JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 4, pp. 37-45, April 2020
https://doi.org/10.9708/jksci.2020.25.04.037

Cleaning Noises from Time Series Data with Memory Effects

Jae-Han Cho*, Lee-Sub Lee*

*Student, Dept. of Computer Engineering, Kumoh National Institute of Technology, Gumi, Korea
*Professor, Dept. of Computer Engineering, Kumoh National Institute of Technology, Gumi, Korea

[Abstract]

The development process of deep learning is an iterative task that requires a lot of manual work. Among the steps in the development process, pre-processing of learning data is a very costly task, and is a step that significantly affects the learning results. In the early days of AI's algorithm research, learning data in the form of public DB provided mainly by data scientists were used. The learning data collected in the real environment is mostly the operational data of the sensors and inevitably contains various noises. Accordingly, various data cleaning frameworks and methods for removing noises have been studied. In this paper, we proposed a method for detecting and removing noises from time-series data, such as sensor data, that can occur in the IoT environment. In this method, the linear regression method is used so that the system repeatedly finds noises and provides data that can replace them to clean the learning data. In order to verify the effectiveness of the proposed method, a simulation method was proposed, and a method of determining factors for obtaining optimal cleaning results was proposed.

▸ **Key words**: Deep Learning, Time Series, Preprocessing, Noise, Data Cleaning

[요 약]

딥러닝의 개발 프로세스는 대량의 수작업이 요구되는 반복적인 작업으로 그 중 학습 데이터 전처리는 매우 큰 비용이 요구되며 학습 결과에 중요한 영향을 주는 단계이다. AI의 알고리즘 연구 초기에는 주로 데이터 과학자들에 의해 완벽하게 정리하여 제공된 공개 DB형태의 학습데이터를 주로 사용하였다. 실제 환경에서 수집된 학습 데이터는 주로 센서들의 운영 데이터이며 필연적으로 노이즈가 많이 발생할 수 있다. 따라서 노이즈를 제거하기 위한 다양한 데이터 클리닝 프레임워크와 방법들이 연구되었다. 본 논문에서는 IoT환경에서 발생 될 수 있는 센서 데이터와 같은 시계열 데이터에서 노이즈를 감지하고 제거하는 방법을 제안하였다. 이 방법은 선형회귀 방법을 사용하여 시스템이 반복적으로 노이즈를 찾아내고, 이를 대체할 수 있는 데이터를 제공하여 학습데이터를 클리닝한다. 제안된 방법의 효과를 검증하기 위해서 본 연구에서 시뮬레이션을 수행하여, 최적의 클리닝 결과를 얻을 수 있는 인자들의 결정 방법을 확인하였다.

▸ **주제어**: 딥러닝, 시계열, 전처리기, 노이즈, 데이터 클리닝

• First Author: Jae-Han Cho, Corresponding Author: Lee-Sub Lee
*Jae-Han Cho (jaehanfs@gmail.com), Dept. of Computer Engineering, Kumoh National Institute of Technology
*Lee-Sub Lee (eesub@kumoh.ac.kr), Dept. of Computer Engineering, Kumoh National Institute of Technology
• Received: 2020. 03. 24, Revised: 2020. 04. 10, Accepted: 2020. 04. 13.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

딥러닝의 개발 프로세스는 대량의 수작업이 요구되는 반복적인 작업으로 그 중 학습 데이터 전처리는 매우 큰 비용이 요구되며 학습 결과에 중요한 영향을 주는 단계다. AI의 알고리즘 연구 초기에는 주로 데이터 과학자들에 의해 완벽하게 정리하여 제공된 공개 DB형태의 학습데이터를 주로 사용하였다. 머신 러닝이 일반화됨에 따라 다양한 응용분야의 적용을 위해 데이터 과학자들이 실제 환경에서 발생되는 데이터를 직접 수집하고 가공해야 한다.

Fig. 1은 전체적인 딥러닝 개발 프로세스를 보여준다[1]. 딥러닝을 수행하기 위한 첫 번째 단계인 데이터 전처리(data preprocessing) 과정은 데이터 클리닝(cleaning)과 레이블링(labeling)을 포함한다. 두 번째 단계는 전 처리된 입력 데이터를 벡터화하는 방법과 중요 특성(feature)을 선택하고, 응용분야에 특화된 특성을 추가하는 작업 등을 수행한다. 세 번째 단계에서는 응용 분야에 적합한 딥러닝의 모델을 선택한다. 네 번째 단계에서는 하이퍼-파라메타를 튜닝하고 다섯 번째 단계에서는 모델이 얼마나 정확한가를 분석한 후 수정을 거친다. 마지막 단계에서 실제 서비스를 제공한다.

