2020년 맞춤형 기술인재 양성과정 빅데이터분석가

승강기 운행 분석 보고서

분석 주제명 : 승강기 운행데이터 분석을 통한 운행 효율성 제고

참여자: 민진원, 박승섭, 이지성, 김소연, 안석현

sba 아카데미 • CSLEE

목 차

1. 분석 개요

가. 분석 배경

□ 1)오늘날 하루에 10억 명 이상의 사람들은 학교/집/직장 등 여러 곳에서 승강기를 이용하고 있으며, 2)일반인 하루 평균 4회 이용하고 있다. 그만큼 승강기는 사람들과 밀접한 관계를 이루고 있으며, 삶의 질을 높이는 중요한 요소이다.

□ 승강기마다 용도에 맞는 효율성 평가 기준이 있으나, 현재 용도별 대부분 건물들의 승강기 운행 효율성은 이에 미치지 못하고 있다. 특히 출·퇴근 시간 및 식사 시간 등 승강기 이용이 집중되는 특정 시간에는 대기시간이 길어지고 사용자들의 불편함을 유발한다.

□ 위와 같은 점을 개선하기 위한 노력 중 하나로 IOT 기능이 탑재된 승강기가 있으나, 이는 중소기업에서는 비용부담으로 활용하기 어려운 실정이다. 그래서 이러한 비용적인 제한을 극복하고 승강기 사용자들의 쾌적한 이용 즉, 효율적인 승강기 운행을 하고자 하는 것이 본 프로젝트의 취지다.

빅데이터 분석 통한 승강기 운행 최적화

[그림 1] AS-IS / TO-BE

1) https://www.youtube.com/watch?v=B7KfXZiVpOE&feature=youtu.be thyssenkrupp Elevator 출처
2) http://eungaram.net/index.php?cate=002001 ㈜은가람 통계자료

나. 분석 목적 및 방향

1) 한 아파트의 운행데이터를 통해, 사용자 편의성을 고려한 맞춤형 승강기 운행 패턴 및 알고리즘을 개발하여 효율적인 운행을 목적으로 하고자 한다.

2) 이러한 개발을 통해 건물의 용도별 (주거/사무/공장) 최적화된 운행 패턴, 입지별 (베드타운/역세권/주상복합) 최적화된 운행 패턴, 층별 (초고층/고층/중층/저층) 최적화된 운행 패턴 제공하여 효율적인 승강기 운행이 사용자의 쾌적한 삶에 도움이 되는 방향으로 설정을 하였다. [그림 1]와 같이 초기 데이터 전처리, 데이터 분석 실행 가능, 최종 효율화 알고리즘 적용하고자 한다.

[그림 2] 승강기 운행 효율화 알고리즘 개발을 위한 주요 3가지 개발요소

[표 1] 운행 효율화 알고리즘 개발을 위한 주요 3가지 개발요소

구분	개발요소	방법
전처리	-운행정보 전처리 및 정규화 -분석 설정	-운행데이터 마이닝을 위한 용도별 라이브러리 정의 -정의별 전처리 방법, 분석, 가시화 설정
분석 실행	-분석 실행 서비스 및 분석 모델 개발	-TensorFlow 플랫폼과 파이썬을 활용한 분석 실행 응용프로그램 개발 -실행 상태 조회
알고리즘 적용	- 알고리즘 적용	-패턴 분석된 맞춤형 알고리즘을 승강기 제어반에 적용

2. 분석 데이터

가. 활용 데이터

1) 주거용(고층/저층 아파트, 빌라 등), 사무용(고층/중층/저층 오피스빌딩)으로 분류해서 운행데이터 수집/저장/분석

[표 2] 활용 데이터

***지역별, 시간대별 패턴 분류 수집/저장/분석**

활용 데이터	구분	중요도	생성 주기	지역 속성	데이터 소스
승강기 운행데이터	㈜대명 엘리베이터	필수	일간	건물별	2018년11월16일 ~ 2018년11월23일 주간 운행 데이터

나. 데이터 상세 설명

1) 운행데이터의 Raw 데이터 분석을 통한 데이터 구분

DATA	DATE	TIME
01005aa5040000000500000064003c0000000800000000080000700200020004200 3100eb02	20181116	103103
01005aa5040000000500000064003c0000000800000000080000700200020004200 3100eb02	20181116	103104

[그림 3] RAW DATA

raw data (Call Status)

0400 0000 0000 0000 0000 0000 0000 0000 0200 0000 0000 0000 0001 0000 ···

Car Call / Hall Call Up / Hall Call Down

Raw Data	Binary Data	Floor	Raw Data	Binary Data	Floor
0100	0000 0001 0000 0000	B1	4000	0100 0000 0000 0000	6F
0200	0000 0010 0000 0000	1F	8000	1000 0000 0000 0000	7F
0400	0000 0100 0000 0000	2F	0001	0000 0000 0000 0001	8F
0800	0000 1000 0000 0000	3F	0002	0000 0000 0000 0010	9F
1000	0001 0000 0000 0000	4F	0004	0000 0000 0000 0100	10F
2000	0010 0000 0000 0000	5F	0000	0000 0000 0000 0000	No Call

Raw Data to Floor Table

[그림 4] Call Status (raw data)

raw data (No Call Status)

0100 5*aa*5 0400 0900 0500 0000 6400 3*c*00 0000 1000 0000 0000 8000 0700 2000 2000 2000 3900 *e*202

Current Floor / Elevator Status / Door Status / Current Floor (Elevator Display)

0100 0101 0000 0000 (①②③④)

① Current Door Status: 00 : moving, 01 : closed, 10 : opened

② Elevator Status: 0 : running, 1 : stopping

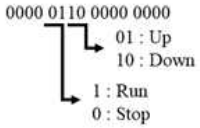
③ Next Door Status: 00 : all off, 01 : up lamp on, 10 : down lamp on

④ Lamp Status: 001 : not moving, 010 : open, 100 : close

[그림 5] No Call Status (raw data)

Raw Data	Binary Data	Door	Elevator	Up Lamp	Down Lamp
2800	0010 1000 0000 0000	Opening	Door Opening		
2900	0010 1001 0000 0000	Opening	Door Opening	on	
2a00	0010 1010 0000 0000	Opening	Door Opening		on
3000	0011 0000 0000 0000	Closing	Door Closing		
3100	0011 0001 0000 0000	Closing	Door Closing	on	
3200	0011 0010 0000 0000	Closing	Door Closing		on
4500	0100 0101 0000 0000	Close	Running	on	
4600	0100 0110 0000 0000	Close	Running		on
6400	0110 0100 0000 0000	Close	Stop		
6500	0110 0101 0000 0000	Close	Run ⇔ Stop	on	
6600	0110 0110 0000 0000	Close	Run ⇔ Stop		on
6800	0110 1000 0000 0000	Close	Close to Open		
6900	0110 1001 0000 0000	Close	Close to Open	on	
6a00	0110 1010 0000 0000	Close	Close to Open		on
7000	0111 0000 0000 0000	Close	Close to Close		
7100	0111 0001 0000 0000	Close	Close to Close	on	
7200	0111 0010 0000 0000	Close	Close to Close		on
A800	1010 1000 0000 0000	Open	Open to Open		
A900	1010 1001 0000 0000	Open	Open to Open	on	
aa00	1010 1010 0000 0000	Open	Open to Open		on
B000	1011 0000 0000 0000	Open	Open to Close		
B100	1011 0001 0000 0000	Open	Open to Close	on	
b200	1011 0010 0000 0000	Open	Open to Close		on

[그림 6] Raw Data to Door status Table



Raw Data	Binary Data	Elevator	Direction
0000	0000 0000 0000 0000	Stop	Stop
0100	0000 0001 0000 0000	Stop	Up
0200	0000 0010 0000 0000	Stop	Down
0500	0000 0101 0000 0000	Run	Up
0600	0000 0110 0000 0000	Run	Down

[그림 7] Raw Data to Elevator status Table

2) 데이터 정제를 통한 불필요한 이상치 제거

☐ event 중, 사용자가 발생시킨 Event를 의미하지 않는 데이터 삭제

☐ current_floor 중, FALSE, 원본행 데이터 삭제

☐ code_status 중, 비 운행층 설정, 홀짝수층 설정 데이터 삭제

☐ 8일치 운행데이터 60만여건 중, 분석에 사용된 데이터는 57만여건

date	time	state	door_state	event_floor	current_floor	event	code_status
20181116	103103	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103104	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103122	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103123	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103124	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103125	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103126	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103127	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103128	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103129	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103130	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103131	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103132	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103133	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보
20181116	103134	STOP	DOOR 닫힘상태	nan	B1	nan	상태정보

