Documentation of the project code:
Supervised Learning for Supervisory Control of Discrete-Event System

Rafael Mannarelli

Abstract

This documentation accompanies the codebase for the project focused on the application of supervised learning techniques within Supervisory Control Theory (SCT) for Discrete-Event Systems (DES). Bridging the gap between traditional SCT approaches and the dynamic capabilities of machine learning, this project aims to deliver a robust framework for designing data-driven supervisory controllers in complex and unpredictable environments.

Contents

1 - Introduction

In the realm of Discrete-Event Systems (DES), such as manufacturing lines, communication networks, and transportation systems, ensuring operational integrity within predefined constraints is paramount. Traditionally, Supervisory Control Theory (SCT) has provided the means to design controllers that guide these systems to function correctly. However, the increasing complexity and evolving nature of modern DES demand a shift from conventional model-based control strategies to more adaptive, data-driven solutions.

Our project acknowledges these challenges by integrating supervised learning algorithms with SCT to facilitate a novel supervisory control mechanism. Through the use of algorithms like Logistic Regression, Long Short-Term Memory (LSTM) networks, and Random Forest, we aim to build a system that learns from sensor-generated data, capturing the intricacies of DES behavior. The accompanying codebase is developed in MATLAB (MATLAB R2023a). MatlabTCT [1] toolkit was used for generating and analyzing illustrative examples, thus demonstrating the efficacy of our approach. It is a MATLAB toolbox for modeling Discrete Event Systems. It provides a graphical interface for designing and analyzing DES, including specifications and supervisors. This framework is used to model the different examples.

2 - Code Structure

The main script initializes the workspace, sets up parameters, defines user interfaces, and then branches into different sections based on the user's choice of operations like data parsing, labeling, and machine learning model application.

2.1 - Parameter Initialization

Parameters like `meanGaussian`, `sigma`, `minU`, `maxU`, `sizeSampling`, and `percentageTrainingData` are set, which define the data distributions, sampling size, and the proportion of data used for training.

2.2 - User Interface Setup

User-specified options are defined here:

- Random Number Generation (`randomType`): Selects the type of distribution (`Uniform` or `Gaussian`) for generating random numbers.
- Machine Learning Algorithm Selection (`MLType`): Chooses the machine learning algorithm to be used (`LR` for Logistic Regression, `LSTM` for Long Short-Term Memory networks, or `RF` for Random Forest).
- Label Type Selection (`labelType`): Decides the label type for the machine learning task (`CD` for control data, `Marker` for marker state, `Event+1` for event prediction).

- Example Configuration (`example`): Sets a specific example configuration, which alters the flow of data preparation and processing (here are the 5 examples available: `EX_AGV`, `EX_RA`, `EX_SIMPLESMALLFACTORY`, `EX_TRASNFERLINE`) and `EX_ORDER`)

2.3 - Data pre-processing

2.3 - Function: `parser_des`

Description: The `parser_des` function is specifically designed to parse and extract important information from a discrete-event system (DES) file, such as transition data and marker states. This function reads a file, identifies sections containing transitions and marker states, and then saves this information into separate, well-structured text files.

Inputs:

1. **file**: The path to the file containing the DES data.
2. **fileName**: The name of the file, used in naming the output files.
3. **path**: The directory path where the output files will be saved.

Output: The function does not return any variables but writes the extracted transition and marker state data to separate text files.

Usage: `parser_des` is a crucial tool for the initial stages of data processing in studies involving discrete-event systems. By extracting and structuring essential information from raw DES files, it facilitates further analysis and modeling of these systems. The function is especially useful for researchers and analysts dealing with DES data in textual format, needing to segregate and structure transitions and states for further computational tasks.

2.3 - Function: `parser_dat`

Description: The `parser_dat` function is designed for parsing and processing text data from a specified file, particularly focusing on extracting and formatting control data for subsequent analysis in discrete-event system studies. This function reads the text file, identifies the relevant control data section, and reformats it into a structured format suitable for machine learning and data analysis applications.

