1001 Games a Night – Continuous Evaluation of an Intelligent Multi Agent based System

Eicke Godehardt, Mohamed Amine Allani, Alexander Julian Vieth, and Thomas Gabel

Frankfurt University of Applied Sciences, Niebelungenplatz 1, 60318 Frankfurt, Germany,
godehardt@fb2.fra-uas.de, allani@stud.fra-uas.de, avieth@stud.fra-uas.de, tgabel@fb2.fra-uas.de

Abstract. The goal of the presented approach is to improve the stability of our RoboCup team code by providing an improved continuous integration software engineering process. As big and even small changes in our code base cannot be judged by just a couple of games, roughly 1000 games were run each night to have a good feeling whether changes were for the better or for worse. In addition, it is now possible to analyze the output to gain even deeper understanding of different approaches and parameters. This is supported by interactive visualization techniques.
As a brute force approach will collect way too much data every night, it is necessary to condense the output and keep just a very small fraction of the detailed log data for further analysis. To decide which log files to keep different outlier detection algorithms are compared and optimized.

Keywords: RoboCup 2D, machine learning, multi agent systems, continuous integration

1 Introduction

This paper describes the effort done to setup a continuous integration and development environment to judge the effects of changes as well as enable deeper analysis of the team's performance. The object of investigation is the RoboCup team FRA-UNIted of the Frankfurt University of Applied Sciences [1]. This team participates in the 2D simulation league [2], where twelve autonomous agents (eleven players and one coach) make individual decisions every 100 milliseconds. One central server is coordinating the teams and provides the calculated update of the world back to all agents.

The FRA-UNIted team has quite a long history which led to a pretty big C++ code base. This makes it very hard to make confident changes without a safety net. A continuous integration and deployment strategy can help. The main issue is, that it is hard to judge the team's performance by a couple of games. It is rather necessary to run around 1000 games to really get a robust result whether the last changes where for the good or for worse.

In Section 2, we will discuss related work, based on which the chosen approach is explained in section 3. Section 4 presents the results of an empirical evaluation of the presented approach, and section 5 concludes.

2 Related Work

Two techniques are actually in the focus of this paper. First of all the overall development environment including the execution of a great number of simulated soccer games. On the other hand the outlier detection process as a central element inside of this environment. Here we quickly go over related work in both areas.

At least one other team of RoboCup 2D league – team Helios – has a similar approach of running many games in an automated manner. In [3], they describe a performance evaluation system used for their team utilizing SlackBot, Amazon S3 and Google Sheets. A user can create a job with branch and opponent information as well as the number of games to a SlackBot. A server then assigns client PCs with the jobs depending on CPU load. Client PCs with high load are considered *busy* and do not get jobs assigned. Client PCs run the assigned games, analyze the resulting log files into a CSV file and push them to a shared storage (Dropbox and Amazon S3). The resulting CSV files are aggregated into a Google Sheet, which the user can use to evaluate the performance.

The main difference is the automatic approach presented here combined with local only resources and the focus on more graphical data analysis possibilities. In addition the authors of this paper have no information whether real log files are obtained by the other approach.

3 Approach

While the paper at hand provides a coarse overview of the used techniques and approaches, the interested reader is referred to [4,5] for more details. In [4], special features of the architecture are highlighted, whereas a deeper investigation of the applied outlier detection process is described in [5].

3.1 Architecture

The CI system consists of six components:

- a Git server
- a Jenkins server providing team builds and version info's
- a web server (using Grails) to collect and serve match data and statistics
- an interactive React web application
- 20 test computers running shell and python scripts
- RoboCup 2D tournament software [2,6]

The only interaction a user has with the system is mainly by pushing a new commit to the git repository and by looking at the visualized results on the React web application. As the name suggest, the CI system functions mostly autonomously, with the process of fetching the newest FRA-UNIted build, playing/analyzing games and pushing summaries back to the web server is all initiated by the test computers, which is different to the approach taken in [3]. This has the advantage of the collecting server only being loosely coupled to the test computers, as the server can be oblivious to the source of incoming matches. Scaling this system up or down requires only minimum effort. To enable the over-night running of matches, the test computers are configured with a cron-job, which fetches the newest version of the CI scripts and executes a specific entry point script.

