Software Testing by Active Learning for Commercial Games

Gang Xiao
Finnegan Southey
Robert C. Holte
University of Alberta, Dept. of Computing Science

Dana Wilkinson
University of Waterloo, Dept. of Computer Science

Abstract

As software systems have become larger, exhaustive testing has become increasingly onerous. This has rendered statistical software testing and machine learning techniques increasingly attractive. Drawing from both of these, we present an active learning framework for blackbox software testing. The *active learning* approach samples input/output pairs from a blackbox and learns a model of the system's behaviour. This model is then used to select new inputs for sampling.

This framework has been developed in the context of commercial video games, complex virtual worlds with high-dimensional state spaces, too large for exhaustive testing. Beyond its correctness, developers need to evaluate the *gameplay* of a game, properties such as difficulty. We use the learned model not only to guide sampling but also to summarize the game's behaviour for the developer to evaluate. We present results from our *semi-automated gameplay analysis by machine learning* (SAGA-ML) tool applied to Electronics Arts' FIFA Soccer game.

Introduction

It is no news that software systems are very complex and difficult to debug. Because exhaustive testing or a proof of correctness for large systems is difficult or impossible, there has been increasing interest in *statistical software testing*. A secondary, and less focused, subject of study has been the use of machine learning for software testing, attempting to learn a model of the system's behaviour based on static or dynamic analysis.

We present an *active learning* framework for blackbox software testing that combines aspects of both of these broad approaches. In our approach, labelled examples are obtained by sampling from the space of initial system states and user actions, and then running the blackbox to obtain the labels. This training set is then used to learn a model of the system's behaviour. Once a model has been learned, it is used to determine where further sampling should occur, increasing the training set for the next iteration of model learning. This intelligent sampling strategy attempts to allocate limited testing resources effectively.

Contemporary commercial video games are an example of very complex software systems that have tight development cycles and high reliability standards because aftermarket patching is not always an option. Furthermore, beyond the standard software correctness requirements, they have a more nebulous goal of "enjoyability" that is impossible to specify or measure exactly. It is up to the designer to decide whether a game's behaviour is appropriate and collected data can only assist in the evaluation. In this context, machine learning serves a dual role where the learned model is used to summarize a game's behaviour for the developer and also to direct the sampling of additional points.

We will briefly review related software testing literature before turning to a description of the task facing game developers and our view of machine learning's role within that task. In order to make the research concrete, we present our case study, Electronic Arts' FIFA Soccer title. This game has been analyzed using our analysis tool, SAGA-ML (semi-automated gameplay analysis by machine learning), which is game independent and treats the game as a black box.

Results are presented to the developer using the SoccerViz visualization tool, a game-specific component that displays the learned models in a more comprehensible format. Details of this specific implementation of our overall framework are explained, including the learning and active sampling approaches. Experimental results show, first and foremost, that the tool correctly identifies so-called "sweet spots" in the game (situations where it is too easy to score). We then compare the performance of a variety of active learning methods in a rule-learning context, including a rule-based sampling approach developed specifically for this task that offers robust performance. A further study demonstrates that active learning becomes more important as dimensionality increases. We conclude with remarks on the reception of this work by the industry and its potential for a greater role in game development and software testing generally.

Related Work

Substantial work in the software testing literature relates to our overall approach. Statistical software testing, often exemplified by the Cleanroom approach (Poore & Trammell 1996), uses sampling to test large input spaces, in some cases offering bounds on the probability of missed errors. Much of this work involves the construction of a statistical model of user input, such as a Markov chain (Whittaker & Thomason 1994), and using that to generate test cases.

Machine learning has also appeared in software testing, albeit in a more sporadic fashion, with the use of classification trees (Porter & Selby 1990)(Cheatham, Yoo, & Wahl 1995), logistic regression (Denaro, Morasca, & Pezzè 2002), and inductive logic programming (Bergadano & Gunetti 1996). On a related line, estimation of distribution algorithms (EDA) (Sagarna & Lozano 2005) and genetic algorithms (GA) (Wegener *et al.* 1997) have also been applied to software testing, combining sampling and implicit (GA) or explicit modelling (EDA) of a system's behaviour.