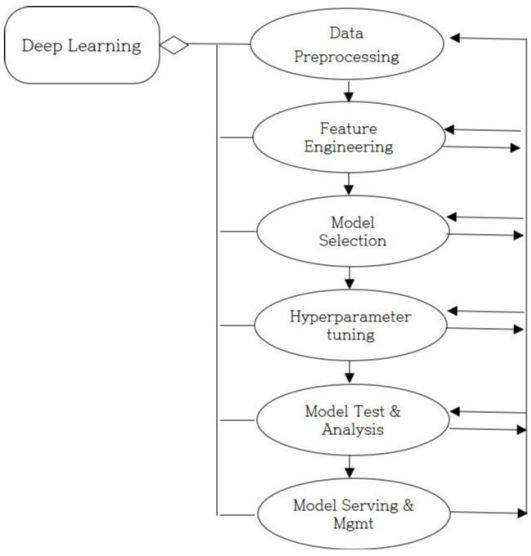

Fig. 1. Human-In-The-Loop Process in Deep Learning

각 단계에서 데이터 과학자는 어떤 작업을 수행해야 할 것인가를 결정하고, 대부분 수작업을 통해 처리한다. 데이터 애플리케이션 개발자와 데이터 과학자는 모델이 원하는 기준을 달성할 수 있을 때까지, 각 단계를 완료한 후 평가를 통해 이전 단계로 돌아가 재작업을 반복적으로 수행해야 한다. 이러한 시행착오를 통한 작업의 반복은 상당한 인력과 자원을 요구한다. 이에 따라 딥러닝 과정은 각 단계들 마다 사람의 손을 타야 하는 Human-In-The-Loop (이하 HITL) 프로세스로 정의할 수 있다[2]. 따라서 이를 자동화하는 것 즉 AutoML[3, 4]이 최근에 진행되는 가장 중요한 연구 분야이다.

Fig. 2는 각각의 단계별로 얼마나 많은 수작업이 필요한가를 통계적으로 보여준다[5]. 데이터 전처리 과정은 데이터 과학자들의 전체 작업의 60%에 해당할 정도로 업무량이 많다. 그 원인은 딥러닝에서 가장 기본적으로 요구하는 대규모의 학습 데이터로 인하여 많은 작업량을 요구하고, 이 과정 자체가 수작업을 요구될 뿐만 아니라, 가장 첫 단계이므로 이후 작업 단계들의 문제에 대한 원인을 제공하여 반복적으로 수행되므로 작업량이 더욱 증가된다. 전처리 이후 단계들 즉, 모델의 훈련 프로세스들과 훈련된 모델 자체를 개선하기 위해 많은 자동화 기술들이 연구되었지만, 오류의 많은 원인들을 제공하는 사전 처리단계가 결과물의 품질에 가장 큰 영향을 미친다.

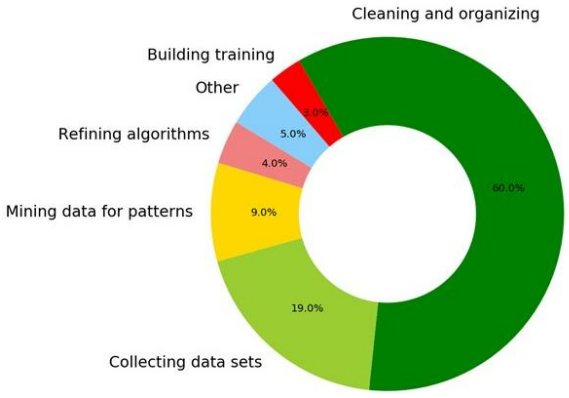

Fig. 2. Time Breakdown for Machine Learning (from 2016 Data Science Report, CrowFlower)

데이터 전처리 단계는 데이터를 분석 가능한 형태로 만드는 작업이다. 전처리 단계에서는 데이터 양식을 표준에 맞게 변환하는 데이터 변환, 데이터의 노이즈를 제거하는 데이터 클리닝, 그리고 학습 데이터의 결과 값 즉 y값에 해당하는 레이블을 기입하는 레이블링이 수행된다. 데이터 변환과 레이블링은 자동화에 대한 연구들이 진행되었고, 데이터 전처리와 클리닝을 위한 다양한 프레임워크들도 제안되었다.

응용 분야에 따른 다양한 도메인에 대한 연구들도 진행되었다. 예를 들어 단순한 데이터 품질을 향상시킬 수 있는 다양한 무결성 제약(도메인 제약, 참조무결성 제약 및 기능적 종속성)에 대한 연구들이 진행되었으며 클라우드 소싱으로 습득된 학습 데이터 처리에 특화된 연구들도 진행되었다. 데이터 클리닝은 도메인에 따라 클리닝 방법이 종속되므로 다

양한 도메인에 따른 클리닝 방법에 대한 연구가 필요하다.

머신 러닝에서는 대량의 데이터가 필수적이고 IoT의 센서 데이터는 대량의 데이터를 확보할 수 있는 가장 중요한 자원이다. 본 논문에서는 데이터 클리닝 중에서 센서 데이터에서 많이 발생이 되는 시계열 데이터 도메인에 대한 노이즈 검출 및 보정에 관련된 방법을 연구하였다.

본 논문은 먼저 관련 연구를 통해서 데이터 클리닝에 활용되는 프레임워크 들을 분석해 보고 본론에서는 시계열 데이터의 노이즈 클리닝 방법을 제시하고, 시뮬레이션을 통해서 검증과정을 보여준다. 그리고 결론을 통해서 연구 결과와 향후 연구에 대해 논의한다.

II. Preliminaries

1. Related works

1.1 Data cleaning related frameworks

HoloClean[6]은 데이터의 범위 점검, 정합성(Integrity)과 같은 품질 규칙과 참조 관계 등을 사용하여 클리닝한다. 특히 관계 데이터베이스의 기존 원리를 차용을 하여 데이터의 오류를 찾아내며 주요 대상이 테이블 형태의 도메인이다.

ActiveClean[7]은 주어진 오류가 포함된 학습 데이터 집합을 사용하여 딥러닝 모델을 학습시킨다. 시스템이 SVM(Supporting Vector Method)와 선형회귀법 등의 방법들을 사용하여 오류 가능성이 높은 데이터를 찾아내고 작업자가 모델의 정확도를 고려하여 오류 여부를 판단한다. 충분한 정확도를 달성할 때까지 상기 작업들을 반복적으로 진행한다. 다양한 분야에 적용할 수 있는 장점은 있으나, 모델을 반복적으로 학습시켜야 하므로 상당한 컴퓨팅 파워를 요구하며, 개발자가 매번 개입해야하는 문제가 있다.