[그림 8] 운행데이터 Data base

3) 정제된 데이터를 2진 코드로 변환

☐ 48자리 2진 코드로 변환

Raw Data		Binary Data
Date (0~14)	Year (5)	00000 ~ 10010
	Month (4)	0001 ~ 1100
	Day (5)	00001 ~ 11111
Time (14~31)	Hour (5)	00000 ~ 10111
	Minute (6)	000000 ~ 111011
	Second (6)	000000 ~ 111011

Date and Time (0 : 31)

[그림 9] Data & Time Raw Data 2진 코드 변환

Raw Data	Binary Data
STOP	001
STOP/DOWN	010
STOP/UP	011
RUN/DOWN	100
RUN/UP	101

Elevator State (31 : 34)

[그림 10] 승강기 상태 2진 코드 변환

Raw Data	Binary Data
Car Call	01
Hall Call Down	10
Hall Call Up	11

Event (46 : 48)

[그림 11] Event Data 2진 코드로 변환

Raw Data	Binary Data
Closing	0001
Closing Down Lamp On	0010
Closing Up Lamp On	0011
Opening	0100
Opening Down Lamp On	0101
Opening Up Lamp On	0110
Close	0111
Close Down Lamp On	1000
Close Up Lamp On	1001

Floor (34 : 38)

[그림 12] state 2진 코드 변환

Raw Data	Binary Data
B1	0001
1F	0010
2F	0011
3F	0100
4F	0101
5F	0110
6F	0111
7F	1000
8F	1001
9F	1010
10F	1011

Floor (38 : 46)

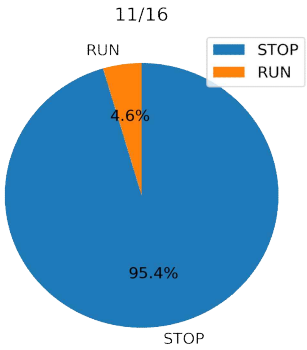
[그림 13] Floor 2진 코드 변환

다. 탐색적 데이터 분석(EDA)

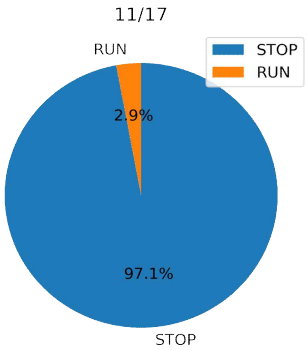
1) 일자별 승강기 유휴시간 및 가동시간

☐ 하루 24시간동안 승강기의 가동시간은 5%를 넘지 않음

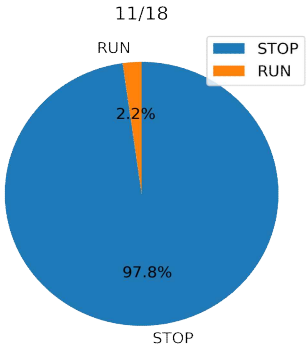
☐ [그림 22]와 같이 분석기간 전체 중 가동시간의 비율은 3.2%에 불과함



