Reproducing When to Intervene: Learning Optimal Intervention Policies for Critical Events

Yuning Wang
Department of Computer Science
Rutgers University
yw895@scarletmail.rutgers.edu

Abstract

In this project, we offer an implementation for the the Dynamic Deep Recurrent Survival Analysis method that solves the optimally timed intervention (OTI) problem using an analytical solution proposed by [2]. We brief introduce the theoretical contribution of the original paper, along with our implementation details.

1 Introduction

In this report, we first introduce the problem to be solve and key equations implemented. Then, we introduce the model architecture and dataset with corresponding details. Finally, we evaluate our model and present the results.

In summary, our contributions are two fold: First, we offer a open source implementation that can reproduce a portion of the experiments conducted in the original paper [2]. Our implementation will serve as a convenient starting point for future researchers on this topic. Second, we empirically validate the optimal formulae proposed by [2] subject to some estimation error of the hazard rate function.

Our implementation and the dataset can be access at `https://github.com/Yuning30/CS535_final_project` and `https://github.com/Yuning30/AMLWorkshop`

2 Problem Statement

2.1 Optimally Time Intervention (OTI) Problem

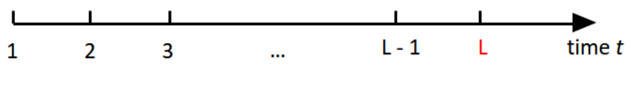


Figure 1: A discrete time series and a critical event happens at timestep L

In this section, we briefly reiterate the theoretical contribution of the original paper [2]. Interested reader should carefully read the original paper.

Given a discrete time series, at each timestep j, we can observe a real-valued vector X_j. An critical event will happens at some timestep $L \leq L_{\max}$ where constant $L_{\max}$ is the maximum length of the time series.

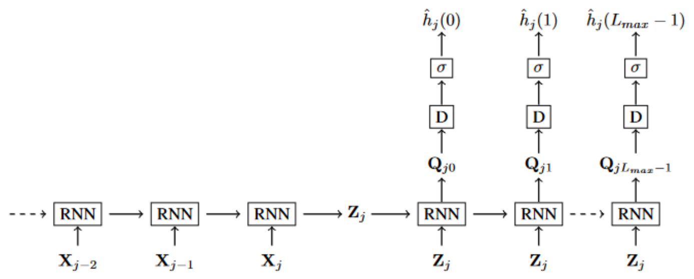


Figure 2: Dynamic Deep Recurrent Survival Analysis (DDRSA) Architecture (taken from Figure. 1 of the original paper)

2.2 Hazard Rate Process

By looking at the discrete time series in a probabilistic way, we can define the corresponding hazard rate process of a discrete time series. At timestep j, the observation is now $\mathcal{X}_j = [X_1, X_2, \ldots, X_j]$ by group all observations up to time step j.

The authors define the hazard rate function $h_j(k)$ (Equation 1) as the probability of the critical event happens at time $j + k$ given all previous observations and the fact that the critical event has not happened before. They also define $H_j = [h_j(0), h_j(1), \cdots, h_j(L_{\max} - 1)]$ [2].

$$h_j(k) := \mathbb{P}\left(L = j + k \mid \mathcal{X}_j, L \geq j\right) \tag{1}$$

2.3 Analytical Solution to OTI Problem

The authors proves that the optimal strategy at timestep j is to intervene if $T_j \leq V_j(H_j)$ [2].

Introduced in Equation 9 and 10 from the original paper, T_j is the expected time to the critical event and $V_j(H_j)$ can be viewed as a time-varying threshold. The C_α is the penalty of failing to intervene [2].

$$T_j = \sum_{k=0}^{L_{\max}-1} \prod_{m=0}^{k} (1 - h_j(m)) \tag{2}$$

$$V_j\left(H_j\right) = \min_{0<l<L_{\max}-j} \sum_{k=0}^{L_{max}} \prod_{m=0}^{k-l} (1 - h_j(m)) + C_\alpha \sum_{m=1}^{l} \left[\prod_{k=0}^{m-1} (1 - h_j(k))\right] h_j(m) \tag{3}$$