BoostClean[8]은 HoloClean과 유사하게 도메인 즉 참조무결성을 사용한 클리닝 기능을 제공하지만, 클리닝 방법보다는 일종의 프레임워크의 성격이 강하다. 즉 미래에 추가될 수 있는 클리닝 규칙의 플러그인 형태로 추가하고 개선할 수 있는 구조적 특징을 제공한다.

MLCLEAN[9]는 전통적인 데이터 클리닝과 모델이 특정 데이터에 편중되지 않게 구성하는 모델 편중 완화 (Model Unfairness Mitigation)와 보안에 대한 공격자들에 의하여 발생된 데이터의 세정(Data sanitization)등 지금까지 연구된 다양한 기본 방법들은 통합하는 프레임워크를 제안하였다.

TARS[10]는 클라우드 소스로 획득한 학습 데이터에 특화되어 있다. 클라우드 소스는 다수의 사용자들로부터 학습 데이터를 공개적으로 수집하는 방법으로 입력 오류 발생 확률이 매우 높다. 이들 간의 정합성을 고려하여 클리닝을 수행한다. 특히 레이블에 대한 클리닝에 특화되어 있다.

지금까지의 연구들을 전반적으로 살펴보면 단순한 데이터 클리닝 방법에서 시작하여 데이터 클리닝의 프레임워크에 대한 연구가 진행되어져 왔으며, 최근에는 클라우드 소스와 같은 문제 영역의 데이터 자체를 연구하는 방향으로 진행되고 있다.

1.2 Time series related researches

태양광, 풍력 발전과 같은 재생 에너지의 생산에 대한 예측모델로서 선형회귀법(Regression)을 사용한 연구가 진행되었다[11, 12]. 이 연구에서는 다양한 측정값과 비용함수들을 수식(1)과 같이 정리하였으나, 계수와 허용오차를 결정하는 방법이 없어 시계열 데이터의 클리닝에 활용하기가 어렵고 메모리 효과를 고려하지 않았다.

$$\tilde{x_t} = \sum_{i=1}^{m} \Phi_i x_{t-i} + w_t = \Phi_1 x_{t-1} + \Phi_2 x_{t-2} + ... + \Phi_m x_{t-m} + w_t \quad (1)$$

공장의 센서 데이터에서 발생이 되는 다양한 노이즈를 클리닝하는 연구도 수행되었다[13]. 이 연구에서는 윈도우의 개념을 도입하여 회귀방법의 하나인 선형회귀방법인 LSM(Least Square Method)으로 다음 예측값을 계산하고 물리적으로 불가능한 데이터를 오류로 판단하였다. 과거의 데이터로 오류를 발견할 수는 있으나 예측값으로 보정값을 제시하는 경우 다음에 언급할 오버런에 의하여 노이즈가 후속 단계에서 전파되는 문제 발생한다.

III. Cleaning Noises from Time Series Data with Memory Effects

1. Memory effect of sensor data

머신 러닝에서는 대량의 데이터가 필수적으로 필요하며, IoT의 센서 데이터는 대량의 데이터를 확보할 수 있는 가장 중요한 자원이다. 이러한 센서 데이터들은 대부분 메모리 효과를 갖는 시계열 데이터의 특징을 갖는다. 측정값은 특정 범위 내에 단순한 랜덤 값이 아닌 과거의 데이터에 부분 종속된다. 예를 들어 보일러의 수온은 0~100도 사이의 범위 내의 값을 갖으나, 환경에 따라서 물리적으로는 t_{i+1}시간에 따른 온도는 t_i시간의 온도보다 -d~ +d 이상으로 변경될 수 없는 제약조건을 가질 수 있다. 본 연구에서는 현재의 측정값이 이전 측정값에 부분적으로 종속되는

현상을 메모리 효과(Memory Effects)라고 정의하였다. 이를 이용하여 보다 효과적으로 노이즈를 검출할 수 있다.

자동차의 OBD(On-Board Diagnostics)[14]에서 생성하는 변속기 오일의 온도, 기상 측정 장치에서 획득할 수 있는 온도와 습도, 태양전지의 충전량 및 스마트 팩토리의 여러 종류의 센서 등 에서 수집되는 데이터 들을 메모리 효과를 같은 시계열 데이터의 예로 생각해 볼 수 있다.

2. Issue about window size

Fig. 3의 노이즈는 수온의 범위 내에 존재하므로, 기존 수온의 범위를 확인하는 방법으로는 노이즈임을 알 수 없지만, 이전 데이터들에 선형 회귀법을 적용하여 예측값을 계산하고, 물리적으로 불가능한 차이가 존재하므로 이 데이터가 노이즈로 판단할 수 있다. 또한 선형 회귀방법의 예측값으로 노이즈를 대체할 수 있다. Fig. 4와 같이 각 시간별 온도가 노이즈임을 확인하기 위해 윈도우를 하나씩 반복적으로 이동시켜 계산을 수행시킨다.