Inputs:

1. **file**: The file path of the text file containing raw data.

2. **fileName**: The name of the file, used to generate the output file name.
3. **path**: The path where the output file will be saved.

Output: The function does not return any variables but writes the processed control data to a new text file.

Usage: `parser_dat` is critical in the preprocessing phase of data analysis for discrete-event systems. By transforming raw text data into a structured and consistent format, it allows for more effective and accurate subsequent data analysis. This function is particularly useful in scenarios where control data is embedded within text files in a non-standardized format, necessitating extraction and reformatting for further analytical processes.

2.3 - Function: `sup_to_data`

Description: This function, `sup_to_data`, is designed to generate simulation data for supervisory control in DES. It simulates the transitions of a system based on specified parameters and creates sequences of events that represent system behavior. The data generated is then transformed into a format suitable for data-driven machine learning analysis.

Inputs:

1. **dataTransitions**: A matrix containing the transitions of the discrete-event system. Each row typically represents a state transition with columns indicating the current state, event, and next state.
2. **dataMarker**: A vector containing marker state information, identifying specific states of interest in the discrete-event system.
3. **mean**, **sigma**: Parameters for Gaussian distribution (used if `randomType` is 'Gaussian'), where `mean` is the average and `sigma` is the standard deviation.
4. **maxU**, **minU**: Upper and lower bounds for uniform distribution (used if `randomType` is 'Uniform')
5. **randomType**: A string specifying the type of random distribution to use ('Gaussian' or 'Uniform').
6. **sizeSampling**: An integer specifying the number of samples to be generated.

Outputs:

1. **dataEvent**: A matrix where each row represents a sampled sequence of events in the discrete-event system. The length of each row is determined by the specified random distribution and sampling constraints.
2. **posList**: A structure array containing control data for each sample. This data represents the sequence of states traversed in each sampled event sequence.
3. **dataEndPos**: A vector indicating the final state in each of the sampled sequences.

Usage: This function is essential for creating simulation data from discrete-event systems for machine learning applications. It allows the integration of traditional supervisory control theory with modern data-driven approaches, facilitating the exploration of complex system behaviors through supervised learning models.

2.4 - Label Generation

Labels for different machine-learning tasks are generated based on the label type specified by the user.

2.4 - Function: `label_control_data`

Description: The `label_control_data` function processes control data from a discrete-event system to generate labels suitable for supervised learning. It maps the end positions of control data to their corresponding states and creates a binary label matrix indicating the presence of controllable events for each sequence.

Inputs:

1. **dataControlData**: A matrix containing control data, where each row represents a control sequence.
2. **dataEndPos**: A vector indicating the final position or state in each control sequence.

Outputs:

1. **labelControlData**: A binary matrix where each row corresponds to a control sequence and each column to a unique control event. A value of 1 indicates the presence of the event in the sequence.
2. **controlEvent**: A vector of unique control events extracted from the control data.
3. **controlList**: A matrix of control sequences with appended final positions from `dataEndPos`.

Usage: This function is critical for generating labeled datasets from control data of discrete-event systems. These labeled datasets are essential for training supervised learning models, particularly in scenarios where understanding and predicting the occurrence of specific control events is vital. The function's ability to translate raw control sequences into structured, binary labels enables more effective application of machine learning techniques in the analysis of discrete-event systems.

2.4 - Function: `label_event_prediction`

Description: The `label_event_prediction` function is specifically designed for generating labeled data for event prediction tasks in discrete-event systems. It processes sequences of events and creates a dataset that maps each sequence to its subsequent event, facilitating the training of models for predictive analysis.

Inputs:

1. **dataEvent**: A matrix where each row represents a sequence of events in the discrete-event system.
2. **event**: A vector or matrix containing unique event identifiers used in the dataEvent matrix.