Closely related to our case study, genetic algorithms have been used for testing of the FIFA Soccer title (Chan *et al.* 2004), generating long sequences of actions that are likely to score. Some IBM research on coverage analysis uses a form of clustering and shares our emphasis on the visualizability of the learned model (Lachish *et al.* 2002). Finally, independently and in parallel with our own work, active learning has been proposed and tried as a software testing methodology (Bowring, Rehg, & Harrold 2004). Work of this kind, originating from the software testing research community, corroborates our belief that active learning has much to offer automated software testing.

The Gameplay Analysis Task

While we believe the active learning framework has much to offer software testing in general, we will specifically consider the task of *gameplay analysis* for commercial computer games, a problem which offers a unique set of challenges beyond the typical software testing requirements. It is difficult to characterize gameplay analysis precisely. This is largely due to the fact that it inevitably involves some human judgement. "Enjoyability" is essentially impossible to quantify. An integral but complex part of enjoyability is the gameplay offered and an important aspect of good gameplay is appropriate difficulty. A game that is too hard is frustrating, while too little challenge can be boring. In multi-player games it is important that the game be fair, offering no player an intrinsic advantage.

In the gameplay analysis task, the developer selects some *metrics* for evaluating gameplay (e.g. probability of winning, average time to win/lose, average resources consumed to win, etc.). The scope of the analysis is necessarily limited to a small part of the game, since even these small pieces may be expensive to evaluate. Sampling and machine learning are used to obtain a model of the game's behaviour, offering both a summary for the developer and predictions for unsampled regions of the space. It is up to the developer to examine this information and decide whether it is acceptable or whether changes are required. It is this interactive approach that leads us to call the process *semi-automated,* and why we emphasize the role of the learned model as a basis for visualization.

Electronic Arts' FIFA Soccer

We have used SAGA-ML to analyze scenarios in the Electronic Arts (EA) soccer game, FIFA'99. Games are generally too complex to be analyzed in their entirety. Therefore, it makes sense to analyze specific scenarios, especially when the game itself distinguishes between scenarios (e.g. different levels, minigames, and control modes). We have explored a small set of scenarios in FIFA, but here we will only consider the *corner kick* and the *shooter-goalie* scenarios.

In the corner kick scenario, the ball has rolled across the endline and was most recently touched by the defenders. It must be kicked back in from the corner by a member of the offense. This is a key opportunity for the player's teammates (positioned and controlled by the game AI) to score. The player must pick a spot for the ball to land when they kick. The success or failure of the player's teammates in scoring depends on where the ball lands. The specific metric we consider is a single number: the probability of scoring given that a single teammate touches the ball after the player kicks it (otherwise the ball might continue in play for some time before scoring). The initial game state is trivial in this case; it is only necessary that the game be in its special corner kick mode. The player action is the target for the ball, (x_k, y_k), so there is a two-dimensional space to explore.

In the shooter-goalie scenario, the player controls a shooter placed somewhere near the goal and shooting toward it, with the goalie placed to defend the goal. All other players are absent from the scenario. This tests the ability of the player to shoot and the goalie to block shots on goal, a critical part of the game. The metric is a single number, the probability of scoring from the player's kick. The initial game state of this scenario consists of the shooter's position, (x_s, y_s), and the goalie's position, (x_g, y_g), on the field. The shot is aimed automatically, so this scenario has a four-dimensional space.

Semi-Automated Gameplay Analysis

The overall architecture of the SAGA-ML approach is shown in Figure 1. The *game engine* is treated as a black box and SAGA-ML interacts with it through an *abstraction* layer. This layer is game-specific and translates game-specific data and function calls to an abstract state format used by SAGA-ML. The *sampler* component uses the abstraction layer to evaluate situations by running the game with an initial state and a sequence of actions, and then observing the outcome. The *learner* uses the data gathered by the sampler to construct a concise *model* (or *summary*) of the game's behaviour. The learner may then request more samples to refine its model. Together the sampler and learner form the active learning part of the system. Finally, the learned model is passed to the game-specific *visualizer* for the designer to evaluate. Note that the visualizer could use the game engine to render views for the developer. We will now discuss each of these components in greater detail using our FIFA scenarios to illustrate.