Fig. 4에서 첫 번째 윈도우는 [0, 1, 2], 두 번째 윈도우는 [1, 2, 3]을 세 번째 윈도우는 [2, 3, 4]일 경우를 보여준다. 이를 일반화를 하여 정리하면 수식(2)와 같다.

$$window(x_i) = [x_{i-w}, x_{i-w+1}, \cdots, x_{i-1}] \quad where \quad w\,is\,window\,size \tag{2}$$

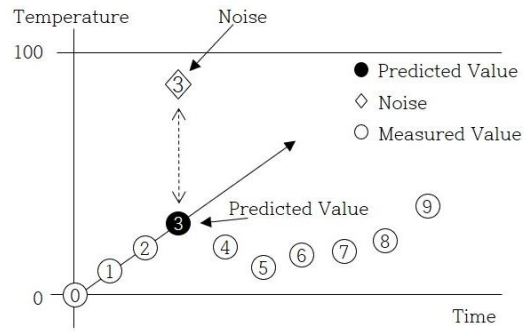

Fig. 3. Noise Detection using Liner Regression

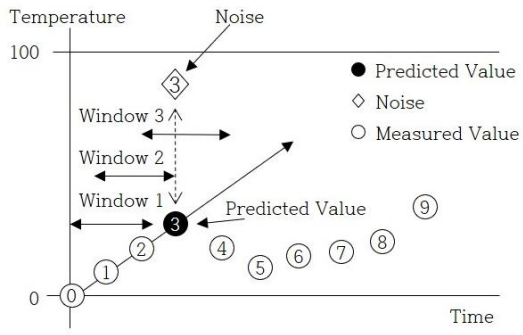

Fig. 4. Noise Identification using Window Size

지금까지 설명한 방법을 실제로 적용해보면 윈도우 크기에 따라 문제가 발생될 수 있다. 윈도우의 크기는 얼마나 많은 과거 데이터를 다음 측정값의 예측에 활용할 것인가를 의미한다. 예를 들어, Fig. 5에서 윈도우의 크기가 3인 경우, 선형 회귀법을 사용하여 x_3 가 노이즈임을 판단할 수 있다. 그러나 윈도우가 4개인 경우 x_3 가 시그널로 인식되어 노이즈임을 파악할 수가 없다. 따라서 윈도우의 크기에 따라, 비정상 데이터가 식별되지 않을 수도 있으므로 적합한 윈도우 크기를 결정하는 것이 중요함을 알 수 있다.

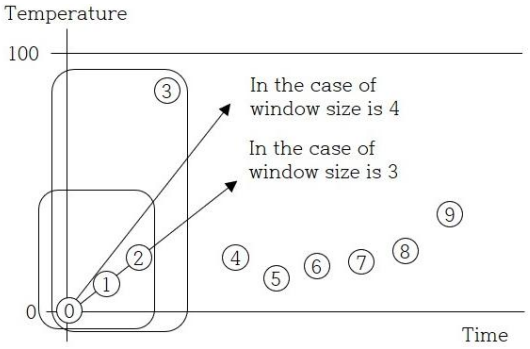

Fig. 5. Noise Detection is depends on Window Size

3. Issue about overrun

Fig. 6는 과거의 데이터만을 고려해서 노이즈 검출하는 경우에 발생되는 문제점을 보여준다. 그림에서 x_3가 노이즈임을 식별하여 예측값으로 대체하였다. 다음 단계인 Fig. 7은 x_4가 시그널임에도 불구하고 알고리즘으로 보면 노이즈로 검출한다. 이런 현상이 지속적으로 반복되어 이후 다수의 정상 데이터를 노이즈로 판단하는 오류가 발생하며 이러한 현상을 본 연구에서는 오버런 현상이라고 정의하였다. 보정값이 노이즈보다 더 정확하지만 노이즈 여부를 판단할 대상의 이전 데이터들만을 사용해서 문제가 발생한다. 그러므로 오버런 현상을 해결하기 위해 본 연구에서는 과거의 데이터뿐만 아니라, 미래의 데이터까지 포함하는 예측값을 계산하는 방법을 제안하였다.

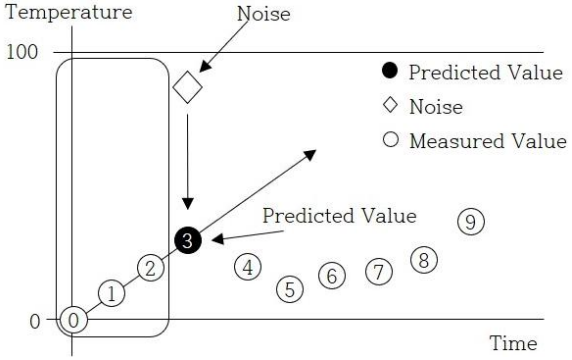

Fig. 6. After the Noise is Replaced by a Predicted Value

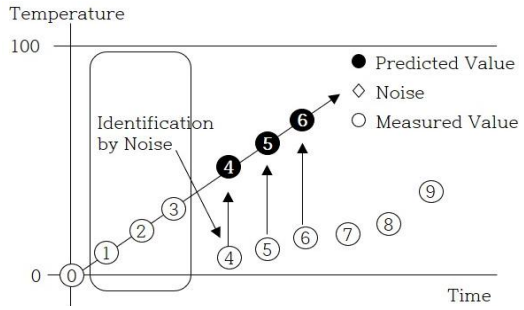

Fig. 7. Overrun Effect

오버런 문제의 해결을 위해 지연(Delay)의 개념을 도입하였다. 지연이 0 라는 것은 값을 예측할 때 미래 데이터가 전혀 고려되지 않음을 의미하고, 지연의 값이 증가할수록 고려되는 미래 데이터의 분량이 증가함을 의미한다. 예를 들어 윈도우 크기가 5이고 분석 대상이 x_i인 경우에 delay = 0이면 [x_{i-5}, x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}] 선형 회귀의 입력 값이며 delay = 1이면 [x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}, x_i] 입력 값이 된다. 이를 일반화하여 수식(3)으로 정리하면 다음과 같다.

$$\begin{aligned} &window(x_i) \\ &= \left[x_{i-w+0+d},\, x_{i-w+1+d}, \cdots, x_{i-1+d}\right] \\ &where \\ &\quad w\ is\ window\ size \\ &\quad d\ is\ delay \end{aligned} \tag{3}$$

Fig. 8과 같이 x_3의 예측을 위해서 x_2, x_3, x_4를 사용하였고, Fig. 9는 x_4의 예측에 x_3, x_4, x_5를 사용하여 오버런 현상을 해결할 수 있음을 보여준다.

