section sets the foundations for this by recoding a framework by Valiant [24], which has become widely known as the *Probably Approximately Correct* (PAC) framework, into the testing setting.

3.1 The PAC Framework

The PAC framework [24] describes a basic learning setting, where the key factors that determine the success of a learning outcome are characterised in probabilistic terms. As a consequence, if it can be shown that a specific type of learner fits this setting, important characteristics such as its accuracy and expense with respect to different sample sizes can be reasoned about probabilistically. The specific elements of the framework are illustrated here with respect to the example problem of learning a deterministic finite state machine from sample sequences (to save space, we presume the conventional definition and notation [27]). Much of the notation used here to describe the key PAC concepts stems from Mitchell's introduction to PAC [14].

The PAC setting assumes that there is some *instance space* X. As an example, if we are inferring a finite state machine with an alphabet Σ, X could be the set of all words in Σ^*. A *concept class* C is a set of concepts over X, so in our case case it the set of all deterministic finite state machines that can accept and reject words in X. A *concept* $c \subset X$ corresponds to a specific target within C to be inferred (in our case it is the finite state machine that accepts a specific subset of words in Σ^*). Given some element x (in our case a word), $c(x) = 0$ or 1, depending on whether it belongs to the target concept. It is assumed that there is some selection procedure $EX(c, \mathcal{D})$ that randomly selects elements in X following some static distribution $\mathcal{D}$ (we do not need to know this distribution, but it must not change).

The basic learning scenario is that some learner is given a set of examples as selected by $EX(c, D)$. After a while it will produce a hypothesis h. The error rate of h subject to distribution $\mathcal{D}$ ($error_{\mathcal{D}}(h)$) can be established with respect to a further 'test' sample from $EX(c, \mathcal{D})$. This represents the probability that h will misclassify one of the test samples, i.e. $error_{\mathcal{D}}(h) \equiv Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$.

In most practical circumstances, a learner that has to guess a model given only a finite set of samples is susceptible to making a mistake. Furthermore, given that the samples are selected randomly, its performance might not always be consistent; certain input samples could happen to suffice for it to arrive at an accurate model, whereas others could miss out the crucial information required for it to do so. To account for this, the PAC framework enables us to explicitly specify a limit on (a) the extent to which an inferred model is allowed to be erroneous to still be considered approximately accurate, and (b) the probability with which it will infer an approximate model. The error parameter ϵ that puts an upper limit on the probability that an inferred model may mis-classify a given input. The δ parameter denotes an upper bound on the probability of a failure to infer a model (within the error bounds).

3.2 A PAC-Compatible Testing Framework

Figure 3.2 shows how the inductive inference and testing processes can fit into the PAC framework [31,26]. The arcs are numbered to indicate the flow of events. The test generator produces tests according to some fixed distribution $\mathcal{D}$ that are executed on the SUT c. With respect to the conventional PAC framework they combine to perform the function of $EX(c, \mathcal{D})$.

The process starts with the generation of a test set A by the test generator (this is what we are assessing for adequacy). These are executed on the SUT, the executions are recorded and supplied to the inference tool. This infers a hypothetical test oracle. Now, the test generator supplies a further test set B, and the user supplies some acceptable error bounds ϵ and δ. The observations of test set B are then compared against the expected observations from the model to compute $error_{\mathcal{D}}(h)$. If this is smaller than ϵ, the model inferred by test set A can be deemed to be *approximately accurate* (i.e. the test set can be deemed to be *approximately adequate*).