One important feature is, that one can find optimal configuration values based on A–B testing. For this, the number of games can be split, e.g., in half, to see the impact of different configuration values of the agents' behavior.

3.2 Outlier Detection

One really important component in the whole setup is the outlier detection to minimize the number of kept log files for further investigation and deserves its own research. This is quite important as you may gain some understanding from the condensed log information, but can no longer deeper investigate and replay the actual log files to see real decisions in a game. So the main idea is to keep log files of a couple "interesting" games – or in other words anormal games. Here outlier detection comes into play. Outlier detection is a wide field and many reviews discuss research in that area [7,8].

In a given dataset, some samples may differ from most observations to an extent where it would be considerable that they were generated by a different process. These samples are known as *anomalies* and their characteristics are different from those of normal samples [9]. The proportion of outliers in any given dataset is referred to as *contamination*.

Under anomalies, one could differentiate between *outliers* and *novelties*. Generally, the amount of these anomalies represents a small portion of a dataset. When their proportion is lower than 5%, these anomalies are called outliers[10]. Therefore the process of detecting outliers is called *outlier detection*. In this context, the training data already contains outliers that are far from the other observations, also known as *inliers*. On the other hand, novelties are completely new observations; the training data does not contain any anomalies. *Novelty detection* is then the process of detecting novelties and it occurs after the training phase.

All algorithms mentioned in this paper work according to a common principle: the model defines normal data-points by identifying dense clusters containing the most similar points in the dataset. This occurs during the training phase. The next step is to select outliers, either by selecting the outliers that lie in different regions by calculating an *anomaly score* representing the "outlierness" of a data-point for each sample. According to this score, outliers can be selected

by either i) *block-testing* which consists in defining a threshold value based on a given degree of contamination and selecting the points that have a score higher or lower than the threshold or by ii) *consecutive testing* which consists in sorting the samples in descending/ascending order with respect to their anomaly score and with the most suspicious observations at the beginning, then evaluating each sample individually until a normal observation is reached.

The main differences between outlier detection algorithms lies in the way how each method calculates an anomaly score [10]. Under these methods one could differentiate between four groups:

- nearest-neighbor based algorithms
- isolation algorithms
- probabilistic algorithms
- domain-based algorithms

In this approach we focus on the first two groups with one example from either group based on the applicability in the given scenario.

Local Outlier Factor Local Outlier Factor (LOF) is an outlier detection algorithm based on analyzing the direct neighborhood of each data point to determine its outlierness [11,12]. Indeed, the density of the neighborhood is taken into account and is used to calculate a score, namely the Local Outlier Factor. A point is defined as outlier if its score is higher or lower than a certain threshold.

Isolation Forest The other kind of outlier detection applicable in the given scenario is Isolation Forest (IF) [13]. It is based on random forests and computes an isolation score for all instances in the dataset. The algorithm builds binary trees which have random split values on each of their nodes. The result is a "forest" of such trees which processes and assigns an isolation score to each data point. The calculated score is based on the average path length from root to leaf in every tree. The shorter the path, the more likely the point is an outlier.

The Gold Standard As both algorithms are instances of unsupervised learning algorithms, a way to judge the results is highly needed. In order to do so, we asked a RoboCup expert in the team to look manually for outliers in our data set. This lead to the gold standard, both algorithms can be measured by.

Since the algorithms should work on batches of 1000 samples, the labelled dataset has to contain 1000 samples. However, since labelling the data is very time consuming, a way to reduce this number was needed. The 1000 unlabelled samples were used to train both algorithms using the default hyperparameters. As the top labeled outliers seem quite good examples of outliers, only the top 100 samples (50 first samples from every algorithm) were selected for manual labelling. The rest of the samples were assumed to be inliers.