Outputs:

1. **labelEventPrediction**: A binary matrix where each row corresponds to a sequence and each column to a unique event. A value of 1 indicates that the event is the next event in the sequence.
2. **dataEventPrediction**: A matrix similar to dataEvent but truncated at a random index for each sequence.
3. **nextEvent**: A vector indicating the next event following the truncated sequence in dataEventPrediction.

Usage: `label_event_prediction` is crucial for preparing datasets for event prediction models in machine learning, especially in the context of discrete-event systems. By generating a dataset that associates sequences with their next events, this function enables the application of supervised learning techniques to predict future events based on past sequences, providing valuable insights into the dynamics and behaviors of such systems.

2.4 - Function: `label_marker`

Description: The `label_marker` function is designed to generate binary labels indicating if the last position is a marker state.

Inputs:

1. **dataMarker**: A vector containing marker states of interest in the discrete-event system.
2. **dataEndPos**: A vector indicating the final positions or states in each sequence of the discrete-event system.

Output:

1. **labelMarker**: A binary vector where each element corresponds to a sequence in the discrete-event system. A value of 1 indicates that the sequence ends in a marker state, while 0 indicates otherwise.

Usage: `label_marker` is an essential function for labeling sequences in discrete-event systems based on their termination in specified marker states. This labeling is particularly useful in supervised learning models where the goal is to predict the likelihood of sequences ending in critical states. By providing a simple yet effective means of generating binary labels, this function aids in the development of predictive models for system behavior analysis.

2.5 - Feature Extraction

Features are prepared differently based on the chosen machine learning model and the provided example.

2.5 - Function: `data_for_LR`

Description: The `data_for_LR` function is specifically designed to process event data for use with Logistic Regression (LR) models in machine learning. This function takes the raw event data and the discrete-event system transitions to generate features that are compatible with the requirements of logistic regression analysis.

Inputs:

1. **dataEvent**: A matrix representing sequences of events in the discrete-event system. Each row corresponds to a sampled sequence, and each column to an event in that sequence.
2. **dataTransitions**: A matrix detailing the state transitions of the discrete-event system, used to identify the unique events.

Output:

1. **features**: A matrix where each column represents the occurrence count of a specific event across all sampled sequences. This matrix serves as the feature set for logistic regression models.

Usage: This function plays a crucial role in preparing data for Logistic Regression models in machine-learning applications focused on discrete-event systems. By converting raw event data into a logistic regression-friendly format, `data_for_LR` enables the application of statistical analysis techniques to understand and predict system behaviors based on event occurrences.

2.5 - Function: `data_for_RNN`

Description: The function `data_for_RNN` is tailored for preparing data for use with Recurrent Neural Network (RNN) models in machine learning. This function processes event sequence data, creating input and output datasets compatible with the RNN format, which is essential for sequence-based learning tasks.

Inputs:

1. **dataEvent**: A matrix containing sequences of events in the discrete-event system. Each row represents a sampled sequence, and each column represents an event in that sequence.
2. **event**: A vector or matrix containing unique event identifiers used in the dataEvent matrix.
3. **label**: A vector containing labels associated with each sequence in dataEvent, used as the target output for the RNN model.

Outputs:

1. **xRNN**: A cell array where each cell contains a binary representation of an event sequence, suitable as input to an RNN.
2. **yRNN**: A vector or categorical array of labels corresponding to each sequence in xRNN.

Usage: This function is pivotal for converting discrete-event sequence data into a format that RNN models can process. By transforming event sequences into binary representations, `data_for_RNN` facilitates the application of RNNs in analyzing patterns and predicting future events in discrete-event systems. It is particularly useful in scenarios where understanding the sequential dependencies and temporal dynamics of events is crucial.

2.6 - Machine Learning Models

- **Logistic Regression (LR)**: A classification algorithm for binary classification.
- **Long Short-Term Memory (LSTM)**: A recurrent neural network for sequence data.
- **Random Forest (RF)**: An ensemble learning method for classification.

