Smart alarm: unsupervised anomaly detection on rodent behavioral multivariate time series data

Xiaoyue Zhu

Erlich lab of NYU-ECNU Institute of Brain and Cognitive Sciences

Abstract Behavioral neuroscience labs rely on consistent animal performance to generate quality data. It is extremely valuable to be able to detect abrupt changes in animal behavior and adjust settings accordingly. Currently, this step is done manually by the experimenters. This project aims to design a smart alarm that performs unsupervised anomaly detection on rodent behavioural data from the Erlich lab of NYU-ECNU. ForeCA, a dimensionality reduction method designed for multivariate time series is used to smooth the noisy data. Auto-regressive integrated moving average (ARIMA) is then trained on individual ForeCA components to forecast the next point. Anomaly detection is performed by comparing the true component value against ARIMA confidence intervals. A small dataset was manually labelled and used to evaluate model performance. The solution achieved 87% precision rate on that small dataset. Further analysis shows that the same precision rate generalizes to a large portion of the entire dataset.

I. Introduction

The Erlich lab of NYU-ECNU Institute of Brain and Cognitive Sciences specializes in collecting and analyzing complex behavioral data from rodents trained on a variety of tasks. The lab has a custom-built high-throughput rodent behavioral training system (**Figure 1a**). Each training rig features an 8-port pokewall with LEDs and infrared sensors to record pokes from the animals (**Figure 1b**). The water-deprived animals are motivated to poke and learn the task structure as they are rewarded with small amounts of water every time they perform correctly in a trial. Each animal is trained for a session of 80 minutes on its assigned task every day. All the session data is stored in our SQL database.

The animals do not perform in a manner conducive to perfect experimentation, due to the rig changes, protocol / task adjustments, and of course other factors which we are still discovering. The motivation of this project is to design a smart alarm system that detects abrupt changes and subsequently alerts lab members to take appropriate action.

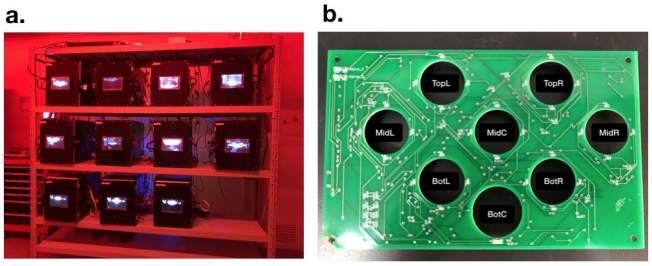

Figure 1. The Erlich lab training rigs and the custom-built pokewall.

II.Datasets

Dataset summary

There are currently 107 animals (52 mice, 55 rats) being trained everyday. In total, they have generated 15,147 sessions to date, with the most trained animal contributing 420 sessions and the newest contributing only 50 sessions. The session data table in SQL contains several columns important to our analysis: the number of trials, the session duration, the total profit (microliter of water rewarded), the hit rate (correlated with performance), the violations rate (negatively correlated with performance) and animal mass. In addition, we store all the poke events within a trial in the trial-event table in JSON format.

Feature extraction

To fully utilize the information contained in the trial-event table, I extracted the poke events from the JSON text string from each trial and computed the session statistics for each session. The resulting data frame has 14 feature columns, with 6 columns from the original session table and 8 poke count columns. **Figure 2** shows example feature distribution of selected columns from subject 2077. Since MidC is a task-relevant port, the animal pokes in it a lot compared to TopR, which is not a task-relevant port.