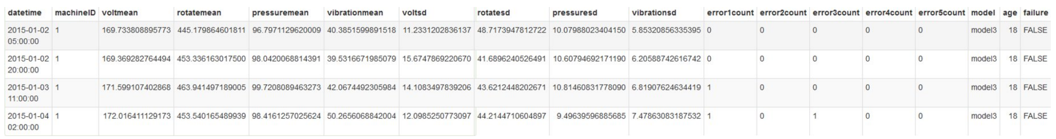
2.4 Implementation Detail

We implement Equation 2 and 3 in file `inference.py`. The input to these functions is the predicted h_j values for the current timestep and C_α. We also implement functions to make continuous decisions for a test sequence data (introduced in Section 4.1) in this file.

3 Network and Training

3.1 Dynamic Deep Recurrent Survival Analysis (DDRSA) Architecture

As we have already seen in Section 2, the optimal solution of the OTI problem can be obtained analytically given the knowledge of the future hazard rate function h. We would like to train a model to give the prediction of the hazard rate function $h_j(0), \ldots, h_j(L_{\max} - 1)$ at a timestep j. The authors propose to use a sequence to sequence model, which they call Dynamic Deep Recurrent Survival Analysis (DDRSA) model to estimate the hazard rate function [2].



datetime	machineID	voltmean	rotatemean	pressuremean	vibrationmean	voltsd	rotatesd	pressuresd	vibrationsd	error1count	error2count	error3count	error4count	error5count	model	age	failure
2015-01-02 05:00:00	1	169.733808895773	445.179864601811	96.7971129620009	40.3851599891518	11.2331202836137	48.7173947812722	10.07988023404150	5.85320856335395	0	0	0	0	0	model3	18	FALSE
2015-01-02 20:00:00	1	169.369282764494	453.336163017500	98.0420068814391	39.5316671985079	15.6747869220670	41.6896240526491	10.60794692171190	6.20588742616742	0	0	0	0	0	model3	18	FALSE
2015-01-03 11:00:00	1	171.599107402868	463.941497189005	99.7208089463273	42.0674492305984	14.1083497839206	43.6212448202671	10.81460831778090	6.81907624634419	1	0	0	0	0	model3	18	FALSE
2015-01-04 02:00:00	1	172.016411129173	453.540165489939	98.4161257025624	50.2656068842004	12.0985250773097	44.2144710604897	9.49639596885685	7.47863083187532	1	0	1	0	0	model3	18	FALSE

Figure 3: example records from the Azure Preventative Maintenance dataset

Fig. 2 illustrate the DDRSA architecture. At a given timesstep j, the encoder network (left half) takes the sequence of previous observations at input and uses its final hidden state as an encoded representation for the observation. Then, the encoded representation is fed into the decoder network (right half) to produce a sequence of hidden states (Q_j values in the figure). Then the hidden states are passed into a dense layer (D in the figure) and a sigmoid function (σ in the figure) to output a sequence of the estimated hazard rate function at timestep j.

3.2 Implementation Detail

The encoder and decoder models are implemented in the `model.py` file using LSTM with hidden size 16. The full DDRSA model and the training code are implemented in `train.py` file. We use the same hyperparameter introduced in the paper: We use Adam optimizer and set the learning rate to 0.01 [2].

4 Experiments

4.1 Azure Preventative Maintenance Dataset

4.1.1 Dataset Overview

The Azure Predictive Maintenance dataset [1] contains the operating records of 100 machines. Each machine can fail after some normal operation time. We following the `Lab Guide.ipynb` to convert the raw data into suitable features for neural networks. We also follow the original paper to down sample the data sequence to 1 record for every 15 hours [2].

After the down sampling, each record consists of features such as machine ID, machine type and age, mean value for voltage, rotation over a time interval and their standard deviations, and other features such as error count for different error type, while errors are not considered failures.

Several example records are shown in Fig 3. Note that the timestep column are not used to train the DDRSA model.