Abstraction

Games have far more variables than we can reasonably expect to interpret. When evaluating a game, we are interested in specific metrics (e.g. goals scored, time elapsed, etc.) but we must also specify the initial state (e.g. position, speed, etc.) and player actions (e.g. move, shoot, etc.) we want to

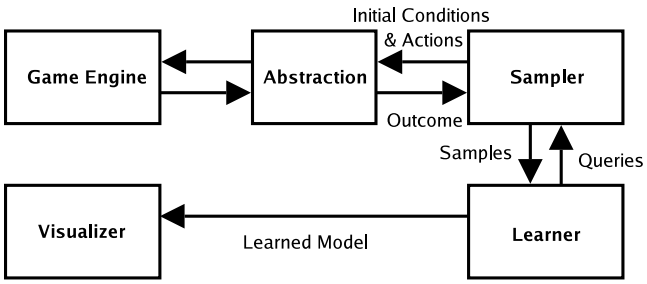

Figure 1: Architecture

sample. The abstraction layer transforms the raw *game variables* to form the corresponding *game state*, *player actions*, and *metrics* used by SAGA-ML. These are three vectors of numbers exchanged between SAGA-ML and the abstraction layer to control and observe the game.

Often, games store internal information that is optimized for rendering graphics or physics simulation, and the raw game variables may be inappropriate for directly modelling gameplay behaviour. For example, FIFA stores player positions as x-y coordinates on the field, but the actual outcome of a shot on goal is determined by the angle between shooter and goalie, the distance between them, and the relationship of the shooter to the goalposts. If the learner only sees the x-y coordinates, it may not be able to deduce the underlying rules used by the game engine or it may learn an overly complex model. Therefore, the abstraction layer also serves to interpret the raw game variables into useful features. For FIFA soccer, we have hand-crafted a set of features, chiefly converting from Cartesian to polar coordinates. In general, game developers will have a good understanding of what features are relevant to the game's behaviour so this transformation, while important, is similar to the code required to implement game behaviours in the first place.

Rule-Based Learning

For this project, we decided to use rule-based learning, chiefly because rules are more readily interpretable than many other machine learned representations. For example, in the shooter-goalie scenario, such a rule might be: IF the shooter is within 5 metres of the goalie AND the angle between shooter and goalie is between 30° and 40° AND the goalie is within 1 metre of the goal's centre THEN the probability of scoring is greater than 70%. Even so, large sets of rules are difficult to absorb, so a game-specific visualizer called SoccerViz has been constructed for displaying these rules in an intuitive fashion. Showing regions on a soccer field and arrows indicating movements, the developer can see the purport of the rules in a natural setting. The decision tree learner C4.5 (Quinlan 1994) was used since it is able to produce suitable rules but the architecture admits the use of any rule-based learner. We have conducted preliminary experiments using the rule learner SLIPPER (Cohen & Singer 1999) that we will not present here.

Active Learning

The literature on active learning offers many algorithms for deciding where to sample next, chiefly differing in the heuristics they use to identify "interesting" regions. We explored several approaches, including *uncertainty sampling* (Lewis & Gale), *Query by Committee* (QBC) (Seung, Opper, & Sompolinsky 1992) (more specifically, we implemented Query by Boosting and Query by Bagging (Abe & Mamitsuka 1998)), and Bootstrap-LV (Saar-Tsechansky & Provost 2004). Furthermore, we developed a new active learning technique specifically for rule-based learning systems which we have called *decision boundary refinement sampling*. Space does not allow us to describe all of these techniques, but we will elaborate on this new method.