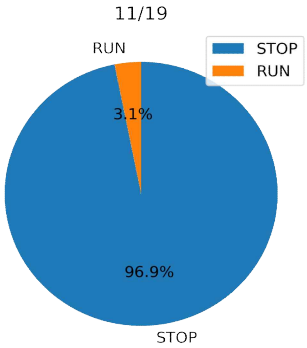
[그림 14] 11/16 유휴•가동시간



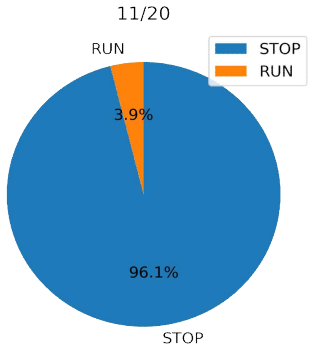
[그림 15] 11/17 유휴•가동시간



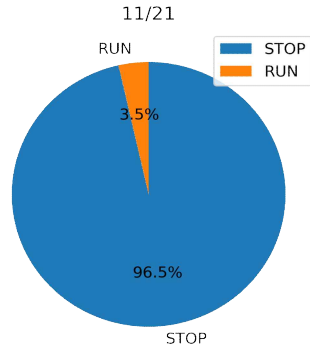
[그림 16] 11/18 유휴•가동시간



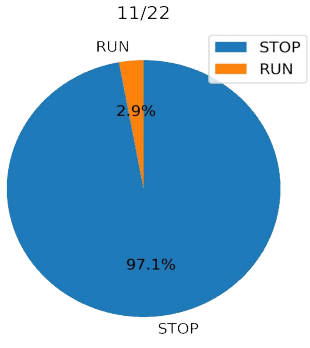
[그림 17] 11/19 유휴•가동시간



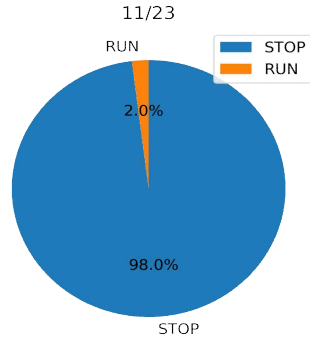
[그림 18] 11/20 유휴•가동시간



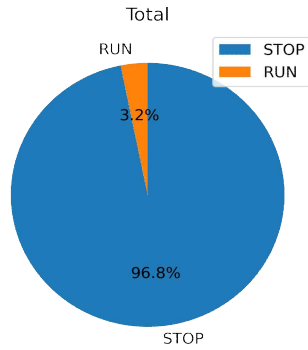
[그림 19] 11/21 유휴•가동시간



[그림 20] 11/22 유휴•가동시간



[그림 21] 11/23 유휴•가동시간

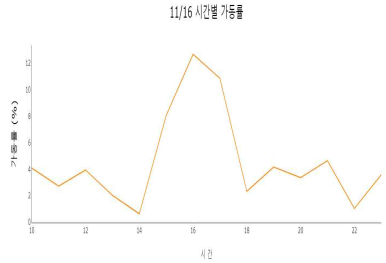


[그림 22] 전체 유휴•가동시간

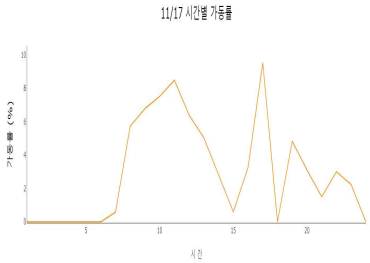
2) 일자별 사용량 추이

☐ 분석기간 내 통상적으로 08시와 17시에 가동률이 가장 높음을 알 수 있음

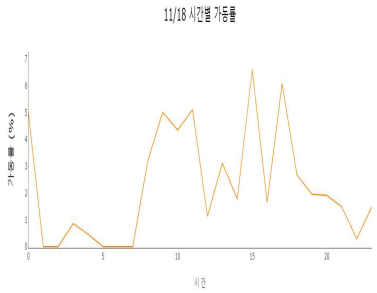
☐ 분석기간 내 시간 별 가동률 변화 패턴이 유사한 것을 알 수 있음



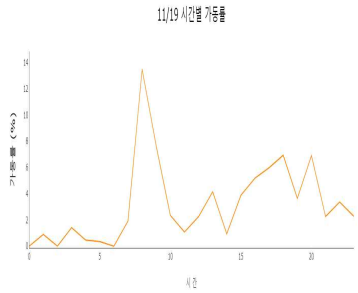
[그림 23] 11/16 시간별 가동률



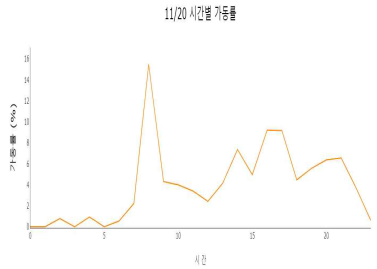
[그림 24] 11/17 시간별 가동률



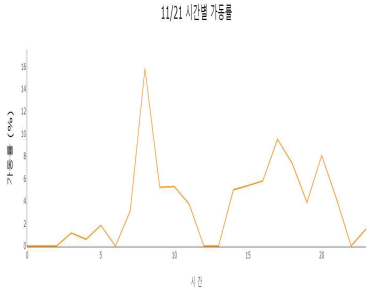
[그림 25] 11/18 시간별 가동률



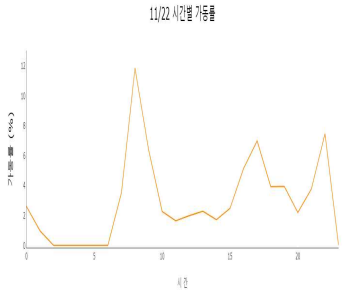
[그림 26] 11/19 시간별 가동률



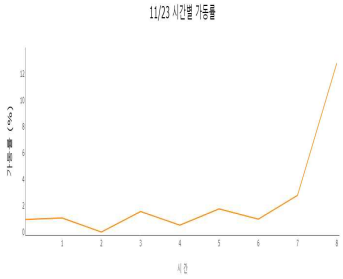
[그림 27] 11/20 시간별 가동률



[그림 28] 11/21 시간별 가동률



[그림 29] 11/22 시간별 가동률

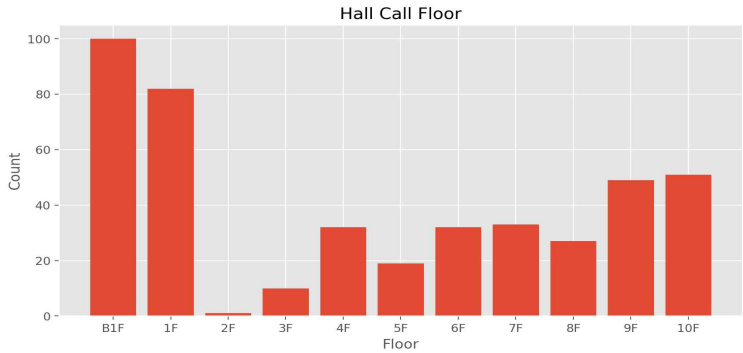


[그림 30] 11/23 시간별 가동률

3) Hall Call(Up, Down) 횟수

☐ [그림 31]을 통해 공동 구역인 지하 1층과 1층의 Call 횟수가 높은 것을 확인할 수 있음

☐ 공동 구역을 제외하고, 9층과 10층의 Call 횟수가 상대적으로 높은 것을 알 수 있음

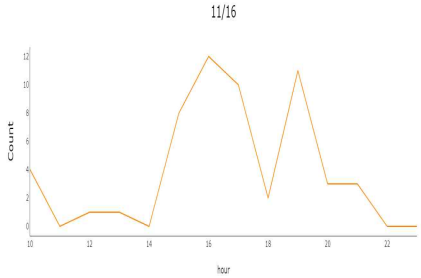


[그림 31] 층별 Call 횟수

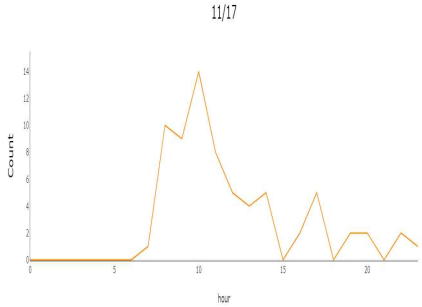
4) 분석기간별 이벤트 패턴 확인

☐ 주말 및 평일에 대하여 다른 패턴이 발생하지 않음

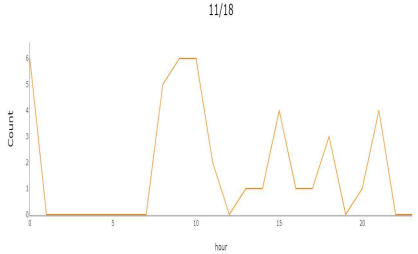
☐ 이는 8일 기간에 대해 일별 구분 없이 동일한 관점 적용이 가능함을 의미함



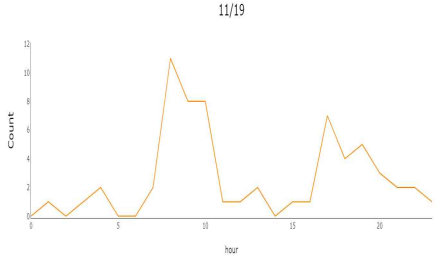
[그림 32] 11/16 시간별 이벤트 발생 횟수



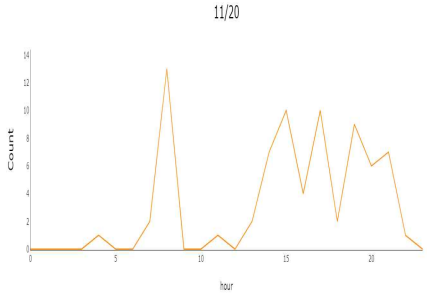
[그림 33] 11/17 시간별 이벤트 발생 횟수



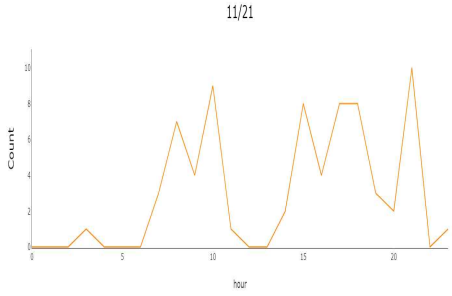
[그림 34] 11/18 시간별 이벤트 발생 횟수



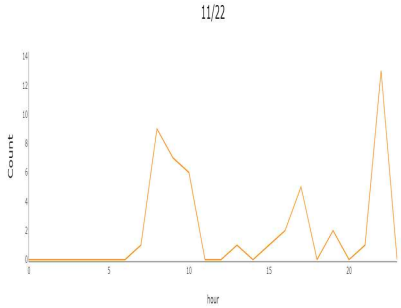
[그림 35] 11/19 시간별 이벤트 발생 횟수



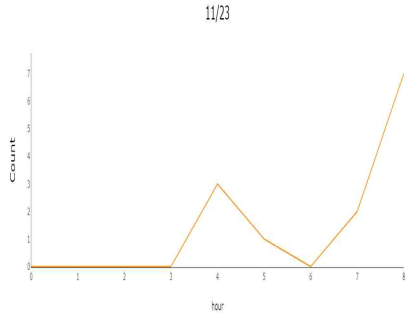
[그림 36] 11/20 시간별 이벤트 발생 횟수



[그림 37] 11/21 시간별 이벤트 발생 횟수



[그림 38] 11/22 시간별 이벤트 발생 횟수



[그림 39] 11/23 시간별 이벤트 발생 횟수

라. 데이터 정의

1) 사용자 대기시간 정의

☐ $v = \frac{t}{s}$**에 의해 엘리베이터가 등속도로 운행**

☐ CarCall의 경우, 승강기 내에서 발생한 Event이므로 사용자 대기시간으로 정의할 수 없음

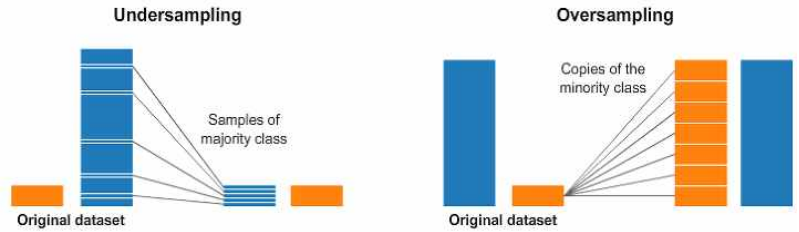
☐ Hall Call 발생층은 승강기 도착층으로 설정

☐ Hall Call 발생 시점 이전 STOP 상태 현재층은 승강기 출발층으로 설정

☐ Hall Call 발생 시점 이전에서 STOP 상태보다 Hall Call Event를 먼저 탐색 시 이 Hall Call Event의 대기시간은 무시

☐ 도착층과 현재층 차이를 Event에서 발생한 사용자 대기시간으로 정의

2) 학습데이터 정의

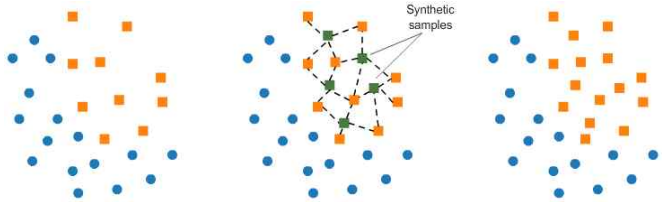


[그림 40] Undersampling and Oversampling

□ 기존 데이터로 인공지능 및 시계열 분석을 진행하는 것은 불가능

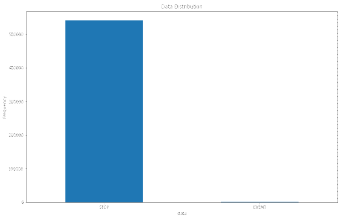
□ 오버 샘플링 기법 중 하나인 SMOTE 기법을 활용하여 데이터의 불균형을 해소하고자 함

- SMOTE (synthetic minority oversampling technique)
  SMOTE는 적은 데이터 셋에 있는 개별 데이터들의 K 최근접 아웃(K Nearest Neighbor)을 찾아서 이 데이터와 K개 이웃들의 차이를 일정 값으로 만들어서 기존 데이터와 약간 차이가 나는 새로운 데이터들을 생성하는 방식

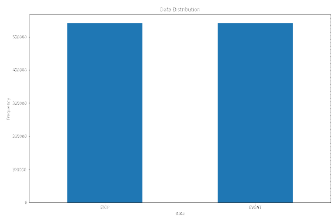


[그림 41] SMOTE

□ 실제 데이터를 기반으로 하여 빈도수를 체크 하고 각 시간 대 층 별 빈도에 따른 샘플링 실시하고 빈도가 가장 높은 층을 Optimal한 위치로 가정



[그림 42] 샘플링 전 데이터 불균형 분포



[그림 43] 샘플링 후 데이터 분포

□ 1시간 단위로, Optimal위치의 등장비율을 통해 정답셋을 생성함

3. 시계열 분석 프로세스

시계열 분석이란 시간의 흐름에 따라 발생하는 시계열 데이터를 통해 미래를 예측하기 위한 분석 방법이다. 앞서 EDA를 통해 승강기 운행 패턴은 요일에 상관없이 유사하며, 승강기 호출 이벤트가 시간별 특정한 패턴을 갖고 발생하는 것을 확인할 수 있었다. 우리는 승강기 운행데이터를 시계열 분석하여 시간별 호출 이벤트가 발생할 층을 예측함으로써 탑승 대기시간을 줄이고자 한다. 시계열 분석 간 주로 사용되는 ARIMA, LSTM, GRU 총 3가지 모델을 적용한다.