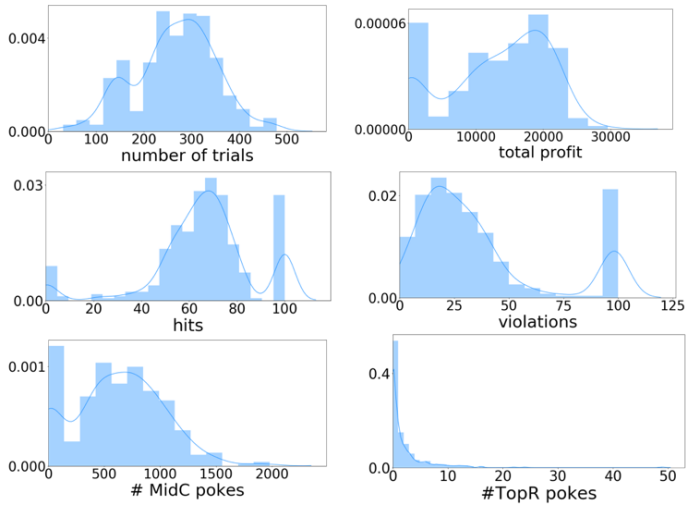

Figure 2. Example feature distributions from subject 2077.

III. Dimensionality Reduction

Rationale

There are two main reasons why performing dimensionality reduction on the dataset is important. Firstly, the current problem can be construed as an anomaly detection problem on multivariate time series with 14 columns. For anomaly detection algorithms to work well, the multivariate time series should be relatively smooth over time. **Figure 3** shows total profit, hits, violations and MidC pokes over all sessions from subject 2077. It can be readily observed that these features are not smooth over the sessions. They need to be smoothed before being fed into the anomaly detection algorithm.

Secondly, training models on high-dimensional data requires a great deal of parameter fitting and is prone to overfitting. Projecting the data onto low dimensions helps reduce the computational cost of model training.

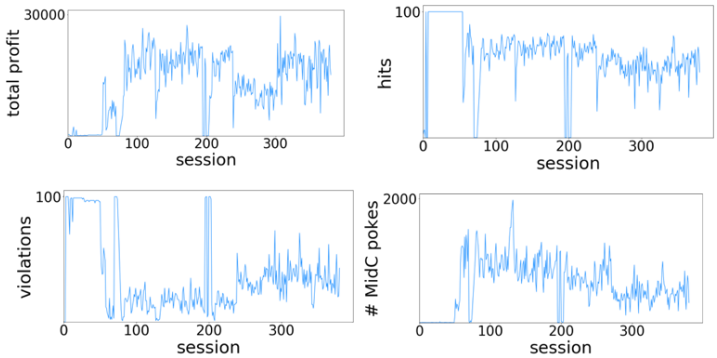

Figure 3. Features are noisy over sessions.

ForeCA

Forecastable Component Analysis (ForeCA) is a dimensionality reduction technique specifically designed for multivariate time series by Goerg (2013). It finds an optimal transformation to separate multivariate time series into a forecastable and orthogonal white noise space. Unlike traditional methods such as principal component analysis (PCA) that projects data onto a low-dimensional space while capturing maximal variance, foreCA projects data onto low dimensions while capturing maximal 'forecastability', Ω.

The advantage of foreCA over PCA is that it takes temporal dependence into account and finds an optimal subspace for forecasting. **Figure 4** compares the results of PCA and ForeCA on the whitened 2077 dataset. Although PCA captures more than 90% of the total variance (**Fig 4a**), further analysis shows the main contributors of variance captured in component 1 are MidLin, TopRin and TopLin (**Fig 4c**). These features contain predictive power of animal performance, but they are not as good as hits, viols and total profits. **Fig 4d** shows MidLin over sessions. It can be seen that the session projections onto PCA component 1 largely reflect MidLin (**Fig 4b**). On the other hand, ForeCA finds 6 components with Ω higher than 0 (**Fig 4e**), suggesting they are forecastable sequences. The main contributors of forecastability captured in component 1 are the animal mass and total profit (**Fig 4g,** note that the sign of the feature weight does not matter). The result is intuitive as the animal gains weight over time (**Fig 4h)** and naturally the mass has the most predictive power. This is good evidence for ForeCA as it can pick up mass as the main predictor, whereas PCA ignores it because it does not contribute to overall variance.

The above results can be replicated for all the other subject datasets tested. Therefore, ForeCA (with 6 components) is chosen over PCA as the dimensionality reduction method.