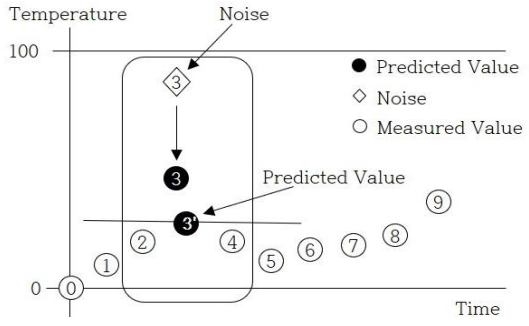

Fig. 8. Prediction using Delay

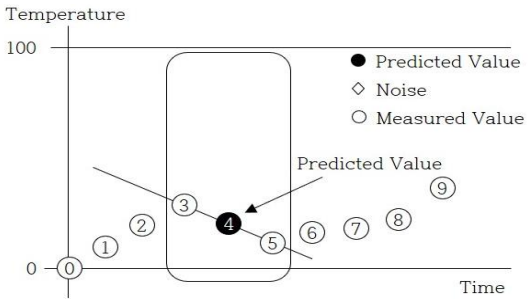

Fig. 9. Resolution of the Overrun Effect

IV. Simulation & Analysis

1. Simulation model

시뮬레이션은 특정 시뮬레이션 도구의 기능이 필요하지 않으므로 파이선 3.7을 사용하여 자체적으로 작성하였다. 아나콘다를 기반으로 노이즈 생성을 위해 NumPy를 데이터 처리를 위해 Pandas를 사용하였다. 그리고 그래프 생성으로 matplotlib를 사용하였으며 선형회귀법 적용을 위해 sklearn의 liner model을 사용하였다. 하드웨어 환경은 윈도우 기반의 i7 CPU를 사용하였다.

시뮬레이션을 통해 윈도우의 크기와 지연과 노이즈 검출 정확도와의 연관성을 확인해 보았다. 시뮬레이션 데이터는 프로그램에서 합성하여 생성하였다. 일종의 뮤테이션 기법[15, 16]을 사용하였다. 정상 시그널에 노이즈를 추가하여 제안된 알고리즘이 얼마나 정확하게 노이즈를 검출할 수 있는지를 확인해 보았다.

Fig. 10은 전체 시뮬레이션 모델을 보여준다. 시그널 생성기는 시계열 데이터의 특징을 담을 수 있도록 랜덤 함수를 사용하여 생성하며 최대 signalDelta 이내로 차이 값을 줄 수 있도록 하였다. 노이즈 생성기는 최대 noiseDelta 이내의 노이즈 값을 임의로 생성한다. 비정상 데이터 생성기는 시그널 데이터와 노이즈 데이터를 결합하여 비정상 데이터 생성한다. 검출기는 위에서 설명한 알고리즘을 통해 비정상 데이터에서 노이즈를 제거하여 수정된 데이터를 생성한다. 시뮬레이션 드라이버는 다양한 인자들에 대해 시뮬레이션 수행시키며 지정된 인자들을 사용하여 다양한 시뮬레이션 결과를 얻기 위해 시뮬레이션을 수행시킨다. 분석기는 수정된 데이터의 정확성을 계산하여 결과를 파일로 저장한다.

2. Simulation of the overrun effect

Fig. 11은 비정상 데이터를 단계별로 식별하는 과정을 보여준다. 첫 번째 그래프는 생성된 시그널과 노이즈를 보여주고 두 번째 그래프는 합성된 비정상 데이터를 보여준다. 마지막 그래프는 클리닝된 결과를 보여준다. 총 5개의 노이즈가 입력되었으나, 3개만 찾아내어 제거하였고, 실 데이터와 큰 차이가 없는 2개의 노이즈 무시되었다. 무시된 노이즈는 이후 프로세스에 큰 영향을 미치지 않으므로 문제가 없다고 판단할 수 있다. 본 시뮬레이션은 noOfData = 50, signalDelta = 3, detectDelta = 7, noiseDelta = 20, 그리고 errorRate = 10%인 것을 가정하여 실행되었다.