가. ARIMA

1) 개념

□ ARIMA(Autoregressive Integrated Moving Average) 모형은 시계열을 예측하는 하나의 접근 방법으로 데이터에 나타나는 자기상관을 표현하는 것에 목적이 있다. 기존의 ARIMA 모형에서 비정상 시계열을 분석할 수 없었지만 ARIMA 모형에서는 정상 시계열화를 통해 분석이 가능하다. 비정상 시계열을 정상 시계열로 변환하는 과정에서 '차분'이라는 개념을 적용한다. '차분'이란 시계열의 수준에서 나타나는 변화를 제거하여 시계열의 평균 변화를 일정하게 만드는 것이다.

□ 모형에서 AR(자기회귀 모형)과 MA(이동평균 모형), ARIMA(자기회귀 누적 이동 평균 모형)등이 있다, 결과값을 도출하는 과정에서 차분 횟수와 ACF(자기 상관 함수), PACF(부분 자기 상관 함수)를 활용하여 ARIMA모형의 모수(p,d,q)를 대입하여 예측 값을 생성한다.

□ AR : 자기회귀-이전 관측값의 오차항이 이후 관측값에 영향을 주는 모형

$$AR(1) : X_t = \phi X_{t-1} + \epsilon_t$$

□ MA : 이동평균-이전의 연속적인 오차항이 관측값에 영향을 주는 모형

$$MA(1) : X_t = \epsilon_t - \beta_1 \epsilon_{t-1}$$

□ ARIMA는 현실에 존재하는 시계열 자료는 불안정하므로, 모형 자체에 비정상성을 제거하는 과정을 포함한 모형

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} - \beta_1 \epsilon_{t-1} - \ldots - \beta_q \epsilon_{t-q}$$

- $\mu = constant$
- $\phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} : ARterms(laggedvaluesofy)$
- $-\beta_1 \epsilon_{t-1} - \ldots - \beta_q \epsilon_{t-q} : MAterms(laggedvaluesofy)$
- $\hat{y}_t = Y_t,\ if\ d = 0$
- $\hat{y}_t = Y_t - Y_{t-1},\ if\ d = 1$
- $\hat{y}_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}),\ if\ d = 2$

2) 분석 과정

□ ARIMA 모수 설정

- auto_arima 함수를 이용한 모수 설정

: auto_arima 함수를 이용하면 가장 적절한 ARIMA 모델 모수 설정이 가능

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
        from statsmodels.tsa.arima_model import ARIMA
        from pmdarima.arima import auto_arima

        series = pd.read_csv('valid.csv')
        series = series["floor"]
        series = series.values.tolist()
        auto_arima(series)

Out[1]: ARIMA(order=(0, 0, 1), scoring_args={})
```

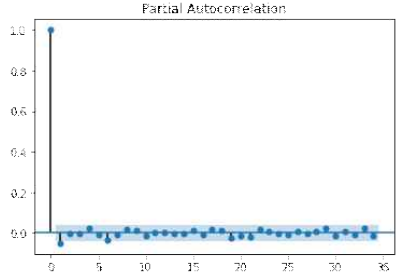
[그림 44] auto_arima

- ACF와 PACF 시각화를 이용한 모수 설정

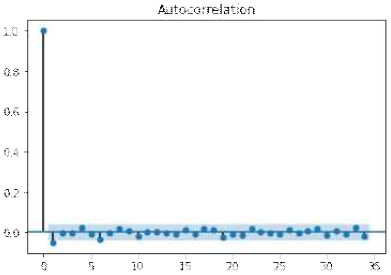
[표 3] ACF와 PACF의 시각화를 이용한 모수 설정

구분	ACF	PACF
AR(p)	지수적으로 감소하거나 소멸하는 사인함수 형태	시차 p 이후 0으로 전달된 형태
MA(q)	시차 q 이후 0으로 전달된 형태	지수적으로 감소하거나 소멸하는 사인함수 형태
ARMA(p,q)	시차(p-q)이후에 지수적으로 감소하거나 소멸하는 사인함수 형태	시차(p-q) 이후에 지수적으로 감소하거나 소멸하는 사인함수 형태

=> ACF와 PACF의 형태를 통한 차분의 여부 및 차수 d를 결정한다.



[그림 45] PACF



[그림 46] ACF

□ ARIMA 모형 구축

'P>|z|' 값이 학습의 적정성을 위해 확인되는 t-test 값이다. 즉, P-value 0.05수준에서 ARIMA(0,1) 모델이 유효함을 알 수 있다.

ARMA Model Results

Dep. Variable:	floor	No. Observations:	2400
Model:	ARMA(0, 1)	Log Likelihood	-7451.635
Method:	css-mle	S.D. of innovations	5.397
Date:	Thu, 08 Oct 2020	AIC	14907.271
Time:	00:36:08	BIC	14918.837
Sample:	0	HQIC	14911.479

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1.floor	0.5065	0.013	37.627	0.000	0.480	0.533

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-1.9742	+0.0000j	1.9742	0.5000

[그림 47] ARIMA(0,0,1) 모델

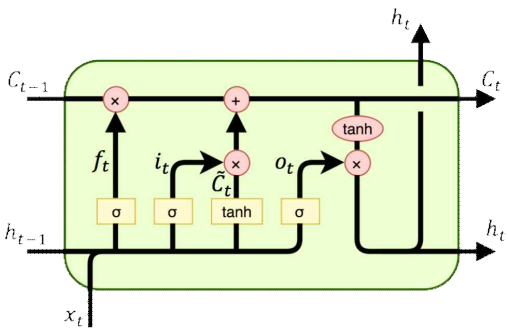
나. LSTM

1) 개념

LSTM이란 시간의 흐름에 따라 변화하는 데이터(Sequence data)를 학습하기 위한 딥러닝 모델로 RNN의 주요 모델 중 하나이다. 직전 데이터뿐만 아니라 과거 데이터를 고려하여 미래의 데이터를 예측하기 위하여 사용한다. RNN은 과거의 데이터를 기반으로 학습하여 최근 데이터를 갖고 현재의 문제를 해결한다. 하지만 위치가 멀어지면 장기 의존성 문제가 발생하여 해결할 수 없게 된다. 즉, 현재로부터 거슬러 올라가는 단계가 많아지면 학습하기에 어려움이 생긴다. 수치로 설명해보면 RNN은 1보다 큰 값들이 계속 발생하면 지속적으로 곱해져 발산하게 되고 1보다 계속 작은 값을 곱하면 0으로 수렴해 사라지게 된다. 이렇게 되면 RNN은 짧은 기억은 가능하지만 긴 기억은 기억하지 못하는 문제가 발생하게 되는데, 이를 장기의존성 문제 (Long-Term-Dependency)라고 한다. LSTM은 이를 개선하여 긴 의존 기간을 필요로 하는 데이터를 학습하는데 효과적인 모델이다.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \qquad f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \qquad i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \qquad o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \qquad C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

[그림 48] LSTM 모델 관계



[그림 49] LSTM 계층 구조

2) 분석 과정

☐ Data import

전처리된 학습데이터를 불러온다.

	floor
0	8
1	10
2	5
3	2
4	5
2396	8
2397	10
2398	9
2399	3

2400 rows × 1 columns

[그림 50] Data import

☐ Scaling

MinMax Scaling을 통해 데이터를 scaling 한다. 이는 데이터의 최댓값이 1, 최솟값이 0이 되도록 변환한다. 이렇게 데이터의 scale을 맞추면 weight의 scale도 일관성 있게 나올 수 있다. scaling 결과 2차원의 numpy ndarray 타입으로 변환된다.

```
array([0.7, 0.9, 0.4, ..., 0.9, 0.8, 0.2])
```

[그림 51] Scaling

☐ Training Set & Test Set Split

학습과 테스트를 위한 트레이닝 셋과 테스트 셋을 구분한다.