The δ parameter is of use if we want to make broader statements about the effectiveness of the combination of learner and test generator. By running multiple experiments, we can count the proportion of times that the test set is approximately adequate for the given SUT. If, over a number of experiments, this

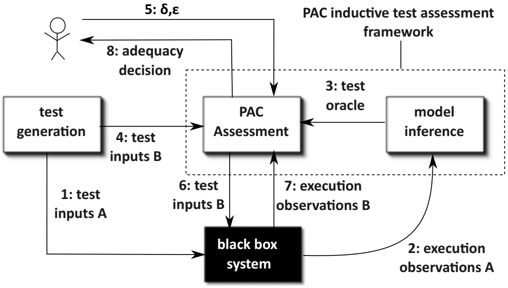

Fig. 1. Inductive testing with the PAC framework

proportion is greater than or equal to $1 - \delta$, it becomes possible to state that, in general, the test generator produces test sets that are *probably approximately adequate* (to paraphrase the term 'probably approximately correct', that would apply to the models inferred by the inference technique in a traditional PAC setting).

4 Estimating Test Set Size

Given that the SUT is a black-box, and that we can only reason about it by experimenting with it, it is seemingly impossible to ascertain a-priori how many test sets will be required to constitute an adequate test set. Surprisingly work that builds on the PAC framework *does* enable us to obtain a bound on the number of tests (if we make certain assumptions about the SUT, which are discussed later). By assuming that the set of test cases is selected randomly from some fixed distribution, it becomes possible to make a probabilistic argument the number of test cases required to arrive at a point where any model that is consistent with the test sets must be sufficiently accurate.

The approach relies on the ability to characterise the complexity of the learning task. In the context of PAC-learning Haussler [9] describes two approaches, each of which is based on a different characterisation of complexity. One of them assumes that it is possible to place an absolute bound the number of possible hypotheses that could be produced by a learner (known as the Version Space), whilst the other assumes that it is possible to place a bound on the internal complexity of the hypothesis space (known as the VC Dimension). These two approaches will be presented in this section, followed by a demonstration of how each of them can be applied to reason about the size of an adequate test set for a black-box SUT that could be modelled by a deterministic finite state machine.

4.1 Bounding Test Set Size with Version Spaces

The question of how many tests belong to an adequate test set is akin to the question of how many tests would be required to ensure that an accurate (within the limits of ϵ and δ) model can be inferred. To establish this, Haussler's Version-Space based approach [9] estimates a lower bound on the number of tests that would be required to ensure that all consistent hypotheses that could possibly be inferred from the test set fall within the acceptable error bounds.

To reason about the possible range of hypotheses, Haussler uses Mitchell's notion of *version spaces* [15]. In the testing context, a test set D consists of inputs x and their expected outputs $c(x)$. Version spaces are defined as follows (using the definition from Mitchell's book [14]):

$$VS_{H,D} = \{h \in H | (\forall \langle x, c(x) \rangle \in D)(h(x) = c(x))\}$$

Haussler defines the the version space as *ϵ-exhausted* if all of the hypotheses that can be constructed from D have an error below ϵ, with respect to any distribution $\mathcal{D}$. More formally [14]:

$$(\forall h \in VS_{H,D}) error_{\mathcal{D}}(h) < \epsilon$$

The number of elements in D that is required to ϵ-exhaust $VS_{H,D}$ is exponential or infinite in the worst case. However, given that we are using the PAC setting, this is not the case, we know that the elements in D are selected independently and at random, the number number of tests m that are required to ϵ-exhaust $VS_{H,D}$ is considerably improved [9]. If $VS_{H,D}$ is finite, it becomes possible to establish a lower bound on m. Assuming that set D is constructed by $m \geq 1$ independent, random test cases, he shows that the probability that $VS_{H,D}$ is *not* ϵ-exhausted is less than $|H|e^{-\epsilon m}$ (see Haussler's paper for the proof [9]). Within the PAC framework, this probability should be less than or equal to δ. This can be factored in to the above probability, and rearranged to impose a lower bound on m, the number of test cases that constitute D:

$$m \geq \frac{(ln|VS_{H,D}| + ln(1/\delta)}{\epsilon} \tag{1}$$

The number of required tests m grows linearly in $1/\epsilon$, it grows logarithmically with $1/\delta$, and it grows logarithmically in the size of $VS_{H,D}$ [9].