To label the samples, the expert based his choice on the following criteria and features:

1. `goals_<team>`: Goals are the major feature used by the expert to assess a game. For most selected outliers, there is a considerably big difference of at least three between `goals_l` and `goals_r`.
2. `shots_on_target_<team>`: In some samples selected as outliers, the team FRA-UNIted had 0 for shots on target. Furthermore, when there is a discrepancy, i.e., a huge gap between `shots_on_target_<team>` and `goals_<team>`, the sample was marked as outlier.
3. `possession_<team>`: A higher possession for FRA-UNIted is improbable. However, a very small value for possession for FRA-UNIted resulting in lowers values for other features like passes is also considered unlikely.

Considering the 50 samples that obtained the largest outlier scores by IF and LOF, 7 and 5 of them were classified as outlierts by the expert, respectively. Under the assumption that none of the remaining samples were outliers, this yields a degree of contamination of 0.007 for IF and of 0.005 for LOF.

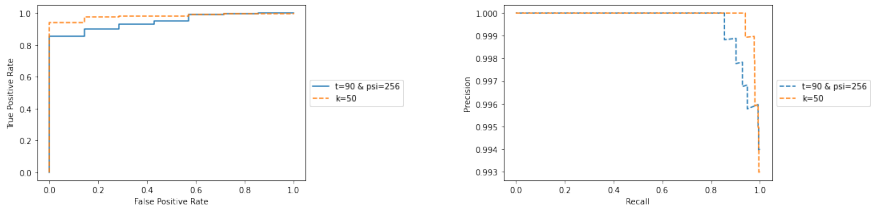
4 Evaluation

In this section, the hyperparameters which have the highest Fowlkes-Mallows index (FMI) score will be selected for both algorithms and compared to one another (FMI score is chosen, because of its resilience to class imbalance). In that context, the IF evaluation with the optimal hyperparameters of the previous experiment ($t = 90$ and $\psi = 256$) and the LOF algorithm with the optimal hyperparameter $k = 150$ are selected. The one on one comparison of their metrics can be seen in the following table.

Hyperparameters	ROC AUC	PR AUC	AP	F1	FMI	Time (s)
IF: $t = 90$, $\psi = 256$	0.94605	0.99960	0.42857	0.99445	0.98893	0.32307
LOF: $k = 150$	**0.98000**	**0.99986**	0.42857	0.99445	0.98893	0.08500

Table 1: Results of the evaluation of the IF and LOF algorithms with their respective optimal hyperparameters (AUC: area under curve, ROC: receiver operating characteristics, PR: precision-recall, AP: average precision, F1: harmonic mean of the precision and recall, FMI: Fowlkes-Mallows index)

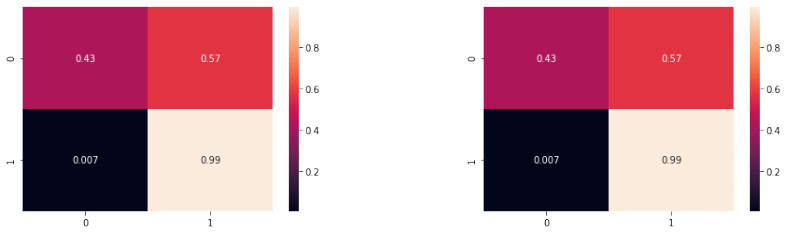
As can be seen in Table 1, although LOF has higher values for ROC-AUC and PR-AUC than IF, both algorithms have equal performance according to the other metrics (F1, AP and FMI). This can be explained by examining their ROC and PR curves (cf. Figure 1).



(a) (b)

Fig. 1: Comparison of IF and LOF with their respective optimal parameters: a) ROC-curve b) PR-curve

Figure 1 and Table 1 show that the AUC of LOF is higher for both, ROC and PR. Indeed, the LOF curve seem to be higher than the IF curve, in the region between 0.0 and approx. 0.6 FPR in the ROC curve and approx. 0.8 and 1 in the PR curve. However, both curves overlap in the other regions. This can be explained by the process that generates these curves and the way the thresholds for these curves are chosen. LOF seems to be performing better than IF when different classification thresholds are chosen. However for the threshold corresponding to the predefined 0.01 contamination ratio, both algorithms have the same performance. This can be demonstrated by the values of the metrics F1-score, AP, and FMI and by examining the confusion matrices in figure 2.