```
(2400, 1) 2400 1680 720
```

[그림 52] Training Set & Test Set Split

☐ Pandas Dataframe 변환

정규화가 완료된 데이터들은 다시 Pandas Dataframe 타입으로 변환한다. 타입을 변경하는 이유는 pandas는 시계열 자료에 대한 다양한 기능을 제공하며 RNN을 트레이닝하기 위한 window를 만들기 유용하기 때문이다.

	Scaled
0	0.8
1	0.0
2	1.0
3	0.7
4	0.4

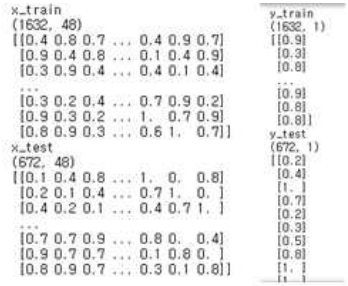
[그림 53] Pandas Dataframe 변환

□ Window 만들기

Pandas shift를 통해 window를 만든다. window는 LSTM을 훈련하기 위한 단위로 고정된 사이즈를 갖으며, window size가 12라면 과거의 시간 데이터 12개를 사용하여 다음 시간 단위의 값을 예측하게 된다. Dataframe의 shift연산을 사용하면 인덱스는 그대로 두고 데이터만 시간 단위로 이동할 수 있어서 데이터를 구성하기에 용이하다.

□ X_training, X_test, Y_training, Y_test 만들기

X_training와 X_test는 각각 1개의 데이터가 window 크기를 가지는 트레이닝 셋, 테스트 셋을 의미하며, Y_training, Y_test는 실제 트레이닝 셋 범위의 실제 값, 테스트 셋 범위의 실제 값을 나타낸다.



[그림 54] Train set & Test set 만들기

□ Ndarray로 변환하기

Dataframe 타입이었던 훈련/테스트 데이터들을 values를 사용하여 Dataframe의 numpy 표현형만 가져온다. 실제 딥러닝 모델의 트레이닝과 테스트로 사용되는 데이터는 일반적으로 numpy의 narray이다.


```
array([[0.4, 0.8, 0.7, ..., 0.4, 0.9, 0.7],
       [0.9, 0.4, 0.8, ..., 0.1, 0.4, 0.9],
       [0.3, 0.9, 0.4, ..., 0.4, 0.1, 0.4],
       ...,
       [0.3, 0.2, 0.4, ..., 0.7, 0.9, 0.2],
       [0.9, 0.3, 0.2, ..., 1. , 0.7, 0.9],
       [0.8, 0.9, 0.3, ..., 0.6, 1. , 0.7]])
```

[그림 55] Ndarray로 변환하기

□ Training set & Test set Reshape

최종 훈련 데이터를 생성하기 위해서는 Keras의 LSTM에서 필요로 하는 3차원 데이터로 변환을 시켜줘야 한다. 따라서 이 형태로 데이터를 reshape해준다.

```
[[[0.4]
  [0.8]
  [0.7]
  ...
  [0.4]
  [0.9]
  [0.7]]
```

[그림 56] Training set & Test set Reshape

□ LSTM 모델 만들기

Keras의 Sequential 모델을 사용하여 LSTM모델을 생성한다.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 20)                1760
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 21
=================================================================
Total params: 1,781
Trainable params: 1,781
Non-trainable params: 0
_________________________________________________________________
```

[그림 57] LSTM 모델 생성

□ Model Fitting

Fit 메소드를 통해 훈련한다. 모델을 훈련한다는 것은 훈련용 데이터 셋을 이용하여 최적화 과정을 통해 모델의 weight를 찾는 것이다. 이때 유사 패턴을 학습한다.

```
Epoch 292/300
1632/1632 [==============================] - 1s 457us/step - loss: 0.0929
Epoch 293/300
1632/1632 [==============================] - 1s 401us/step - loss: 0.0934
Epoch 294/300
1632/1632 [==============================] - 1s 411us/step - loss: 0.0971
Epoch 295/300
1632/1632 [==============================] - 1s 431us/step - loss: 0.0966
Epoch 296/300
1632/1632 [==============================] - 1s 418us/step - loss: 0.0938
Epoch 297/300
1632/1632 [==============================] - 1s 452us/step - loss: 0.0927
Epoch 298/300
1632/1632 [==============================] - 1s 415us/step - loss: 0.0934
Epoch 299/300
1632/1632 [==============================] - 1s 475us/step - loss: 0.0934
Epoch 300/300
1632/1632 [==============================] - 1s 428us/step - loss: 0.0935
```

[그림 58] Model Fitting

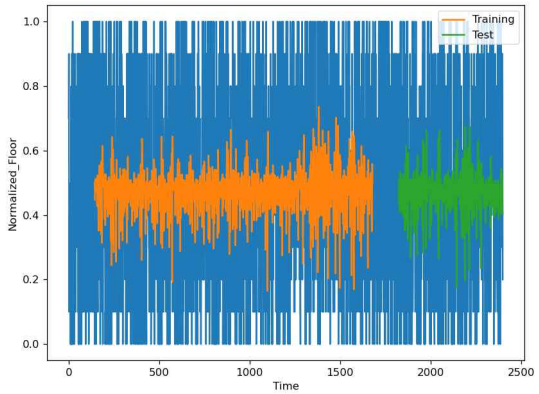
□ Model predict

Model.predict를 활용하여 Reshape한 X_test에 대한 예측 값을 구할 수 있다.

```
[[0.15971999]
 [0.19364105]
 [0.23279335]
 [0.3018855 ]
 [0.55090237]
 [0.69392836]
 [0.59898657]
```

[그림 59] Model Predict

□ Plot함수를 사용하여 시각화



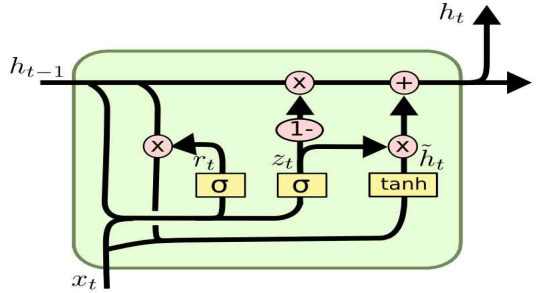
[그림 60] LSTM 시각화

다. GRU

1) 개념

□ GRU(Gated Recurrent Unit)이란 RNN 프레임워크의 일종으로, LSTM의 구조를 간단하게 개선한 모델이다. LSTM 모델에서는 장기의존성 문제 해결을 위한 매개변수가 많아 학습 시간이 오래 걸리는 단점이 있다. GRU는 기존 LSTM의 매개변수를 줄임으로써 성능은 유지하되, 학습 시간을 단축시킨다.

□ LSTM은 Forget Gate, Input Gate, Output Gate 총 3개의 게이트를 통해 Cell State와 Hidden State 2개의 정보를 출력했으나, GRU는 LSTM의 Forget Gate와 Input Gate를 통합한 Reset Gate와 Update Gate를 통해 Hidden State 1개의 정보만 출력함으로써, 학습 간 가중치가 적기 때문에 시간 효율성이 높다.



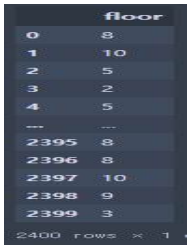
[그림 61] GRU 구조[3)]

3) 출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

2) 분석 과정

□ Data 불러오기

전처리된 학습데이터 2400개를 불러온다.



	floor
0	8
1	10
2	5
3	2
4	5
...	...
2395	8
2396	8
2397	10
2398	9
2399	3

2400 rows × 1

[그림 62] 전처리 데이터

□ Scaling

MinMax Scaling을 통해 학습 데이터를 0~1까지 범위로 Scaling한다. Scaling을 통해 예측 정확도를 높일 수 있다.

```
array([0.7, 0.9, 0.4, ..., 0.9, 0.8, 0.2])
```

[그림 63] Scaling

□ Training Set & Test Set Split

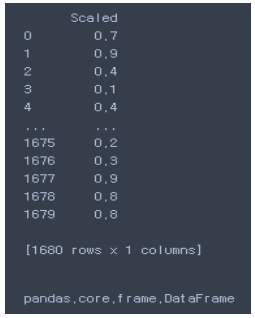
트레이닝 데이터와 테스트 데이터의 비율을 각각 70%, 30%로 설정한다.

```
Total Size: 2400
Train Size: 1680
Test Size: 720
```

[그림 64] 데이터 분리

□ Training Data & Test Data Dataframe 변환

정규화가 완료된 Training Data 및 Test Data를 Array에서 Pandas Dataframe으로 변환한다. 타입을 변경하는 이유는 시계열 데이터 처리에 대한 다양한 기능을 제공하며 기계학습을 위한 window를 만들 때 유용하기 때문이다.