4.1.2 Dataset Parsing Detail

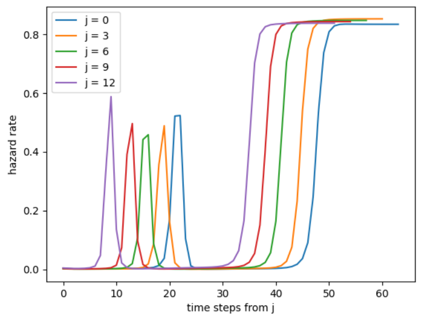
In this section, we introduce the details about how to convert the sequence data into suitable training data for the DDRSA architecture. This procedure is not explicit mentioned in the original paper but is key to reproduce the results. This procedure is implementedin the `dataset.py` file.

We follow the steps below to obtain the DDRSA training data

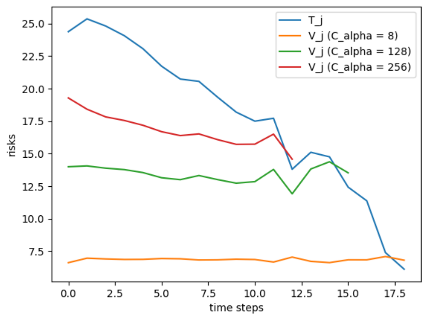
1. For each machine's record, divide the whole sequence to small sequences such that each small sequence only contains a failure timestep in the end.
2. Filter out the sequences whose length is greater than 64.
3. For each sequence, for each possible timestep, use the observations before this timestep as $\mathcal{X}_j$, and use the binary labels after this timestep as the ground truth value for H_j.

4.2 Prediction of Hazard Rate Function

In 4a, we visualize the predicted values for the hazard rate function for different timesteps. We can see that later timestep, such as $j = 12$, will see the peak value earlier than earlier timestep, such as $j = 0$. This matches our expectation.



(a) Hazard Rate Function Prediction



(b) Visualizing T_j and V_j

Figure 4: Prediction of Hazard Rate Function

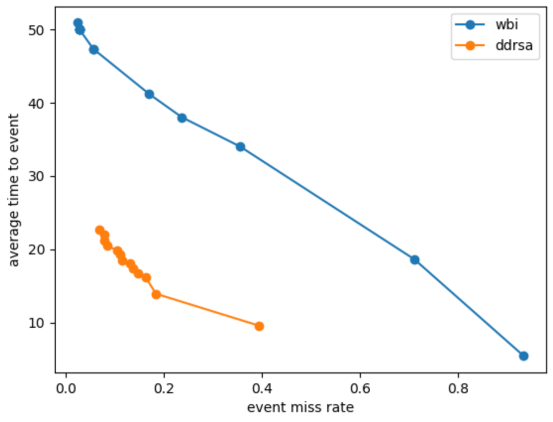


Figure 5: Comparison Between DDRSA and WBI

In 4b, we visualize the T_j and V_j values calculated using 2 and 3. We will intervene once the T_j value fails below the V_j value. We can see the when the penalty of missing the rate is higher (e.g. $C_\alpha = 256$), the policy will intervene earlier.

Note that, the DDRSA models used in our experiments are trained using sequences whose length are less or equal to 64 because the DDRSA model performs badly on datasets containing both short (≤ 64) and long (> 64) sequences in our experiments.

4.3 Comparison with WBI baseline

In this section, we plot the average time to event against the event miss rate for different C_α values for both WBI and DDRSA model. According to the authors, the WBI baseline treats the intervention problem as a classification problem. It only use one LSTM network that takes the observation of previous several steps as input and produce a hazard rate. A constant thershold is then used to decide to intervenet or not [2]. Its implementation can be found in the `train.py` file.

From 5, we can find that the DDRSA model has better performance than the WBI model. The DDRSA model has lower average time to event value when keeping the event miss rate constant. It also has lower event miss rate when keeping the average time to event constant. Our results are aligned with the results in the original paper [2].

5 Conclusion

We have presented our implementation of the DDRSA model and validated key equations in the original paper. We hope our implementation can be a starting point for future researchers to prototype their ideas.

References

[1] Microsoft. Dataset from azure modeling guide for predictive maintainence, 2017.

[2] NIRANJAN DAMERA VENKATA and Chiranjib Bhattacharyya. When to intervene: Learning optimal intervention policies for critical events. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.