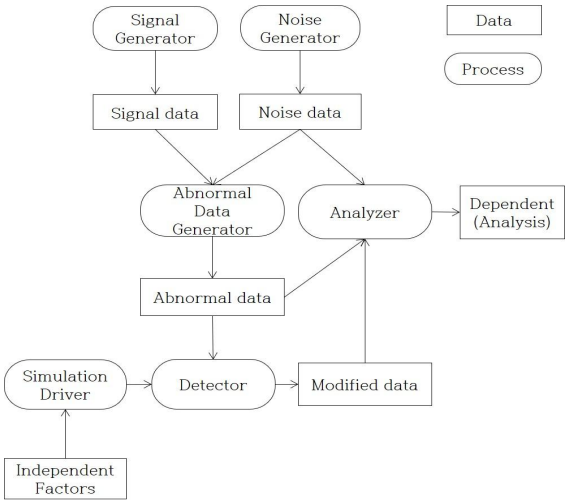

Fig. 10. Simulation Model

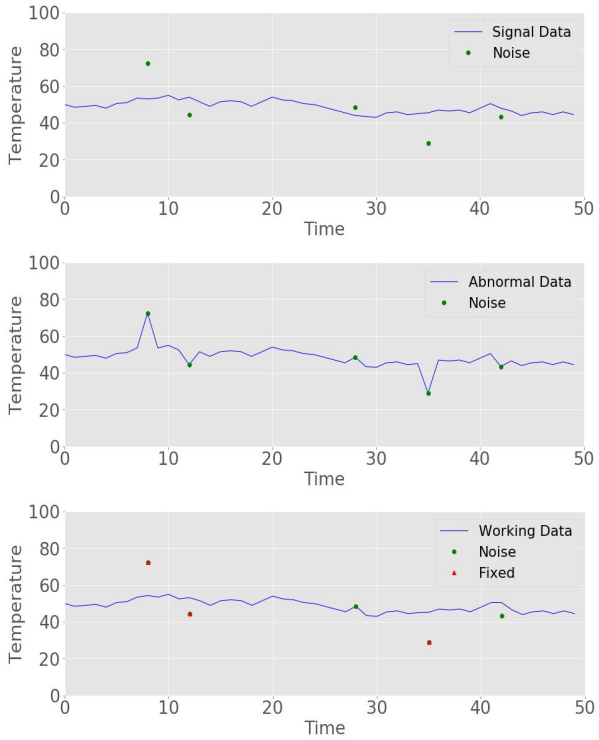

Fig. 11. Noise Cleaning Example

이 후 시뮬레이션은 noOfData = 500, signalDelta = 3, detectDelta = 7, noiseDelta = 20, 그리고 errorRate = 5% 인 것을 가정하여 실행되었다.

Fig. 12는 생성된 비정상 데이터를 보여주고, Fig. 13은 윈도우 크기가 5, 지연이 1로 설정하여 보정된 결과를 보여준다. 이 경우 윈도우의 크기에 비해 지연이 작으므로 과거 데이터의 반영 비율이 높으며 과거의 데이터가 많이 고려되고, 노이즈가 전파되고 누적되어서 200개 이상에서는 오버런 현상이 발생됨을 알 수 있다.

Fig. 14는 윈도우 크기가 5, 지연이 3인 경우의 시뮬레이션 결과를 보여준다. 지연을 이전보다 크게 설정하여 분석 대상 이전과 이후의 데이터를 균형적으로 포함하면, 오버런 발생이 방지됨을 알 수 있다.

따라서 윈도우의 크기와 지연에 따라 정확도가 변화될 수 있음을 확인할 수 있다. 노이즈 검출을 정확도 계산은 비정상 데이터를 보정한 결과와 원래 시그널 데이터의 차이를 계산하였다. 계산식은 가장 많이 사용하는 아래의 수식(4)와 같은 RSS(Residual sum of square)를 비용함수[17, 18]로 사용하였다.

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$
$$where$$
$$cost\ is\ cost\ value$$
$$m\ is\ the\ number\ of\ data$$
$$H\ is\ the\ Hyperthesis\ function$$
$$y\ is\ the\ label\ of\ x \quad (4)$$

Table. 1은 전체 시뮬레이션 결과의 일부이다. 각 윈도우 크기와 지연에 대한 노이즈 검출 정확도와 삽입된 노이즈들을 얼마나 찾아내는가를 확인하기 위한 노이즈 검출 횟수도 계산하였다. 시그널과 노이즈의 차이가 적다면 노이즈로 검출되지 않을 수 있으나, 노이즈 검출 횟수가 실제 적용에서는 큰 영향을 주지 않으며, 실제 적용에 있어 중요한 것은 비용함수다.

Table. 2는 각 윈도우 크기 별로 가능한 모든 지연에 대한 평균 비용을 보여주며, Fig. 15는 이를 그래프로 표현한 것이다. 전반적으로 윈도우 크기가 증가할수록 평균 비용이 감소한다는 것을 알 수 있다. 호출되는 라이브러리의 제약으로 그 이상의 시뮬레이션을 수행할 수 없었으나, 윈도우의 크기를 과도하게 설정할 경우 정확도가 저하될 수 있음을 예측할 수 있다. 윈도우 크기가 6일 때 갑자기 비용이 증가한 이유는 지연이 1일 때 초기부터 오버런 현상이 발생하기 때문이다. 이는 통계에 의한 결과로 7 이상에서는 큰 문제가 없음을 보여준다.