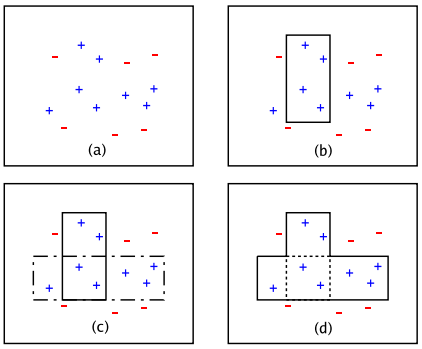

Figure 2: (a) Positive and negative samples in a 2-D space. (b) A single, positive rule learned from samples. (c) A second, overlapping positive rule learned from samples. (d) Overlapping rules joined into a single positive region (dashed lines are original rule boundaries).

Decision Boundary Refinement Sampling In a two-dimensional space, such as the corner kick scenario, the rules describe rectangles where the prediction is positive (true) or negative (false) (this is just an example, predictions could be more complex, e.g. probabilities). Figure 2 (a) shows the results of a set of samples. Figure 2 (b) shows a single, positive rule learned from these samples represented as a rectangle. Figure 2 (c) shows a second, overlapping rule learned from the same data. If there are several rules that overlap, but all agree in their prediction, then we can merge them together to form a *region* (as shown in Figure 2 (d)). This idea extends to higher-dimensional data, where the regions are composed of hyper-rectangles.

Decision boundary refinement sampling was developed specifically for rule learners. To ensure that a region's boundary is correct, new samples are randomly placed on both sides of the boundary within a small margin. The size of the margin is specified by a parameter. These samples help confirm or refute the boundary during the next iteration of learning. Sampling near the boundaries between overlapping, agreeing rules is wasteful, so we cannot simply sample at rule boundaries but must identify the regions to do effective sampling. Fortunately, it is straightforward to construct the regions and place the samples accordingly. A secondary mechanism used was *default rule sampling*. The *default rule* refers to those regions of the space not covered by the learned rules. Sample are placed uniformly in these regions. In the experiments here, 50% of the samples are placed by boundary refinement, and 50% by default rule sampling.

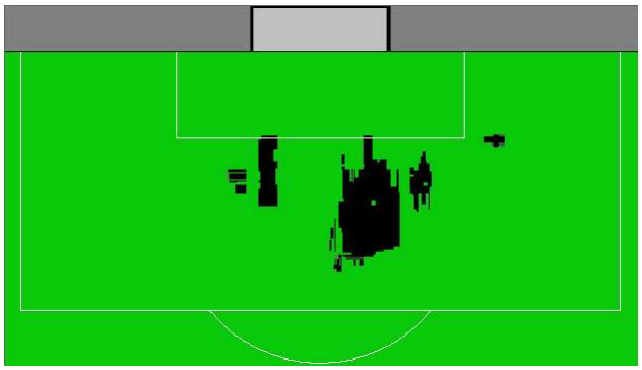

Figure 3: High probability scoring rules learned for corner kick.

Experimental Results

We present two main results. The first is the discovery of a significant sweet spot in the FIFA Soccer game corner kick scenario. Since this is the ultimate, practical objective of the work, this is a substantial achievement. The second is an accuracy study that more objectively assesses our approach.

Visualization of Sweet Spots in Corner Kick

Our visualization tool, SoccerViz, displays high scoring probability regions (>40%) in the corner kick scenario as rectangles placed on the field. While the visualization is not the focus of this paper, we show an example in Figure 3. This result is quite significant as the large region near the centre represents a definite "sweet spot". Moreover, it is large enough to be easily exploitable by the player. This result was of substantial interest to Electronic Arts and we regard it as a strong demonstration of the method's practical value.

Accuracy Testing

Testing data for our accuracy studies was obtained by exhaustive sampling over a fine grid in the search space. In low-dimensional scenarios, we can use this approach to obtain a gold-standard for evaluating our techniques. We label each grid element positive (i.e. >40% probability of scoring) or negative based on this sample and refer to these as the *target concepts*. The instability of C4.5 (i.e. small changes to data can give very different rules) adds substantial variance to the plots so accuracy results were smoothed by windowed averaging over six neighbouring iterations.