	Scaled
0	0.7
1	0.9
2	0.4
3	0.1
4	0.4
...	...
1675	0.2
1676	0.3
1677	0.9
1678	0.8
1679	0.8

[1680 rows x 1 columns]

pandas.core.frame.DataFrame

[그림 65] Dataframe 변환

□ Window 생성

Pandas shift를 통해 window를 만든다. window는 GRU를 훈련하기 위한 단위로 고정된 사이즈를 갖는데 window size가 12라면 과거의 시간 데이터 12개를 사용하여 다음 시간 단위의 값을 예측하게 된다. Dataframe의 shift연산을 사용하면 인덱스는 그대로 두고 데이터만 시간 단위로 이동할 수 있어 데이터 셋을 구성하기에 용이하다.

	Scaled	shift_1	shift_2	shift_3	shift_4	shift_5	shift_6	shift_7	shift_8	shift_9	shift_10	shift_11	shift_12
1670	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0	0.0	0.4	0.6	0.8	0.9
1671	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0	0.0	0.4	0.6	0.8
1672	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0	0.0	0.4	0.6
1673	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0	0.0	0.4
1674	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0	0.0
1675	0.2	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4	1.0
1676	0.3	0.2	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6	0.4
1677	0.9	0.3	0.2	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6	0.6
1678	0.8	0.9	0.3	0.2	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2	0.6
1679	0.8	0.8	0.9	0.3	0.2	0.4	0.3	0.7	0.2	0.2	0.2	0.8	0.2

[그림 66] window 생성

□ X_training, X_test, Y_training, Y_test 만들기

X_training와 X_test는 각각 1개의 데이터가 window 크기를 가지는 트레이닝 셋, 테스트 셋을 의미하며, Y_training, Y_test는 실제 트레이닝 셋 범위의 실제 값, 테스트 셋 범위의 실제 값을 나타낸다.

```
X_train
[[0.5 0.  0.3 ... 0.4 0.9 0.7]
 [0.  0.5 0.  ... 0.1 0.4 0.9]
 [0.6 0.  0.5 ... 0.4 0.1 0.4]
 ...
 [0.3 0.2 0.4 ... 0.2 0.6 0.6]
 [0.9 0.3 0.2 ... 0.8 0.2 0.6]
 [0.8 0.9 0.3 ... 0.2 0.8 0.2]]

X_test
[[0.  0.6 0.3 ... 1.  0.  0.8]
 [0.4 0.  0.6 ... 0.7 1.  0. ]
 [0.1 0.4 0.  ... 0.4 0.7 1. ]
 ...
 [0.7 0.7 0.9 ... 0.2 0.  0.3]
 [0.9 0.7 0.7 ... 0.9 0.2 0. ]
 [0.8 0.9 0.7 ... 1.  0.9 0.2]]
```

[그림 67] Train set & Test set 만들기

□ Training set & Test set Reshape

최종 훈련 데이터를 생성하기 위해서 Keras의 GRU에서 필요로 하는 3차원 데이터로 변환을 시켜야한다. 따라서 이 형태로 데이터를 reshape한다.

```
X_train Reshape
(1668, 12, 1)

X_train
[[[0.5]
  [0. ]
  [0.3]
  ...
  [0.4]
  [0.9]
  [0.7]]

 [[0. ]
  [0.5]
  [0. ]
  ...
  [0.1]
  [0.4]
  [0.9]]

 [[0.6]
  [0. ]
  [0.5]
  ...
  [0.4]
  [0.1]
  [0.4]]

 ...

X_test Reshape
(708, 12, 1)

X_test
[[[0. ]
  [0.6]
  [0.3]
  ...
  [1. ]
  [0. ]
  [0.8]]

 [[0.4]
  [0. ]
  [0.6]
  ...
  [0.7]
  [1. ]
  [0. ]]

 [[0.1]
  [0.4]
  [0. ]
  ...
  [0.4]
  [0.7]
  [1. ]]

 ...
```

[그림 68] Training set & Test set Reshape

□ GRU 모델 생성

Keras의 Sequential을 사용하여 GRU 모델을 생성한다.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 20)                1380
_________________________________________________________________
dense (Dense)                (None, 1)                 21
=================================================================
Total params: 1,401
Trainable params: 1,401
Non-trainable params: 0
_________________________________________________________________
```

[그림 69] GRU 모델링

□ Model Fitting

Fit 메소드를 통해 훈련한다. 모델을 훈련한다는 것은 훈련용 데이터 셋을 이용하여 최적화 과정을 통해 모델의 weight를 찾는 것이다. 이때 유사 패턴을 학습한다.

```
120/120 [==============================] - 0s 4ms/step - loss: 0.0945
Epoch 294/300
120/120 [==============================] - 0s 4ms/step - loss: 0.0946
Epoch 295/300
120/120 [==============================] - 0s 4ms/step - loss: 0.0946
Epoch 296/300
120/120 [==============================] - 0s 4ms/step - loss: 0.0943
Epoch 297/300
120/120 [==============================] - 0s 4ms/step - loss: 0.0942
Epoch 298/300
120/120 [==============================] - 0s 4ms/step - loss: 0.0944
Epoch 299/300
120/120 [==============================] - 1s 5ms/step - loss: 0.0942
Epoch 300/300
120/120 [==============================] - 1s 5ms/step - loss: 0.0942


<tensorflow.python.keras.callbacks.History at 0x20569c1b190>
```

[그림 70] Model Fitting

□ Model predict

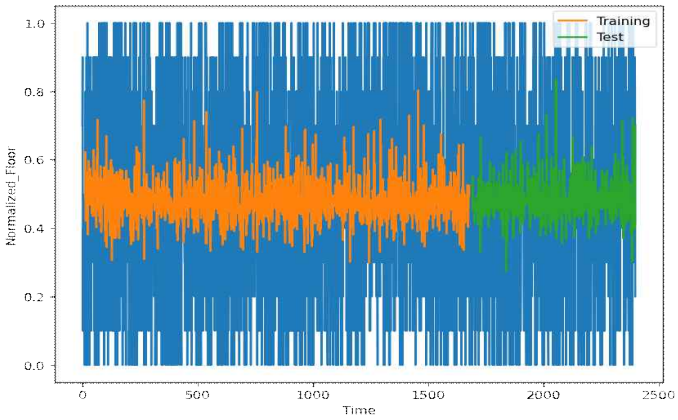
Model.predict를 활용하여 Reshape한 X_test에 대한 예측 값을 구할 수 있다.