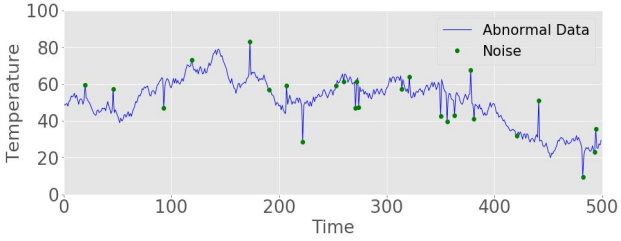

Fig. 12. Generated Abnormal Data

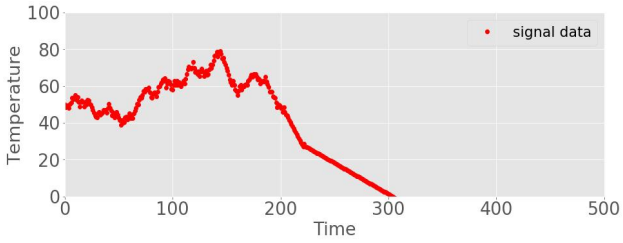

Fig. 13. Case of Window Size is 5 and Delay is 1

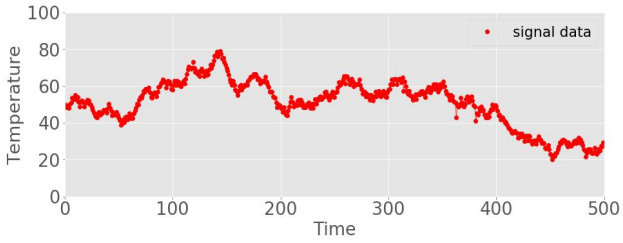

Fig. 14. Case of Window Size is 5 and Delay is 3

Table 1. Average Costs by Window Size and Delay

No	WinSize	Delay	Cost	Actual NoOfDetected
1	4	0	18909.96	8
2		1	15130.34	4
3		2	0.878524	10
4		3	1.158857	10
5	5	0	10857.36	3
6		1	2571.041	3
7		2	0.891394	10
8		3	0.875384	10
9		4	1.126275	10
10	6	0	53012.1	2
11		1	45609.68	2
12		2	0.258836	8
13		3	0.255503	8
14		4	0.267594	8
15		5	0.253228	8
.....				
31	9	0	0.510524	8
32		1	0.501697	8
33		2	0.265574	8
34		3	0.258545	8
35		4	0.293307	8
36		5	0.293531	8
37		6	0.291472	8
38		7	0.49451	9
39		8	0.486676	9

Table 2. Average cost by Window Size

Window Size	Average
4	8510.5856
5	2686.2595
6	16437.136
7	0.286060
8	0.351989
9	0.377315
10	0.394812
11	0.405083
12	315.86181
13	269.74870
14	0.525888
15	0.589917

Fig. 16과 Fig. 17은 각 윈도우 크기에 대하여 지연 값에 따른 평균 비용을 보여준다. 윈도우의 크기가 작은 경우(4~7) 오버런 현상에 의해 정확도가 저하되지만, 지연을 증가시켜 해결할 수 있음을 알 수 있다. 윈도우 크기가 큰 경우(12~15)는 작은 경우보다 매우 개선된 결과 값을 보여주며, 더 우수한 정확도를 얻으려면 역시 적절한 지연 값을 설정할 필요가 있다.

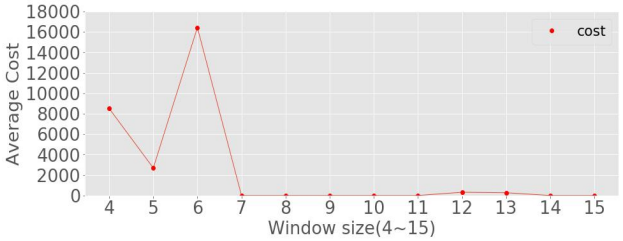

Fig. 15. Average Cost by Window Size

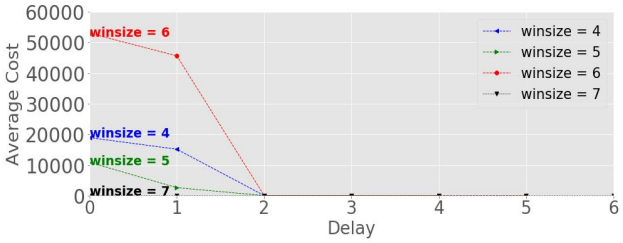

Fig. 16. Average Cost by Delay 1

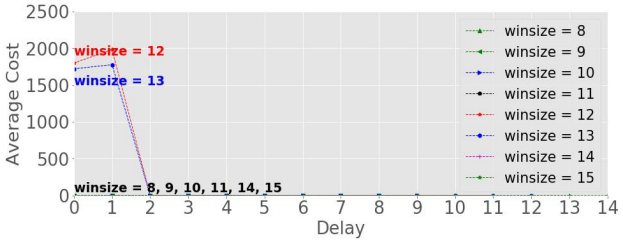

Fig. 17. Average Cost by Delay 2

시뮬레이션의 결과를 정리해 보면, 데이터의 통계적 특성에 따라 적합한 윈도우의 크기가 달라질 수 있다. 따라서 보다 정확도를 향상시키기 위해서는 실제 측정 데이터에서 노이즈와 시그널을 분리하고 본 시뮬레이션을 수행시켜 최적의 윈도우 크기를 찾아낼 수 있다. 지연은 대부분 윈도우 크기의 절반에 해당하는 값을 설정하면 정확도를 높힐 수 있음을 알 수 있다.

V. Conclusions

본 연구에서는 IoT 환경에서 발생이 될 수 있는 다양한 센서 데이터로부터 발생이 되는 시계열 데이터의 노이즈를 감지하고 제거를 위한 방법을 제안하였다. 주요 제안방법은 선형회귀법을 사용하여 시스템이 데이터 과학자들의 개입 없이 반복적으로 노이즈를 자동적으로 찾아낼 수 있도록 하였고 노이즈를 대체할 수 있는 데이터를 제공하여 학습 데이터를 클리닝 하였다.

제안된 방법의 효과를 검증하기 위해서 시뮬레이션을 수행하였으며 최적의 클리닝 결과를 얻을 수 있는 인자들의 결정 방법을 확인하였고, 적절한 윈도우 사이즈와 지연을 통해 오버런 현상을 방지 할 수 있다는 사실을 알아낼 수 있었다.