Blurring When evaluating the performance of any learning system, the eventual use is an important consideration. In gameplay analysis, we are interested in identifying behaviours in the game that can be systematically reproduced (e.g. an area where the player can repeatedly score with ease). However, human players have limited precision in control, so tiny regions of the state space with high-scoring probability are not really what developers are concerned about since the player is unlikely to be able to visit that exact state repeatedly. Therefore, we ignore very small anomalous regions in our target concepts by "blurring" the regions. This is done by using a flood-fill on the fine grid to identify regions with a common prediction. If a region is below a certain size (essentially a resolution parameter) and surrounded on all sides by a differing prediction, it is relabelled to that prediction. This effectively removes small contradictory regions from larger consistent regions in the target concepts. Figure 4 shows the target concepts before and after blurring. The white patches are grid elements with a high probability of scoring. The large region shows many small negative grid elements in the unblurred version that are gone after blurring. Blurring should also be applied to the learned concepts. However, C4.5's built-in decision tree pruning precludes such tiny regions so blurring has no effect.

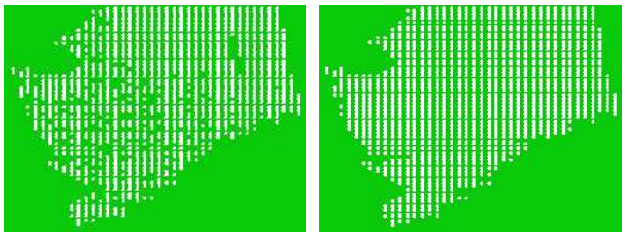

Figure 4: Blowup of corner kick target concepts displayed as crosshatch: Unblurred (left) vs. Blurred (right)

For the corner kick and shooter-goalie scenarios, initial uniform random samples of 1000 and 500 points respectively were taken and each iteration of active learning sampled 300 additional points. All samplers used default parameters. The true positive rate results (TP) are shown in Figure 5 (a) and (b) and false positive rate (FP) in Figure 6 (a) and (b). For the corner kick scenario, most methods perform on par with random sampling in terms of TP rate, which is somewhat disappointing. FP rates are worse for random in general. For shooter-goalie, the picture is somewhat more promising, with several active learners outperforming random on TP and offering comparable performance on FP.

Dimensionality Study

The fact that uniform random sampling performs quite well raises the question of whether active learning is worthwhile. We hypothesize that active learning becomes more important as dimensionality increases because uniform random must sample sparsely. Unfortunately, it is impractical to obtain a gold standard for accuracy testing in higher dimensional spaces. Instead, we studied the impact of dimensionality with synthetic data. We randomly placed 37 disjoint target concepts (each dimension's width randomly in [0.001075, 0.021933]) with a total volume of 1, in spaces of increasing dimension (4, 6, 8, and 10—the range of each dimension is [0,1]). Uniform random and the active learners we compared on true positive rate (TP) (Figure 5 (c), (d), (e) and (f)) and false positive rate (FP) (Figure 6 (c), (d), (e) and (f)). After an initial uniform random sample of 500 points, each iteration of active learning sampled 100 additional points. All samplers used default parameters. Our aim was not a detailed comparison but simply to discover if any active learner is advantageous in higher-dimensional spaces. In these studies, we see uniform random sampling falling behind in terms of both TP and FP as dimensionality increases, confirming our hypothesis that low dimensionality explains its good performance on the two real scenarios.