4.2 Bounding Test Set Size with the Vapnik-Chervonenkis Dimension

Depending on the SUT, it may be impossible to easily impose an upper limit on the size of $VS_{H,D}$ (e.g. the SUT could be a function that computes a real number). For this case, Haussler proposed an alternative approach to bound the test set size that does not rely on the size of $VS_{H,D}$, but uses a measure of complexity of H known as the Vapnik-Chervonenkis or VC dimension [25] (though this is not necessarily finite either).

To define the notion of a VC dimension, it is necessary to first introduce the notions of *dichotomies*, and *shattering* (see Haussler's paper [9] for details). If I is some subset of the instance space X, then an hypothesis $h \in H$ induces a *dichotomy* on I by dividing I into those examples that are classified as belonging to h, and those that are not. $\Pi_H(m)$ denotes the maximum number of dichotomies that can be induced by H on any set of m instances.

If H induces all possible 2^I dichotomies of I, then H *shatters* I. The VC dimension of H $VC(H)$ is the cardinality of the largest subset of X that is shattered by H. Equivalently, it is the largest m such that $\Pi_H(m) = 2^m$ [9].

Using a proof that is analogous to the one used in the version space approach, Haussler shows that the probability that $VS_{H,D}$ is not ϵ-exhausted is less than $2\Pi_H(2m)2^{-\epsilon m/2}$. From this, and by incorporating further results from Blumer *et al.* [4], it can be rearranged to yield the lower bound on m:

$$m \geq \frac{4log_2(2/\delta) + 8VC(H)log_2(13/\epsilon)}{\epsilon} \tag{2}$$

Mitchell [14] notes that this measure will often produce tighter bounds than the equivalent estimation using the Version Space approach. The bound m grows logarithmically in $1/\delta$, but grows log linear in $1/\epsilon$. As will be shown in section 4.3, the choice between the version space and the VC approach depends on the ability of the tester to characterise the complexity hypothesis space for the SUT.

4.3 Bounding Test Sets for SUTs That Are Finite State Machines

This subsection demonstrates the two above techniques, showing how they can estimate the number of tests that are required to test a black-box SUT. For the purposes of illustration, it is assumed that H is the range of deterministic finite state machines over some known alphabet. It is important to note that the use of state machines is merely for the purpose of illustration – Haussler's approach can be applied to a broad range of other representations.

As mentioned previously, the choice between the Version Space approach and the VC dimension approach depends upon the ability to characterise the complexity of H in an appropriate way. In practice the size of $VS_{H,D}$ is infinite for any DFA inference technique, because there are an infinite number of possible DFAs that are consistent with a given test set. The VC dimension for DFAs is also infinite; for any subset of words in X it is possible to produce an exact hypothesis to shatter them, and the largest subset of X is infinite [10].

As a consequence, to ascertain a limit on the test set, it becomes necessary to make some assumptions about the DFA, or the context in which it will be tested. The remainder of this section shows how such assumptions can be used to make the two techniques possible. Specifically, the Version Space approach can be used by imposing a bound on the length of the test cases (implying a bound on the depth of the DFA). Alternatively the VC dimension approach can be used by imposing an upper limit on the number of states in the DFA.

Using the Version Space approach by bounding test case length. For DFA inference, the relationship between the set of samples D and the version space $VS_{H,D}$ was described by Dupont [7]. He showed how the version space can be interpreted as a lattice where the most specific element is a prefix tree automaton (PTA) (a tree-shaped minimal DFA that exactly represents the sample D [7,28,29]), and the most general element is the universal DFA that accepts every element in the DFA alphabet Σ. The size of this version space, which is what we are interested here, is infinite if the depth of the PTA is unrestricted.

However, if the length of the test cases is limited to a chosen length n and the size of the alphabet is denoted σ, the maximum size of a PTA can be computed as: $\sum_{i=0}^{n} \sigma^i$.