본 연구의 예에서 입력 벡터는 시간이고 출력 값의 레이블은 수온이다. 연료의 분사량, 외부 온도 및 습도 등과 같은 자세한 입력 벡터를 확보할 수 있다면 보다 정확한 예측을 딥러닝을 통해 알아낼 수 있다. 그러나 현실적으로 이러한 자세한 정보를 확보하거나 비용이 많이 요구되는 경우가 있다. 본 연구에서는 자세한 입력 벡터가 없어도 메모리 효과를 적용하여 향상된 결과를 얻을 수 있었다. 본 연구는 온도에 대한 것만 아니라 습도, 광량 등 다양한 메모리 효과를 갖는 시계열 데이터들에 적용할 수 있다.

향후 에는 강화학습을 이용하여 윈도우 크기와 지연 인자를 찾는 방법을 연구할 필요가 있다.

ACKNOWLEDGEMENT

This research was supported by Kumoh National Institute of Technology (2017-104-065).

REFERENCES

[1] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, “Helix: Holistic optimization for accelerating iterative machine learning,” Proceedings of the VLDB Endowment, pp. 446-460, Dec. 2018.

[2] S. D. Nunes, P. Zhang, J. S. Silva, “A survey on human-in-the-loop applications towards an internet of all,” IEEE Communications Surveys & Tutorials, 17(2), pp. 944-965, Feb. 2015. DOI:10.1109/COMST.2015.2398816

[3] C. Xue, J. Yan, R. Yan, S. M. Chu, Y. Hu, Y. Lin, “Transferable AutoML by Model Sharing Over Grouped Datasets,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9002-9011, June. 2019. DOI:10.1109/CVPR.2019.00921

[4] M. Terry, D. Sculley, N. Hynes, “The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets,” Machine Learning Systems Workshop at NIPS. 2017.

[5] Y. Roh, G. Heo, and S. E. Whang, “A survey on data collection for machine learning: a big data - ai integration perspective,” IEEE Tran. on Knowledge and Data Engineering, June. 2019. DOI:10.1109/TKDE.2019.2946162

[6] Rekatsinas, Theodoros, et al. “Holoclean: Holistic data repairs with probabilistic inference,” arXiv preprint arXiv:1702.00820, Feb. 2017.

[7] S. Krishnan, J. Wang, E. Wu, M. Franklin, J, K. Goldberg, “Activeclean : Interactive data cleaning for statistical modeling,” Proceedings of the VLDB Endowment, 9, pp. 948-959, Aug. 2016.

[8] Krishnan, Sanjay, et al. “Boostclean: Automated error detection and repair for machine learning,” arXiv preprint arXiv:1711.01299, Nov. 2017.

[9] K. H. Tae, Y. Roh, Y. H. Oh, H. Kim, & S. E. Whang, “Data cleaning for accurate, fair, and robust models: A big data-AI integration approach,” In Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning, pp. 1-4, June. 2019.

[10] M. Dolatshah, M. Teoh, J. Wang, and J. Pei, “Cleaning crowdsourced labels using oracles for statistical classification,” Proceedings of the VLDB Endowment, Vol. 12, pp. 376-389, Dec. 2018.

[11] Mahmoud Ghofrani and Musaad Alolayan, “Time Series and Renewable Energy Forecasting,” intechopen, Dec. 2017. DOI:10.5772/intechopen.70845

[12] K. Bandara, C. Bergmeir, S. Smyl, “Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach,” Expert Systems with Applications, Vol. 140, Feb. 2020.

[13] Won-chang Lee, Jae-Han Cho and LeeSub Lee, “Time Series Abnormal Data Detection for Smart Factory,” International

Journal of Control and Automation, Vol. 11, No. 1, pp. 91-98, Jan. 2018.

[14] J. Zaldivar, C. T. Calafate, J. C. Cano, and P. Manzoni, “Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones,” IEEE Trans. on Local Computer Networks, pp. 813-819, Oct. 2011.

[15] Smith, H. Ben, Laurie Williams, “On guiding the augmentation of an automated test suite via mutation analysis,” Empirical software engineering, pp. 341-369, June. 2009.

[16] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, Y. Wang, “Deepmutation: Mutation testing of deep learning systems,” IEEE Trans. on Software Reliability Engineering, pp. 100-111, Oct. 2018.

[17] C. Finn, S. Levine, P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” In International conference on machine learning, pp. 49-58, June. 2016.

[18] F. P. Luus, B. P. Salmon, F. Van den Bergh, B. T. J. Maharaj, “Multiview deep learning for land-use classification,” IEEE Geoscience and Remote Sensing Letters, pp. 2448-2452, Oct. 2015.

Authors

Jae-Han Cho received the Ph.D. student in department of computer engineering at Kumoh National Institute of Technology, Gyeongsangbuk-do, Korea. He received B.S and M.S. degree in Computer Engineering from Kumoh National Institute of Technology, Korea in 2011, 2014, respectively. Jae-Han Cho is interested in Software Testing and Data Engineering and Deep-Learning.

Lee-Sub Lee, he is an Associate professor of Department of Computer Engineering at the Kumoh National Institute of Technology Gyeongsangbuk-do, Korea. He received B.S. in Mathematics and M.S. degree in Computer Engineering from Sogang University, Seoul, Korea. He received his Ph.D. in Computer Engineering from Korea University, Seoul, Korea. He has worked as a senior researcher at Samsung SDS 1990 to 2004. Lee-sub Lee is research work has been on the Software Engineering and Database System. His recent interest focuses on Software Testing, Deep-Learning and Bigdata,