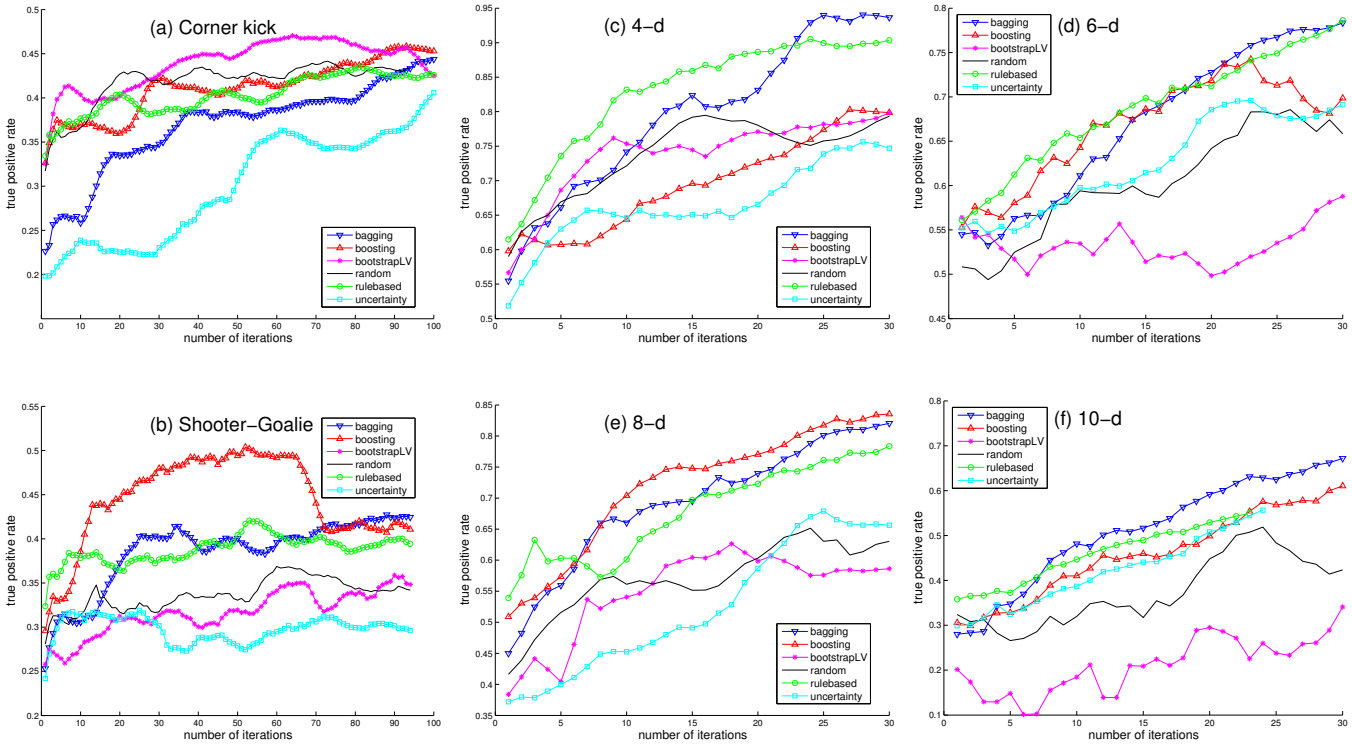

Figure 5: True Positive Rate: TP rate per iteration for six sampling methods on corner kick, shooter-goalie, 4, 6, 8, and 10 dimensional synthetic data.