In Dupont's lattice version space, any hypothesis $h \in H$ corresponds to a particular partition of the set of states in the PTA (corresponding to the merging of states into their respective equivalence classes). Thus, the size of $VS_{H,D}$ is bounded by the number of possible set partitions of a set the size max – the Bell number of max.

The extremely rapid growth of this number limits the use of the version space approach in this finite state machine setting, and the example shown here is restricted to a very simple SUT. We consider a setting where the length of a test set is restricted to 7, and the size of the alphabet is 3. In this case, the maximum size of the PTA is 3,280 states[1]. The upper bound on the number of DFAs that can be generated from such a PTA as computed by the Bell number of 3280 is approximately $1.5 * 10^{7722}$.

Now the task for the tester is to decide a realistic error margin for the assessment of the test adequacy. To simply state that the test set should always be sufficiently comprehensive to produce an exact model is unrealistic. In our example the tester might consider it sufficient if the model inferred from the test set has an error ≤ 0.1, and that this should happen with a probability of 90%. In other words, $\epsilon = 0.1$ and $\delta = 0.1$ (calculated by $1 - 0.9$). This now allows us to apply Haussler's version-space estimation (see equation 1):

$$\begin{aligned} m &\geq \frac{(ln|VS_{H,D}| + ln(1/\delta)}{\epsilon} \\ \approx m &\geq \frac{(ln(1.5 * 10^{7722}) + ln(1/0.1)}{0.1} \\ \approx m &\geq \frac{17,778.67977 + 2.303}{0.1} \\ \approx m &\geq 177,809.8277 \end{aligned} \tag{3}$$

Taking these values at face value, the task of constructing an adequate test set of this size for such a relatively simple scenario is unrealistic. It is however important to bear in mind the proportions of the problem space. From a possible $1.5 * 10^{7722}$ hypotheses, it is possible to assert that a consistent learner will produce an accurate hypothesis from a 177,810 tests – i.e. to statistically justify this test set will be adequate.

[1] A small Erlang module with all of the routines used to compute the results in this paper is available
`http://www.cs.le.ac.uk/people/nwalkinshaw/Files/ictss_code.zip`

Of course, considering the relative simplicity of the system in question, this is very large number of tests (despite the vast size of the hypothesis space). There are two things that one has to bear in mind when interpreting this number. Firstly, it is a conservative worst-case estimate. It does not take any failed / impossible tests into account (which would eliminate vast numbers of false hypotheses from an inference standpoint), and does not place any expectations on the learner to do anything with the input data other than be consistent (i.e. not to produce an hypothesis that contradicts the input data). In practice, negative sequences merge out a vast number of invalid merges, and inference techniques often use heuristics [11] to efficiently home-in on the correct merges. Ultimately a justifiable upper bound, even if it is too large, is better than no bound at all, because it provides at least a rough guide for the complexity of the SUT, and the associated testing effort.

Using the VC dimension approach by bounding the number of states in the SUT. In certain cases, it might not be possible to bound $VS_{H,D}$. In the previous setting, any larger alphabets or test lengths would become too large to compute in a practical way, and it might simply be impossible to guess an upper bound on the maximum length of a test case anyway. The VC dimension alternative is useful because it does not rely on a finite version space, but instead provides an internal measure of the complexity of a potentially infinite hypothesis space.

Unfortunately, depending on the representation, it is not always possible to calculate a finite VC dimension. For arbitrary DFAs the VC dimension is infinite and can only be made finite by making assumptions about its maximum number of states – if this is n states, the VC-dimension is bounded by $n \log_2 n$ [10]. Thus, in this case, the choice between version spaces and VC-dimension approaches is determined by the nature of any additional knowledge of the DFA.

Estimating a suitable number of states n relies on the intuition of the tester, from their prior knowledge of the SUT. For this example, let us guess that the SUT contains at most 300 states. The VC dimension is thus bounded by $300 * \log_2(300) = 2,468.65$.