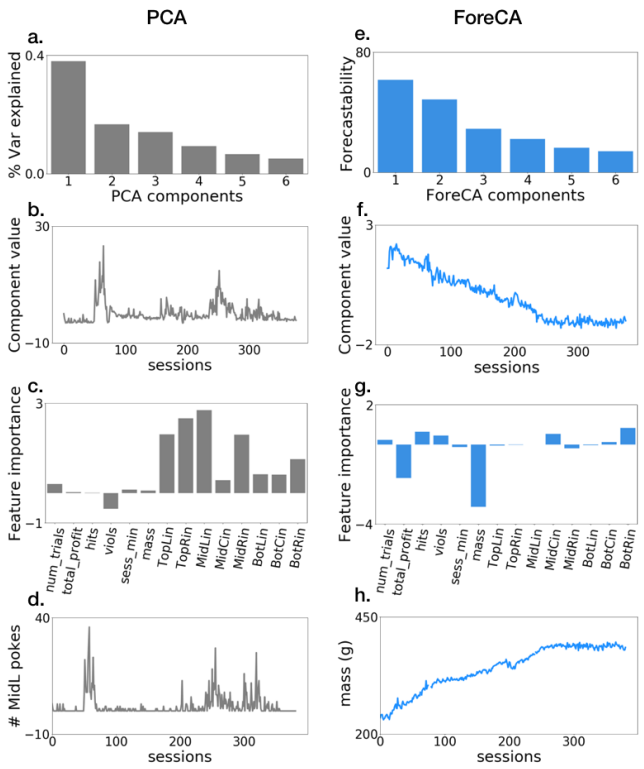

Figure 4. Compare results from PCA and ForeCA.

IV. Time Series Prediction

Previously, the initial noisy 14-dimensional time series is transformed into 6 smoother ForeCA components. This section describes the methods used to predict ForeCA component values in the next time point.

ARIMA

Auto-regressive integrated moving average (ARIMA) is a popular statistical method for time series forecasting. It is a linear regression-based approach that works quite well for short-term forecasting. It is chosen as the main prediction algorithm for three reasons. Firstly, the smart alarm needs to forecast once per day for each animal. As the model needs to be trained on the most up-to-date (excluding the last one) ForeCA components, the model for each animal needs to be trained every day, entailing 107 models trained daily. Thus, the model should be trained quickly to reduce computational cost. Secondly, the model only needs to forecast one step ahead. This further justifies choosing a simple method over a complex one. Thirdly, our largest dataset is only 420 sessions long (from subject 1273), which is still considered a small dataset for more complex methods such as neural networks. The limitation further asserts itself when forecasting on new animals with even smaller datasets. Makridakies et al. (2018) shows that ARIMA tends to outperform machine learning methods on univariate time series forecasting especially when the data size is small.

ARIMA(3, 1, 1) is chosen as the forecasting model. In plain words, it means to construct a linear regression model using 3 lags of the univariate time series with first order differencing (stationarized) and 1 lag of forecast errors.

Rolling window prediction with ARIMA

Rolling window forecasting can be explained by the following example. Imagine there are 200 observations in a time series. A model is trained on the first 100 data points, then it makes a forecast on observation 101. Then the training set is rolled one step forward (from 1 - 100 to 2 -101), the model is trained again on the new training set to make a forecast on observation 102. The process is repeated until observation 200 is included in the training set. Rolling window prediction with ARIMA with window size 60 is performed on individual ForeCA components of all subject datasets. Window size 60 is used because ARIMA tends to have trouble converging when the window size is too small. **Figure 5** shows example results on the first two ForeCA components of subject 2077. The true component scores are shown in gray, the predicted values in light red and the shaded areas represent 90% confidence intervals. Based on visual examination, ARIMA(3, 1, 1) with window size 60 is able to follow the trend and forecast with satisfactory accuracy.