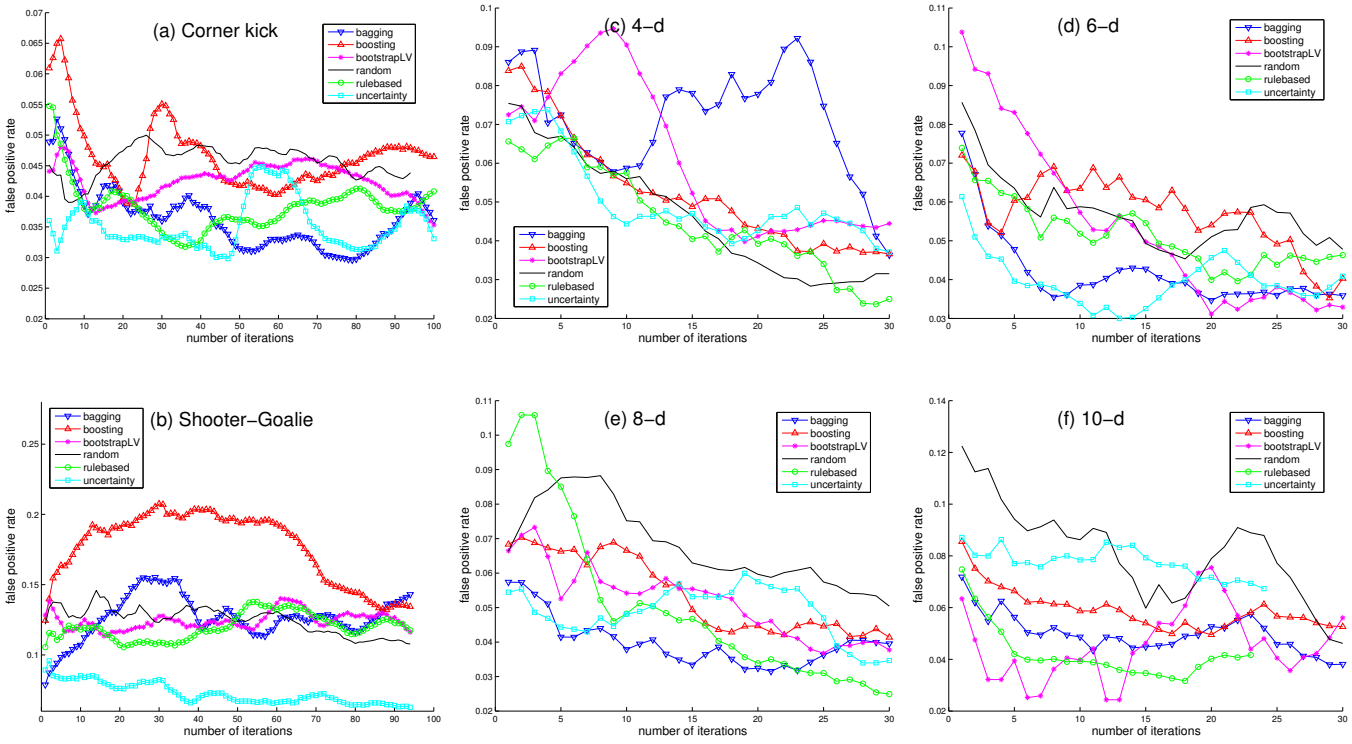

Figure 6: False Positive Rate: FP rate per iteration for six sampling methods on corner kick, shooter-goalie, 4, 6, 8, and 10 dimensional synthetic data.

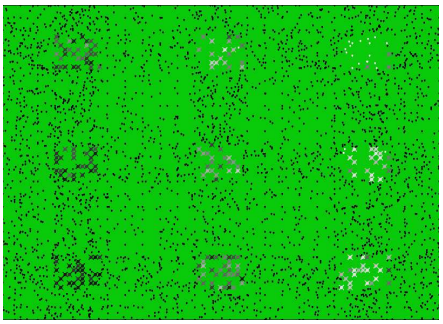

Figure 7: Sampling behaviour of Bootstrap-LV on 9 rectangles with probabilities ranging from 0.1 to 0.9.

A comment on Bootstrap-LV

In studying these active learning methods, we have made a discovery regarding Bootstrap-LV. Figure 7 shows the Bootstrap-LV sampling of a field with nine target concepts (scoring probabilities range from 0.1 to 0.9 with zero everywhere else). Bootstrap-LV wastes the majority of its samples in regions that are almost certainly negative. Visually, it looks similar to uniform random sampling. This represents an intrinsic weakness in Bootstrap-LV. Specifically, when the number of data points with high scores (uncertainty or "local variance") is greatly outweighed by the number with low scores, Bootstrap-LV will tend to select only samples with low scores. This follows from one of its key defining properties—that it treats the scores as probabilities and samples from this distribution instead of simply choosing the N points with the highest scores. For example, suppose there are 100 unlabelled points, of which 4 have fairly high scores, e.g. x, and 96 have substantially lower scores, e.g. $x/10$. Because there is a 24:1 ratio of "lows" to "highs", but their score ratio is only 1:10, a low-scoring point is 2.4 times more likely to be chosen than a high-scoring point each time Bootstrap-LV draws a sample. This issue is worthy of further exploration in case some correction can be applied.