This enables us to substitute for equation 2. As in the initial case for the version space example, let us assume that $\epsilon = 0.1$ and $\delta = 0.1$:

$$\begin{aligned}
m &\geq \frac{4log_2(2/\delta) + 8VC(H)log_2(13/\epsilon)}{\epsilon} \\
\approx m &\geq \frac{4log_2(2/0.1) + 8 * 2,468.65 * log_2(13/0.1)}{0.1} \\
\approx m &\geq 1,387,032
\end{aligned} \tag{4}$$

As in the previous version-space approach, this number is a conservative worst-case estimation. It fails to take any heuristic capabilities of the inference technique into account. As previously, depending on the circumstances it might be possible to take this added efficiency into account by increasing the value of ϵ. If, using the same rationale, ϵ is increased to 0.4, the result is a much reduced bound of $m \geq 248,012$.

It is important to bear in mind that it does not make sense to compare the two approaches as presented here with respect to their estimated test sizes, because this would be comparing estimations for (potentially) completely different systems. Nonetheless, with respect to DFAs, if the number of states can be bounded it is better to use the VC-dimension approach, because it is easier to compute an estimate for more complex systems and tends to compute a much lower bound than the version-space approach (this latter fact applies to all representations, not just DFAs [9,14]).

5 Using the PAC Setting to Empirically Assess Test Sets

The ability to predict the sizes of test sets is only one side of the benefit of using the PAC framework for testing, and has been already explored to some extent in previous literature [31,21,20]. From an empirical aspect, the framework is equally valuable, because it presents us with a basis for making statistically justified measurements of test set adequacy for black-box SUTs, by assessing the performance of the inferred models. This section presents a practical example of this. It not only shows the value of being able to assess test sets, but also highlights an important practical consideration that can lead to problems of accuracy when computing the lower bounds for test size computed by the techniques presented in the previous section.

Current techniques that combine testing with model inference make a binary decision; a test set is adequate if it leads to an exactly accurate model (as assessed within the limits of some model-based testing technique), and inadequate otherwise. This is problematic because there is no basis for homing in on an adequate test set. This section illustrates how the PAC testing framework (as shown in Figure 3.2) can be applied in a practical context to provide feedback about test set adequacy.

5.1 The SUT, and the Choice of Test Generation and Model Inference Techniques

The SUT in question simulates the behaviour of an SSH client, in terms of the FSM specification described by Poll *et al.* [18]. It accepts sequences of instructions as specified, but will throw an “unexpected input” exception if given a sequence of inputs that is not part of the specification. The system is written in Erlang (implemented using the `gen_fsm` behavioural pattern – available with the source code provided with this paper).

Let us assume that we have a small sample of 10 test scenarios that execute some of the expected behaviour of the system. Because these only exercise a tiny (albeit functionally significant) fraction of program behaviour, it is necessary to substantially bulk up the test set if we want it to be adequate. Given that we are presuming no further domain knowledge about the system, the rest of the tests will have to be generated randomly. For this we use a random generator that produces a set of unique random sequences from the given alphabet up to a certain length (for this example we choose 13 to be the maximum test case length, and choose the length of each test case randomly).

> The problem with any resulting test set, no matter how large, is that we do not know how *adequate* it is.

This is where the PAC framework can offer a solution. By generating two non-intersecting test sets, using one to infer a model, and the other to assess its accuracy, it is possible to obtain an insight into how adequate the first test set is.

To infer a model from the tests we choose Price's EDSM blue-fringe state merging algorithm [11] – until recently the most accurate algorithm for inferring state machines from arbitrary examples. We use the openly available Ruby implementation by Bernard Lambeau that was developed as a baseline for the StaMInA inference competition [27].