(a) (b)

Fig. 2: Confusion matrices: optimal parameters LOF (a) and IF (b)

Both matrices in figure 2 are identical for both algorithms which suggests that both algorithms perform the same when the threshold is set according to the contamination score 0.01.

Additionally, it is important to note that the detected (hand-marked) outliers are exactly the same for both algorithms, however not all predicted outliers are the same. This has repercussions on how both algorithms will be used in the final version of the system.

5 Conclusion

The RoboCup 2D simulation league continues to evolve, with a steady increase in the capabilities on how the teams play soccer. As every match of 2D soccer is tainted to various degrees by randomness, a need to play a large number of matches arises, to limit the impact of chance on the overall results. For this reason, a continuous integration (CI) system has been implemented. This paper discussed various aspects of the system, which is used by the team of the Frankfurt University of Applied Sciences' RoboCup team – FRA-UNITed, with the goal to assess its performance. The main points discussed where:

- Risk of overfitting, by only playing against the same team
- No interaction and influence on the CI system besides new commits
- Unused potential for configuration testing

In this work, a new CI system was demonstrated, which allows the managing and usage of arbitrary team binaries, along with the ability to freely specify configuration values for arbitrary config files.

Evaluating outlier detection algorithms in the context of analyzing RoboCup 2D games is not trivial. Indeed, the gold standard may be defined according to subjective beliefs that assess the data. Furthermore, the patterns that are used to identify these outliers can be complicated and there is a high possibility that more feature are needed in order to score a higher true negative rate. Therefore, the results of the evaluation can be considered satisfactory, knowing that almost half of the outliers specified by the golden standard are successfully detected.

References

1. T. Gabel, Y. Eren, F. Sommer, A. Vieth, E. Godehardt, RoboCup 2022: Robot Soccer World Cup XXVI, Bangkok, Springer, CD supplement (2022)
2. I. Noda, H. Matsubara, K. Hiraki, I. Frank, Applied Artificial Intelligence **12**(2-3), 233 (1998)
3. M. Yamaguchi, R. Kuga, H. Omori, T. Fukushima, T. Nakashima, H. Akiyama, in *RoboCup 2021 Symposium and Competitions, Worldwide* (2021)
4. A.J. Vieth, A Continous Integration System with Diversified Opponents and Dynamic Team Configurations for RoboCup 2D. Master's thesis, Frankfurt University of Applied Sciences (2022)
5. M.A. Allani, Analysis and Visualization of RoboCup Games using Machine Learning (orig German: Analyse und Darstellung von RoboCup Spielen mit Maschinellem Lernen). Master's thesis, Frankfurt University of Applied Sciences (2021)
6. A. Hechenblaickner. Hech League Manager (2004-2010)
7. K. Ord, International Journal of Forecasting **12**(1), 175 (1996). Probability Judgmental Forecasting
8. O. Alghushairy, R. Alsini, T. Soule, X. Ma, Big Data and Cognitive Computing **5**(1), 1 (2020)
9. D. Hawkins, *Identification of Outliers*. Monographs on Applied Probability and Statistics (Chapman and Hall, London [u.a.], 1980)

10. R. Domingues, M. Filippone, P. Michiardi, J. Zouaoui, Pattern recognition **74**, 406 (2018)
11. S. Mishra, M. Chawla, in *Emerging Technologies in Data Mining and Information Security*, ed. by A. Abraham, P. Dutta, J.K. Mandal, A. Bhattacharya, S. Dutta (Springer Singapore, Singapore, 2019), pp. 347–356
12. M.M. Breunig, H.P. Kriegel, R.T. Ng, J. Sander, in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Association for Computing Machinery, New York, NY, USA, 2000), SIGMOD '00, p. 93–104. DOI 10.1145/342009.335388. URL https://doi.org/10.1145/342009.335388
13. F.T. Liu, K.M. Ting, Z.H. Zhou, in *2008 eighth IEEE International Conference on Data Mining* (IEEE, 2008), pp. 413–422