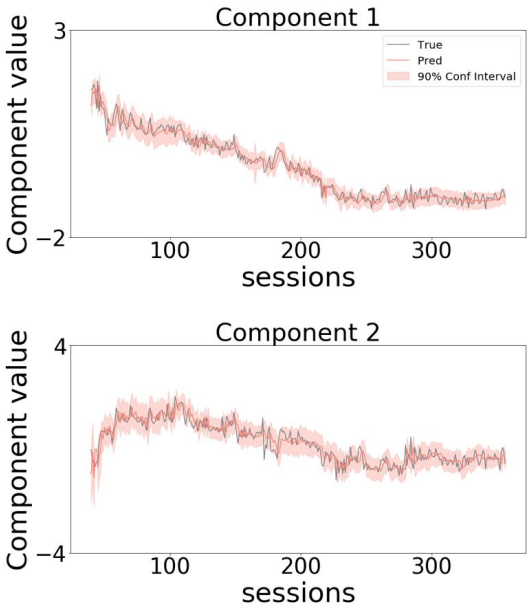

Figure 5. ARIMA rolling window predictions on the first two ForeCA components of subject 2077.

ARIMA and LSTM prediction results

To further evaluate ARIMA performance, its prediction results are compared with that of Long short-term memory (LSTM). LSTM is a special class of recurrent neural network (RNN) that is able to learn long-term dependencies of inputs. It is arguably the state-of-art neural network when it comes to time series prediction. Unlike ARIMA which only handles univariate time series, LSTM can take multivariate inputs. To enable parallel comparison, rolling window prediction with size 60 was performed using one layer LSTM with 32 units, step size 3 and feature size 6. A dropout layer with 0.15 dropout rate is added to avoid overfitting. The metrics chosen to evaluate the prediction results is the root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - f(x^{(i)}))^2}.$$

Where N is the total number of sessions, $y^{(i)}$ is the actual component value and $f(x^{(i)})$ is the model prediction.

Rolling window prediction is performed using ARIMA and LSTM on 5 randomly chosen subject datasets (1045 sessions). **Table 1** summarizes the statistics of RMSEs of 6 ForeCA components. It shows that LSTM moderately outperforms ARIMA on component 2, 3 and 4. Nevertheless, the improved prediction accuracy is not a sufficient ground for choosing LSTM over ARIMA for one major reason: ARIMA gives confidence intervals whereas LSTM does not. Confidence intervals are critical for anomaly detection in the next step. In fact, I have tried using Monte Carlo methods to sample from LSTM (with 0.5 dropout rate) prediction posterior. The confidence intervals computed from the sampled distribution were too narrow for meaningful anomaly detection to occur.

Table 1. RMSEs of ARIMA and LSTM

	ARIMA		LSTM	
Component	*mean*	*std.*	*mean*	*std.*
1	0.040	0.081	0.037	0.056
2	0.078	0.157	0.056	0.092
3	0.147	0.295	0.125	0.145
4	0.135	0.270	0.103	0.085
5	0.217	0.436	0.224	0.378
6	0.187	0.375	0.196	0.127

V. Anomaly Detection

After obtaining ARIMA confidence intervals, anomaly detection becomes straightforward. If the actual component value lies below or above the confidence interval, it is marked as an outlier in this component domain. Naturally, the problem arises as to how to combine the outliers from the 6 component domains. After experimenting with a few subject datasets, I defined a parameter `comp_threshold` to denote how many components out of 6 have to mark this session as an outlier for this session to be qualified as an anomaly. After experimenting with different `comp_threshold` values, 2 is chosen as the best value and is set for all the subject datasets.

Next, I manually inspected and labelled 3 subject datasets (462 sessions total). The criterion I adopted is to ask myself: would I want to check on the animal and adjust its settings based on the sudden changes? The anomaly detection results are shown in **Table 2**. The precision is 86.7% and recall is 78.8% against my labels. It should be noted that the labelling is subjective, different experimenters may give different labelling results.

To confirm that the precision obtained from the small dataset is representative of the entire dataset, I then run anomaly detection on 43 subject datasets (8825 sessions) and stored all the detected anomalous sessions. I then manually either accepted or rejected the model predictions. The resulting confusion matrix is shown in **Table 3**. 2.05% of the entire dataset is marked anomalous by the algorithm, which is consistent with my qualitative experience of animal training: they only deviate from their performance trajectories infrequently. Out of these 181 detected anomalies, 161 are labelled as true anomalies by me, making the precision 88.90%. The recall is impossible to estimate as that would require me to manually find the other true anomalous sessions out of 8644 'normal' sessions.