5.2 Application of the PAC Framework

We start by generating test sets A and B (see Figure 3.2). The PAC framework assumes that these are drawn randomly from the same distribution, but must not overlap. To ensure that this is the case a large set of unique random test cases is generated, and the contents of A and B are selected at random from this superset. Set A will be used to train the model, and will be the test set that we assess, and test set B will be the set with which we assess the accuracy of the model (and so the adequacy of A).

Due to time constraints, we terminate the generation algorithm after an hour. In that time sets A and B have been populated with 42,410 tests each. Now the tests and their respective outcomes from set A are used to infer a model using the StaMInA tool. The model is then used to predict the outcomes for test set B. The error rate can then tell us how adequate test set A. If we use the conventional definition of $error_{\mathcal{D}}(h)$ to compute the error, we end up with an adequacy assessment of 99.99%.

Upon closer inspection, splitting the test cases up into true or false positives and negatives shows that this figure has to be interpreted with care. Out of the 42,410 test cases, 42,397 tests are true negatives, five tests are true positives, five tests are false positives and three are false negatives. Ultimately, the fact that there is such a high overlap between sets A and B says more about distribution from wich they were sampled than it does about the SUT. PAC-learning assumes that the distributions A and B reflect the routine behaviour of the system in question, in which case this measure of overlap is appropriate.

In a testing context, this measure is not particularly helpful. A randomly generated test case will generate arbitrary distributions of test cases that do not evenly represent the input domain of the SUT, and may lead to heavily skewed error rates. To account for this, we use the Balanced Classification Rate (BCR) [27][2], which balances the ability of the inferred model to reject false negatives

[2] This measure is commonly used in machine learning, and should not be attributed to the author, but is described in this paper with respect to DFA inference.

against its ability to reject false positives: $BCR_{\mathcal{D}}(h) = \frac{1}{2}(TP/(TP + FN)) + (TN/(TN+FP))$. If we apply this to calculate the adequacy of test set A above, we obtain a more balanced test adequacy assessment of 0.8124.

5.3 Discussion

This section has demonstrated how to assess the adequacy of a test set for a black-box system. However, in drawing the distinction between the different measures for calculating classification error ($error_{\mathcal{D}}(h)$ and $bcr_{\mathcal{D}}(h)$), it highlights an important caveat for interpreting the test-size estimations produced by the techniques in section 4. These predictions only apply when the test set can be reliably assessed using the $error_{\mathcal{D}}(h)$ measure, i.e. when the distribution of tests is roughly balanced between valid and invalid cases.

If this is not the case, the estimated lower bound on the number of required test cases will probably be a significant underestimation. Given that the SSH implementation has 19 distinct states [18], this can be illustrated with the VC-dimension approach. For 19 states we obtain a VC-dimension of 80.71. We might guess a conservative ϵ value of 0.1, and choose a δ value of 0.1. Substituting into equation 2, this gives us an estimated lower bound of 45,515 test cases.

This happens to be relatively close to the number of tests we generated in section 5.2. Had we used the conventional measure for $error_{\mathcal{D}}(h)$, this would certainly be accurate. However, given that out of 42,410 random test-cases only 8 of these produce valid outputs from the SUT, it is clear that a much larger number of random test cases would be required to fully exercise the SUT in terms of its valid behaviour as well (and so reach a high level of adequacy with respect to the BCR). Future work (see section 7) will elaborate the techniques from section 4 to develop more accurate lower bounds that take into account the the balance between valid and invalid test cases.

6 Related Work

As discussed in the Background section, there has been much work on relating the fields of machine learning and software testing, and several relevant references are included in this paper. Due to limited space, this section shall focus on the more specific topic of the use of probabilistic techniques to reason about test sets for black-box systems without specifications.