Each model's subsection includes data preparation specific to that model, the training process, and validation to compute the accuracy.

2.6 - Function: `logistic_regression`

Description: The `logistic_regression` function implements logistic regression using gradient descent. It takes feature data and corresponding binary labels, splits them into training and testing sets, trains a logistic regression model, and calculates predictions for the testing data.

Inputs:

1. **features**: The features for Logistic Regression
2. **label**: A binary vector corresponding to each observation in the feature matrix.
3. **percentageTrainingData**: The percentage of the dataset to be used for training the model.

Outputs:

1. **predictions**: The probability values predicted by the logistic regression model for the testing data.
2. **testLabel**: The actual labels for the testing data.
3. **testFeatures**: The features of the testing data.
4. **theta**: The trained parameter values (weights) of the logistic regression model.

Usage: `logistic_regression` is essential for conducting binary classification tasks using logistic regression. This function allows the application of a fundamental machine learning technique to datasets, facilitating the prediction of binary outcomes based on feature data. It is particularly useful in scenarios requiring probabilistic outcomes and decision-making based on statistical learning.

2.6 - Function: `recurrent_neural_network`

Description: The `recurrent_neural_network` function is designed to train a recurrent neural network (RNN), specifically an LSTM (Long Short-Term Memory) network, on sequence data. It takes sequence data and corresponding labels, splits them into training and testing sets, constructs an LSTM model, trains it, and evaluates its performance on the testing data.

Inputs:

1. **xRNN**: A cell array where each cell contains sequence data for training the RNN.
2. **yRNN**: A matrix containing labels corresponding to each sequence in xRNN.
3. **percentageTrainingData**: The percentage of data to be used for training the model.

4. **event**: A vector or matrix containing unique event identifiers used in the sequence data.
5. **numClasses**: The number of classes for classification in the output layer.

Outputs:

1. **predictLabel**: The predicted labels for the test data.
2. **testLabel**: The actual labels for the test data.
3. **layers**: The layers of the constructed LSTM network.
4. **net**: The trained LSTM network model.
5. **options**: The training options used for training the LSTM network.

Usage: `recurrent_neural_network` is vital for building and evaluating LSTM models for sequence data classification tasks. It allows the application of advanced machine learning techniques to sequence data, facilitating the prediction of future events or classification based on temporal patterns. This function is particularly useful in scenarios where understanding the sequential dependencies in data is crucial.

2.6 - Function: `random_forest`

Description: The `random_forest` function is designed for building and evaluating Random Forest models for classification tasks. It takes feature data and corresponding labels, splits them into training and testing sets, trains a Random Forest model, and evaluates its performance on the testing data.

Inputs:

1. **features**: The features of Random Forest (same as Logistic Regression)
2. **label**: A binary matrix where each row corresponds to an observation and each column to a different classification task.
3. **percentageTrainingData**: The percentage of the dataset to be used for training the model.

Outputs:

1. **predictLabel**: The predicted labels for the testing data.
2. **testLabel**: The actual labels for the testing data.
3. **testFeatures**: The features of the testing data.
4. **model**: A cell array of trained Random Forest models, one for each column in the label matrix.

Usage: `random_forest` is an essential function for conducting classification tasks using the Random Forest algorithm. By providing a framework for model training and evaluation, this function enables the application of machine learning to predict outcomes based on feature data. It is particularly useful in scenarios where multiple classification tasks need to be performed on the same dataset.

3 - Validation and Results

The script prints the accuracy of the model's predictions on the test set providing insight into the model's performance.

4 - Conclusion

This MATLAB script is structured to provide a flexible environment for different machine-learning applications. Users can specify parameters, select different algorithms, and choose labeling types to experiment with various scenarios in their data analysis.

References

[1] W. M. Wonham and K. Cai. *Supervisory Control of Discrete-Event Systems*. Communications and Control Engineering. Springer, 2018.