```
 - 예측값 -
[[0.5363572 ]
 [0.45247823]
 [0.55171645]
 [0.46643317]
 [0.4712006 ]
 [0.48684293]
 [0.4802572 ]
 [0.45400125]
 [0.45617473]
 [0.44679707]
```

[그림 71] 예측 결과

□ Plot함수를 사용한 시각화

실제 데이터는 파란색 그래프이고, 트레이닝 셋 데이터로 학습 중 결과를 주황색 그래프로, 트레이닝 이후 검증 결과를 초록색 그래프로 표현하였다. 그래프 시각화를 통해 학습이 얼마나 잘되고 예측 정확도가 얼마나 높은지 확인할 수 있다.



[그림 72] 전체 범위에 대한 데이터 시각화

4. 분석 결과

가. ARIMA

□ 대기시간 감소의 극대화를 위한 최적의 모델을 찾기 위해 모수 p, d, q를 조정하여 분석하였다.

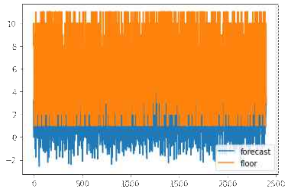
□ 본 프로젝트에서 모수 d의 경우, ACF와 PACF를 통해 차분이 필요하지 않음을 확인할 수 있었고, 실험 결과 대기시간 감소를 알 수 없어 결과에서 제외되었다.

□ p가 0일 때, q가 증가함에 따라 대기시간이 감소하다가 일부 시점이 지나면서 다시 증가하는 추세를 확인할 수 있었고, p가 증가함에 따라 오히려 대기시간이 증가하면서 모델 성능이 떨어짐을 보였다.

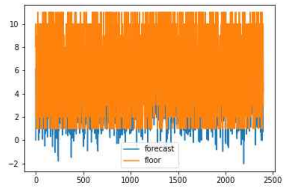
□ p를 증가시킬 경우 표에 나타난 X표시 이후의 q값에 대해서는 모델이 생성되지 않음을 확인하였다.

q \ p	0	1	2	3	4
0		8:23:39	8:23:36	8:25:24	8:23:37
1	8:04:52	8:40:47	8:39:45	8:40:47	8:40:47
2	8:28:54	8:39:45	8:40:47	X	8:40:47
3	8:00:28	8:39:26	8:40:28	X	8:37:45
4	8:39:19	8:39:26	8:40:47	X	X
5	8:18:41	8:41:00	8:40:28	X	X
6	8:13:02	X	X	X	X
7	7:51:09	X	X	X	X
8	7:46:06	X	X	X	X
9	7:54:29	X	X	X	X
10	7:56:42	X	X	X	X
11	7:55:39	X	X	X	X
12	8:03:38	X	X	X	X
13	8:01:07	X	X	X	X
14	7:57:03	X	X	X	X
15	7:54:37	X	X	X	X
16	8:01:51	X	X	X	X
17	8:05:10	X	X	X	X
18	8:09:52	X	X	X	X

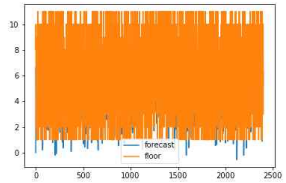
[그림 73] ARIMA 분석 결과



[그림 74] ARIMA(0,0,1)



[그림 75] ARIMA(0,0,4)



[그림 76] ARIMA(0,0,8)

나. LSTM

1) 파라미터별 결과 비교

☐ 최적의 대기시간 절감 모델을 찾기 위해 LSTM 모델 생성 간 Batch Size, Window Size, Epoch의 파라미터를 조정하며 비교 분석을 진행하였다.

☐ LSTM 모델 성능 비교 간 Batch=14, Window=168, Epoch=10 일 때 대기시간이 가장 작았다.

Batch size = 7

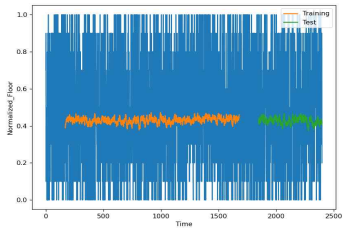
w \ epoch	10	30	50	100	150
12	8:02:48	8:01:46	8:03:45	8:05:05	8:07:06
24	8:05:09	8:02:48	8:02:48	8:01:04	8:28:24
48	7:52:45	8:02:39	8:02:48	8:02:21	8:03:10
72	7:56:43	8:02:48	8:02:21	8:00:56	8:11:22
96	8:01:19	8:00:00	8:00:29	8:01:48	8:01:00
120	8:02:48	8:02:48	7:59:51	8:00:38	8:03:26
144	8:02:48	8:02:04	8:01:40	8:00:51	8:10:03
168	7:53:08	8:02:32	7:55:08	8:02:34	7:59:02
192	7:57:24	8:02:48	8:02:48	7:57:47	8:00:25
288	8:01:13	8:02:48	8:02:48	8:05:32	8:02:48
384	8:02:03	7:50:29	8:02:48	8:02:47	8:02:48

[그림 77] LSTM 대기시간 결과표 (Batch=7)

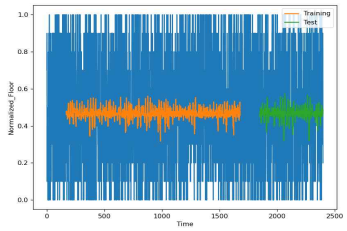
Batch size = 14

w \ epoch	10	30	50	100	150
12	8:02:48	8:02:48	8:02:48	8:02:48	8:04:29
24	8:05:45	8:02:48	8:00:51	8:05:35	8:08:07
48	8:04:13	7:59:11	8:02:40	8:02:48	8:03:12
72	7:50:25	7:58:57	8:02:48	8:00:35	8:01:39
96	8:01:49	8:03:09	8:01:26	8:06:30	8:02:19
120	7:56:06	8:02:48	8:02:48	8:02:16	8:02:48
144	8:01:37	8:02:48	8:02:48	8:02:35	8:06:17
168	7:47:51	7:57:14	8:02:48	8:01:58	7:58:58
192	7:56:23	7:56:17	8:02:06	8:02:48	8:03:00
288	8:02:48	8:02:48	8:02:48	8:02:35	8:04:00
384	8:02:48	8:02:48	8:02:43	8:02:35	8:03:30

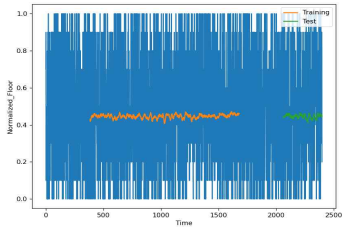
[그림 78] LSTM 대기시간 결과표 (Batch=14)



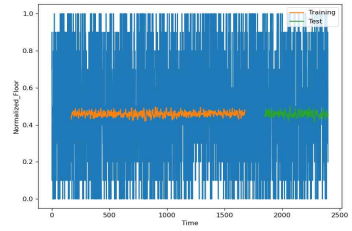
[그림 79] LSTM (B=14/W=168/E=10)



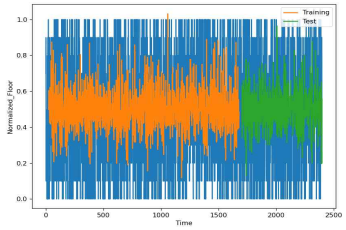
[그림 80] LSTM (B=14/W=168/E=50)



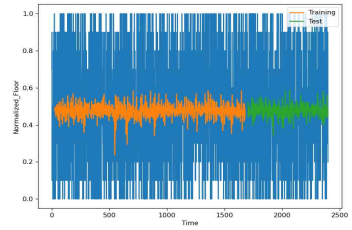
[그림 81] LSTM (B=7/W=384/E=30)



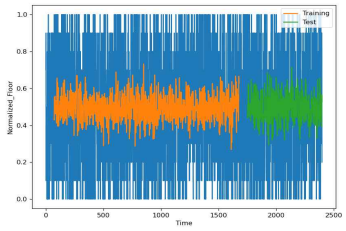
[그림 82] LSTM (B=7/W=168/E=50)



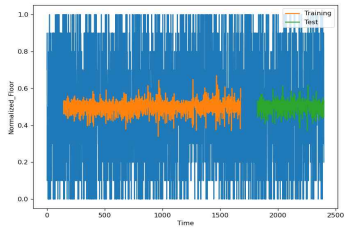
[그림 83] LSTM (B=7/W=24/E=150)



[그림 84] LSTM (B=14/W=24/E=150)



[그림 85] LSTM (B=7/W=72/E=150)



[그림 86] LSTM (B=14/W=144/E=150)

다. GRU

1) 파라미터별 결과 비교

☐ 최적의 대기시간 절감 모델을 찾기 위해 GRU 모델 생성 간 Batch Size, Window Size, Epoch의 파라미터를 조정하며 비교 분석을 진행하였다.

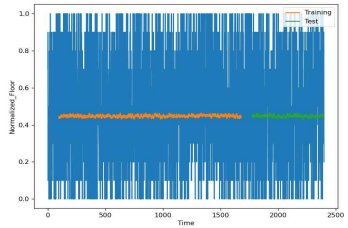
☐ GRU 성능 비교 간 Batch=14, Window=168, Epoch=10 일 때 대기시간이 가장 작았다.

GRU batch = 14								
w \ epoch	10	30	50	100	150	200	250	300
12	8:02:48	8:02:48	8:02:48	8:02:48	8:04:53	8:06:07	8:10:13	8:00:43
24	8:02:48	8:02:48	8:02:48	8:04:08	8:14:51	8:12:19	8:24:37	8:26:22
48	8:06:51	8:00:10	8:02:00	8:01:14	8:04:07	8:02:22	7:56:32	8:20:30
72	8:02:48	7:57:42	8:02:48	8:02:42	8:09:25	8:07:45	8:04:23	8:09:31
96	8:02:48	8:02:48	8:02:48	8:02:48	8:01:15	8:12:56	8:05:17	8:17:08
120	7:57:53	8:02:48	8:02:48	8:02:50	8:00:52	7:57:40	8:01:22	8:05:10
144	8:01:22	8:01:14	8:02:07	8:02:34	8:05:44	8:15:12	8:03:13	8:11:22