The idea of testing "to a confidence level" by joining the fields of Inductive Inference and Testing was first raised by Cherniavsky and Smith in 1987 [6] (although their work was primarily concerned with learning exact models). The subject was subsequently explored by Zhu *et al.* [32,31]. They used the PAC framework as a theoretical basis to reinterpret and justify a set of fundamental test adequacy axioms they had proposed in earlier work. They also suggest using an alternative to the VC-dimension and version-space estimation approaches to predict the necessary size of a test set (Haussler's Pseudo-dimension) though given that the work is theoretical in nature, there is no suggestion of how this might be used in a practical context (e.g. how to compute the pseudo-dimension for a given type of black box SUT).

The combination of PAC learning and testing was the subject of a substantial amount of work by Romanik *et al.*. By adopting the PAC framework, they proposed the notion of *approximate testing* [20]. They show how familiar machine learning concepts such as the VC-dimension can be used to reason about the general (approximate) testability of particular classes of program, with a particular interest in reasoning about certain classes that are *un-testable.* In subsequent work, Romanik [21] considers the relationship between the internal complexity of a program (i.e. its source code branching) and its testability, and proposes an extension of the VC-dimension (the VCP-dimension) to measure this.

Although Zhu and Romanik made pioneering theoretical contributions to the research on combining the two fields, it was perhaps the relative primitiveness of machine learning techniques at the time that prevented the practical application. It is only recently that inference techniques have developed the capabilities to infer (approximately) accurate models of software systems. To the best knowledge of the author, this paper the first work that attempts to experiment with the combination of PAC-learning and testing in a practical sense for the sake of assessing test sets.

Many of the recent testing techniques to involve machine learning (c.f. work by Raffelt *et al.* and Shahbaz *et al.* [19,23] are based on Angluin's L^* algorithm [1]. In her paper, she discusses how her algorithm could be adapted to suit a PAC setting. To the best of the author's knowledge, this has not yet been implemented in a testing context, but suggests that it would in principle be straightforward to adapt these existing testing techniques to apply the PAC-based principles that have been discussed in this paper.

7 Conclusions and Future Work

The challenge of producing a comprehensive test set for a black-box system without recourse to a specification is seemingly impossible. There is no obvious basis for determining whether a test set is adequate, and for identifying a candidate set of test cases from a potentially infinite set of potentials.

Against this backdrop, machine learning is a particularly interesting discipline, because it provides a wealth of techniques to reason in a systematic way about hidden systems by way of experimentation. Valiant's PAC framework provides a useful formal basis for this, and has formed the basis for a limited amount of theoretical work on software testing [31,21,20]. Specifically, the PAC framework can be used to reason about the accuracy of the inferred model which, in turn, provides feedback about the adequacy of the test set that was used to infer it. Furthermore, the use of the PAC framework enables the estimation of how many test sets might be required to produce an adequate test set.

In the light of the recent emergence of numerous testing techniques that are founded on specific machine learning techniques [2,5,8,12,17,19,22,23,26,28,29], this paper has sought to investigate the practical application use of the PAC framework. The paper shows how the PAC framework can be applied in practice. it shows how test set sizes can be predicted, and how the framework can be used to obtain a statistically valid assessment of test adequacy in practice.

The practical example in this paper has highlighted one problem of applying the PAC framework in a testing context. It is assumed that there is a rough balance between valid and invalid test cases, and the conventional measure of error can be misleading when this is not the case (which is typical in a testing context). Future work will attempt to adapt Haussler's predictions [9], to produce more accurate predictions for typical random testing situations, where the test set is not balanced.

Acknowledgements. Much of the background material that relates inductive inference to software testing was influenced by discussions with Gordon Fraser at Saarland University in Saarbrücken.