Table 2. Confusion matrix using 3 datasets

	Detected		
I think	Anomaly	Normal	**Recall**
Anomaly	26	7	78.80%
Normal	4	425	
Precision	86.70%		

Table 3. Confusion matrix using 43 datasets

	Detected	
I think	Anomaly	Normal
Anomaly	161	N.A.
Normal	20	N.A.
Precision	88.90%	

VI. The Expert System

The final step in the smart alarm system following anomaly detection is the expert system. It performs checks on the detected anomalous sessions and determines whether the case is serious enough to alert the experimenters. Example expert system can be found in the pseudo-code in **Figure 6**. The expert system can be calibrated to individual protocols by the experimenters. For example, in a protocol where the animal needs to use all 8 ports, big changes in total profits and hits should be prioritized over changes in TopRin or MidLin. On the other hand, in a protocol where TopR and MidL is never used, sudden increases in these pokes can indicate pokewall malfunction.

This step is currently being implemented in the lab. User calibration and feedback will help improve the smart alarm system in the future.

```
# if the animal has just been transferred to a new protocol
if anomalous_session.protocol != previous_session.protocol:
    alert = False
# the animal may be adapting to the new rig environment
if anomalous_session.rigid != previous_session.rigid:
    alert = False
# the animal has just been changed settings in the protocol
# it will affect its performance
if anomalous_session.settings != previous_session.settings:
    alert = False
# only alert the experimeter when the change is above a certain threshold
# hits can be any other session data
if anomalous_session.hits - previous_session.hits > change_threshold:
    alert = True
```

Figure 6. Pseudo-code of the expert system.

VII. Conclusion and Future Works

In this paper, I have described a novel solution combining ForeCA and ARIMA to achieve unsupervised anomaly detection on animal performance data. Apart from detecting abrupt changes in animal performance, the solution also has broader application in behavioural neuroscience. It can perform change point detection and make decisions based on the animal learning curve, thereby enabling more robust automation of the training pipeline.

ForeCA was used because it projects the noisy high-dimensional time series into smooth low-dimensional space while preserving temporal dependencies. However, the choice of 6 components is rather arbitrary. It is entirely likely that using just 3 or 4 components can have the same precision rate as 6 components. Further testing is needed before fully implementing the solution to minimize computational cost. ARIMA was chosen primarily for practical concerns. The main shortcoming is that it only performs on univariate component series, disregarding the correlations between the original 6-dimensional time series. Although setting the `comp_threshold` to 2 helps circumvent the issue somehow, the choice of 2 is justified on empirical rather than theoretical grounds.

In the future, it would be favorable to try out new methods from the current literature. For example, Malhotra et al. (2016) describes a LSTM-based encoder-decoder model for anomaly detection on multi-sensor data. The idea is that the reconstruction error of an anomaly should be higher than that of normal data. Zhang and Chen (2019) proposed a similar method where LSTM is the encoder and decoder part of the VAE (Variational Autoencoder) model. Of course, the criteria of choosing the final model should be based on model performance, computational cost and interpretability.

VIII. Final Thoughts

Unsupervised anomaly detection on multivariate time series is an active and challenging research field. Unlike supervised learning, unsupervised learning suffers from the lack of training labels and thus the lack of standard evaluation metrics. The issue is especially prominent in this project, as even the definition of an anomalous session can be subjective. One way to approach it is to ask whether the solution performs well in the context of the problem? Hopefully, the user feedback will provide the answer. In the end, I have learned a lot from tackling this project. And that is perhaps the ultimate evaluation metric in the context of this problem.

References

Goerg, G., 2013, February. Forecastable component analysis. In *International Conference on Machine Learning* (pp. 64-72)

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P. and Shroff, G., 2016. LSTM-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*.

Makridakis, S., Spiliotis, E. and Assimakopoulos, V., 2018. Statistical and Machine Learning forecasting methods: Concerns and ways forward. PloS one, 13(3), p.e0194889.

Zhang, C. and Chen, Y., 2019. Time Series Anomaly Detection with Variational Autoencoders. arXiv preprint arXiv:1907.01702.