Conclusions

We presented an active learning framework for automated software testing, demonstrating it on the gameplay analysis task. The approach samples a blackbox to learn a model of its behaviour that serves two purposes: visualization and active learning. Our study on FIFA Soccer identified significant sweet spots in the game. Furthermore, Electronic Arts is adopting the approach in-house for testing future FIFA titles. Beyond these anecdotal results, we show that active learning offers scalability to higher-dimensional scenarios. Future work includes new, higher-dimensional scenarios, refining our own rule-based active learning method, and working with new games or other types of software.

Acknowledgements Thanks to the Natural Sciences and Engineering Research Council of Canada, IRIS, and the Alberta Ingenuity Centre for Machine Learning for their financial support, and Electronic Arts for the FIFA source and continuing feedback.

References

Abe, N., and Mamitsuka, H. 1998. Query learning strategies using boosting and bagging. In *Proc. of the 15th Intl. Conf. on Machine Learning (ICML-98)*.

Bergadano, F., and Gunetti, D. 1996. Testing by means of inductive program learning. *ACM Transactions on Software Engineering and Methodology* 5(2):119–145.

Bowring, J. F.; Rehg, J. M.; and Harrold, M. J. 2004. Active learning for automatic classification of software. In *Proc. of the Intl. Symposium on Software Testing and Analysis (ISSTA-2004)*, 195–205.

Chan, B.; Denzinger, J.; Gates, D.; Loose, K.; and Buchanan, J. 2004. Evolutionary behavior testing of commercial computer games,. In *Proc. of the 2004 Congress on Evolutionary Computation (CEC-2004)*, 125–132.

Cheatham, T. J.; Yoo, J. P.; and Wahl, N. J. 1995. Software testing: a machine learning experiment. In *Csc '95: Proc. of the 23rd Annual Conf. on Computer Science*, 135–141.

Cohen, W. W., and Singer, Y. 1999. A Simple, Fast, and Effective Rule Learner. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*.

Denaro, G.; Morasca, S.; and Pezzè, M. 2002. Deriving Models of Software Fault-Proneness. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002)*.

Lachish, O.; Marcus, E.; Ur, S.; and Ziv, A. 2002. Hole analysis for functional coverage data. In *Dac '02: Proc. of the 39th Conf. on Design Automation*, 807–812.

Lewis, D. D., and Gale, W. A. A sequential algorithm for training text classifiers. In *Proc. of 17th ACM Intl. Conf. on Research and Development in Info. Retrieval*.

Poore, J. H., and Trammell, C. J., eds. 1996. *Cleanroom Software Engineering: A Reader*.

Porter, A. A., and Selby, R. W. 1990. Empirically guided software development using metric-based classification trees. *IEEE Software* 7(2):46–54.

Quinlan, J. R. 1994. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann.

Saar-Tsechansky, M., and Provost, F. 2004. Active sampling for class probability estimation and ranking. *Machine Learning* 54(2):153–178.

Sagarna, R., and Lozano, J. A. 2005. On the Performance of Estimation of Distribution Algorithms Applied to Software Testing. *Applied Artificial Intelligence*.

Seung, H. S.; Opper, M.; and Sompolinsky, H. 1992. Query By Committee. In *Proc. of the 5th Workshop on Computational Learning Theory*, 287–294.

Wegener, J.; Sthamer, H.; Jones, B. F.; and Eyres, D. E. 1997. Testing real-time systems using genetic algorithms. *Software Quality Journal* 6(2):127–135.

Whittaker, J., and Thomason, M. 1994. A Markov chain model for statistical software testing. *IEEE Transactions on Software Engineering* 20(10):812–824.