References

1. Angluin, D.: learning regular sets from queries and counterexamples. Information and Computation 75, 87–106 (1987)
2. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005)
3. Bergadano, F., Gunetti, D.: Testing by means of inductive program learning. ACM Transactions on Software Engineering and Methodology 5(2), 119–145 (1996)
4. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the vapnik-chervonenkis dimension. Journal of the ACM 36, 929–965 (1989)
5. Briand, L., Labiche, Y., Bawar, Z., Spido, N.: Using machine learning to refine category-partition test specifications and test suites. Information and Software Technology 51, 1551–1564 (2009)
6. Cherniavsky, J., Smith, C.: A recursion theoretic approach to program testing. IEEE Transactions on Software Engineering 13 (1987)
7. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? (1994)
8. Ghani, K., Clark, J.: Strengthening inferred specifications using search based testing. In: International Conference on Software Testing Workshops (ICSTW). IEEE, Los Alamitos (2008)
9. Haussler, D.: Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. Artificial Intelligence 36, 177–221 (1988)
10. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010)
11. Lang, K., Pearlmutter, B., Price, R.: Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–12. Springer, Heidelberg (1998)
12. Last, M.: Data mining for software testing. In: The Data Mining and Knowledge Discovery Handbook, pp. 1239–1248. Springer, Heidelberg (2005)
13. Lee, D., Yannakakis, M.: Principles and Methods of Testing Finite State Machines - A Survey. Proceedings of the IEEE 84, 1090–1126 (1996)

14. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
15. Mitchell, T.: Generalization as search. Artificial Intelligence 18(2), 203–226 (1982)
16. Nagappan, N., Murphy, B., Basili, V.: The influence of organizational structure on software quality: an empirical case study. In: International Conference on Software Engineering (ICSE), pp. 521–530. ACM, New York (2008)
17. Perkins, J., Ernst, M.: Efficient incremental algorithms for dynamic detection of likely invariants. SIGSOFT Software Engineering Notes 29, 23–32 (2004)
18. Poll, E., Schubert, A.: Verifying an implementation of ssh. In: Workshop on Issues of Theory of Security (WITS), pp. 164–177 (2007)
19. Raffelt, H., Steffen, B.: Learnlib: A library for automata learning and experimentation. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 377–380. Springer, Heidelberg (2006)
20. Romanik, K.: Approximate testing and its relationship to learning. Theoretical Computer Science 188(1-2), 175–194 (1997)
21. Romanik, K., Vitter, J.: Using Vapnik-Chervonenkis dimension to analyze the testing complexity of program segments. Information and Computation 128(2), 87–108 (1996)
22. Shahamiri, S., Kadira, W., Ibrahima, S., Hashim, S.: An automated framework for software test oracle. Information and Software Technology 53 (2011)
23. Shahbaz, M., Groz, R.: Inferring mealy machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 207–222. Springer, Heidelberg (2009)
24. Valiant, L.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)
25. Vapnik, V., Chervonenkis, A.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications 16(2), 264–280 (1971)
26. Walkinshaw, N.: The practical assessment of test sets with inductive inference techniques. In: Bottaci, L., Fraser, G. (eds.) TAIC PART 2010. LNCS, vol. 6303, pp. 165–172. Springer, Heidelberg (2010)
27. Walkinshaw, N., Bogdanov, K., Damas, C., Lambeau, B., Dupont, P.: A framework for the competitive evaluation of model inference techniques. In: Proceedings of the First International Workshop on Model Inference In Testing (MIIT), pp. 1–9. ACM, New York (2010)
28. Walkinshaw, N., Bogdanov, K., Derrick, J., Paris, J.: Increasing functional coverage by inductive testing: A case study. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 126–141. Springer, Heidelberg (2010)
29. Walkinshaw, N., Derrick, J., Guo, Q.: Iterative refinement of reverse-engineered models by model-based testing. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 305–320. Springer, Heidelberg (2009)
30. Weyuker, E.: Assessing test data adequacy through program inference. ACM Transactions on Programming Languages and Systems 5(4), 641–655 (1983)
31. Zhu, H.: A formal interpretation of software testing as inductive inference. Software Testing, Verification and Reliability 6(1), 3–31 (1996)
32. Zhu, H., Hall, P., May, J.: Inductive inference and software testing. Software Testing, Verification, and Reliability 2(2), 69–81 (1992)