168	8:02:48	8:02:48	8:02:48	8:03:36	8:03:33	8:02:44	8:04:46	8:05:05
192	8:01:06	8:00:10	7:59:32	8:02:02	8:10:04	7:59:45	8:04:24	8:17:26
288	8:02:48	8:19:36	8:02:48	8:02:48	8:03:24	8:04:46	8:04:46	8:04:50
384	8:02:48	7:57:30	8:01:15	8:06:10	8:07:17	8:05:01	8:10:57	8:15:31

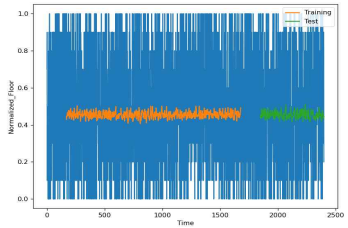
[그림 87] GRU 대기시간 결과표 (Batch=14)

GRU batch = 7							
w \ epoch	10	30	50	100	150	200	250
24	8:03:41	8:02:48	8:04:58	8:03:35	8:15:03	8:15:25	8:26:27
48	7:58:15	7:56:36	8:03:43	8:02:00	8:02:39	8:07:29	8:08:34
72	8:00:37	8:02:48	8:04:53	8:03:03	8:08:17	8:04:57	8:03:21
96	7:52:15	8:02:00	8:02:48	8:01:23	7:58:46	8:05:03	8:09:10
120	8:02:48	8:02:48	8:02:48	8:02:25	8:02:48	7:58:22	8:02:01
144	8:01:45	8:03:08	8:02:16	7:59:08	8:01:59	8:07:39	8:14:08
168	8:06:52	8:02:48	8:02:48	7:54:30	7:58:59	7:59:27	8:09:59
192	8:01:51	8:02:11	8:02:48	7:59:52	7:59:49	8:03:31	8:15:19
288	8:02:48	8:01:14	8:02:48	8:03:46	8:02:42	8:16:56	8:06:32
384	8:02:35	8:01:10	8:01:18	8:00:39	8:05:01	8:11:47	8:01:14

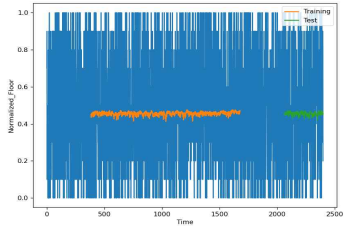
[그림 88] GRU 대기시간 결과표 (Batch=7)



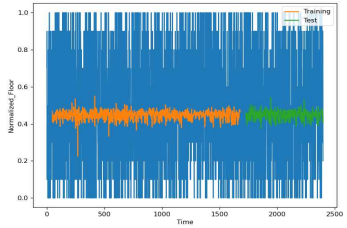
[그림 89] GRU(B=7/W=96/E=10)



[그림 90] GRU(B=7/W=168/E=100)



[그림 91] GRU(B=14/W=384/E=30)

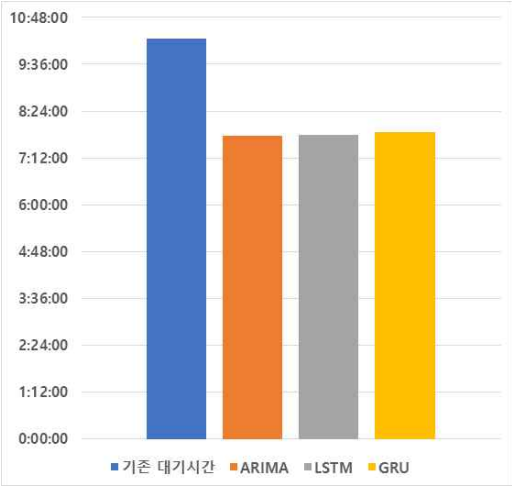


[그림 92] GRU(B=14/W=48/E=250)

라. 종합결과

1) 세 가지 모델 분석 결과 **평균 23.83%의 대기시간 감소**가 나타났다. 시계열 분석을 통해, 시간별 최적의 층에 승강기를 배치시킴으로써 **대기시간을 최소화할 수 있음**을 확인할 수 있다.

2) 최종적으로 **ARIMA, LSTM, GRU 모델의** 최소 대기시간을 비교하였다. 3가지 모델 모두 기존 대비 각각 24.26%, 23.97%, 23.26%만큼 대기시간이 감소했으며, **ARIMA** 모델의 성능이 제일 좋았지만, **LSTM**과 **GRU** 모델의 경우 추가적인 파라미터 조정을 통해 더 낮은 대기시간 산출이 가능할 것으로 판단된다.



[그림 93] 대기시간 비교

[표 4] 모델별 대기시간 비교

분석기간 전체 대기시간			
변경 전	ARIMA	LSTM	GRU
10:15:23	7:46:06	7:47:51	7:52:15
기존 시간 대비	-2:29:17 (24.26%)	-2:27:32 (23.97%)	-2:23:08 (23.26%)

5. 활용 방안

가. 한계점 및 개선 방향

1) 부족한 데이터의 한계

□ 현재 승강기 운행 데이터는 아예 저장되지 않거나, 길어도 한 달 주기로 삭제되고 있는 상황이고, 현재 본 프로젝트에서 사용된 데이터는 8일치로 데이터 분석을 하기에 턱없이 부족한 상황 (전체 데이터가 60만 건에 불과)

□ 빅데이터 분석을 통해 버려지는 승강기 운행데이터 재가공 및 활용 방안 모색을 위해서는 **장기간의 운행데이터가 필요할 것으로 판단**

2) Call과 No Call 데이터의 불균형으로 인한 Call 부족

□ 전체 운행 데이터 중에서 승강기 Call 발생 빈도는 1%도 채 되지 않음 (정제된 데이터 54만여 건 중에 1100여 건)

□ 승강기 Call 1100여 건 중에서도 승강기 내부 이벤트인 Car Call을 제외하면 400여 건 정도의 Hall Call 만이 실제 데이터로 사용됨 (일일 평균 50여 건의 Hall Call)

□ 월간 데이터로 가정하더라도 Hall Call은 1500여 건 남짓으로 예상되므로 **최소 연간 단위의 운행데이터를 기반으로 한 빅데이터 분석이 필요**

3) Raw data error로 인한 데이터 정제 시간 증가 및 어려움

□ Call 데이터 수신 지연으로 인해 이미 승강기 상태정보에서는 움직이고 있는 상태의 데이터가 수신되고 있는데 이때 Call 요청이 들어오는 문제가 발생함

□ 초 단위 데이터가 2~3초씩 건너뛰어 입력되는 문제로 인해 분석에 어려움이 존재

□ 오버 샘플링 기법 적용으로 예측 모델이 학습데이터에 과적합 되거나 예측 모델 성능을 저하시키는 문제가 발생하고 패턴의 유사성이 존재하기 때문에 **시계열 분석에 국한되지 않고 패턴 분석 기법을 활용해 분석하고 예측해 볼 수 있음**.

나. 기대 효과 및 활용 방안

1) 기술적 측면

□ 폐기되는 데이터를 클라우드 기반 빅데이터 수집 및 저장하는 것과 분석을 통해 기존에 없는 인사이트 추출 및 비즈니스와 서비스 플랫폼으로 활용할 수 있다. 시계열 분석 및 상관분석을 통해 데이터 기반의 현재 특징추출 및 모형 생성이 가능하다. 더 나아가 기계학습 및 인공지능을 활용하여 지속적 운행데이터를 통한 학습 모델 갱신 및 진화를 할 수 있다. 이를 활용하는 것은 **승강기의 운행데이터를 클라우드 기반으로 저장하여 빅데이터 분석 및 예측을 통해 승강기 용도에 맞는 모델을 생성**하는 것이다.

□ 승강기의 운행 패턴과 승강기의 부품들의 정보를 분석하여 기존의 A/S 이력을 비교하여 고장이 나지 않아도 사전 예방하는 차원으로 방향을 설정할 수 있다. 승강기의 고장이 발생한 후 수리를 하러 가는 것은 사용자의 불편함을 초래하게 된다. 승강기의 이용이 잦아지면 당연히 부품 소모가 빠르게 진행되고 사용할 수 있는 기간까지 줄어들게 된다. 그래서 이를 예측하여 불편함을 사전 방지하고 사고 예방까지 할 수 있다.

2) 경제적 측면

□ 신규 시장 창출 및 글로벌 비즈니스 확대로 다양한 수준의 서비스 차별화를 통한 시장의 확대 및 글로벌 승강기 서비스 시장진입을 꾀할 수 있다.

□ 유명 해외 승강기업체들 역시 IoT 기반 승강기를 공급하고 있지만, 개발도상국에서는 비용부담으로 수요가 적을 것으로 예상되어, 개발도상국 시장은 블루오션으로 판단된다. 클라우드 기반으로 국가의 상황에 맞는 승강기 운행데이터를 활용하여 **효율화 운행 알고리즘을 개발 및 제공하여 개도국 시장으로 수출 할 수 있다.**

3) 산업적 측면

□ ICT 기술을 통합 제조업과 서비스업의 융합 활성화로 빅데이터, 인공지능, 클라우드 기술들이 전통적인 제조업과 융합하여 제조 서비스화를 통한 신산업 동력 확보를 할 수 있다.

□ 현재 대부분의 대기업들만 IoT 기반의 승강기 기술을 활용하고 있으며, 그에 반해 중소 승강기업체들은 비용적인 부담으로 활용하기 어려운 현실에 있다. 하지만 클라우드 기반의 서비스 플랫폼을 공급하면 비용부담으로 IoT 기반 승강기를 사용하지 못하는 중소기업들도 IoT 기반 승강기를 상용화할 수 있다.

4) 사회적 측면

□ 미활용 데이터의 활용과 SW 융합을 통한 4차 산업혁명의 생활화로 우리 주변에 흔히 이용하고 보는 승강기 내의 잠자던 데이터를 활용할 수 있을 뿐만 아니라 ICT 융복합 체험 및 생활화를 할 수 있다.

□ 모델만을 판매하는 것이 아니라 기술을 판매하여 중소기업들의 매출 증대에도 큰 도움이 될 수 있고 국가적인 위상도 올라가는 등 종합적인 측면에서 이익이 될 수 있다.