딥러닝특론

8 주차 과제

인공지능응용

K2025029 금동환

목차

1. 개요

이 프로젝트는 산업용 센서 시계열 데이터(SWaT)를 이용해 ARIMA 기반 이상 탐지 모델을 구축하는 파이프라인으로 구성하는 것을 목표로 함.

학습된 ARIMA 모델로 새로운 시계열에 대한 추론을 하고, 잔차를 바탕으로 이상치를 예측한다.

A. 구성 요소

i. 공통 유틸리티: common_utils.py

데이터 로드, 데이터 전처리, ARIMA 모델 정의, 학습 및 예측 디바이스 선택

ii. 학습 스크립트: train_arima.py

정상 구간 데이터를 이용해 학습, 학습된 파라미터를 저장

iii. 예측 및 시각화: predict_arima.py

이상치가 포함된 데이터에 대해 잔차를 계산, 이상치 판단, 이상치 시각화

B. ARIMA

AR+MA 의 조합으로 잔차 기반 이상 탐지에 최적화된 모델

C. SWaT 데이터셋

산업 제어 시스템용 공개 벤치마크 데이터, 수많은 논문에서 사용된 검증된 데이터셋 모델

2. 코드

A. common_utils.py

```
"""
common_utils.py

■ 역할
  • load_csv(): CSV 에서 지정 태그 시계열 로드, 전처리
  • ARIMAModel: ARIMA(p,d,q) 모델 정의
  • get_device(): CUDA 사용 가능 여부 확인, GPU 목록 출력
"""

import unicodedata
import pandas as pd
import torch
import torch.nn as nn
from typing import Tuple

# 인덱스의 타임스탬프 포맷
DATE_FMT = "%d/%m/%Y %I:%M:%S %p"


def load_csv(csv_path: str, tag: str) -> Tuple[torch.Tensor, pd.DatetimeIndex]:
    """
    CSV 에서 지정 태그 열을 읽어
    1 초 간격으로 리샘플링·보간한 시계열 Tensor 와
    DatetimeIndex 를 반환.
    """
    # 구분자 자동 감지
    with open(csv_path, 'r', encoding='utf-8') as f:
        hdr = f.readline()
    sep = ';' if hdr.count(';') > hdr.count(',') else ','

    # 모든 열을 문자열로 읽어 Timestamp 와 tag 칼럼 추출
    df = pd.read_csv(csv_path, sep=sep, dtype=str, encoding='utf-8')
    cols = df.columns.tolist()
    ts_col = next(c for c in cols if "timestamp" in unicodedata.normalize("NFKC",
c).lower())
    if tag not in cols:
        raise ValueError(f"'{tag}' 열을 찾을 수 없습니다. 헤더: {cols}")
    df = df[[ts_col, tag]]

    # Timestamp 전처리, 공백 제거 > datetime 변환 > 인덱스로 설정
    df[ts_col] = df[ts_col].str.strip()
    df[ts_col] = pd.to_datetime(df[ts_col], format=DATE_FMT, dayfirst=True)
    df.set_index(ts_col, inplace=True)

    # 중복 타임스탬프 제거 (중복된 타임스탬프는 첫번째 값만 유지)
    df = df[~df.index.duplicated(keep='first')]

    # sensor 값 문자열 > float32
    df[tag] = df[tag].str.replace(",", ".", regex=False).astype("float32")

    # 결측치 처리, 1 초 해상도 재샘플링 + 선형 보간
```

```
    series = df[tag].asfreq("1s").interpolate("linear")

    # 텐서 변환 및 인덱스 반환
    tensor = torch.tensor(series.values, dtype=torch.float32)
    return tensor, series.index


class ARIMAModel(nn.Module):
    """
    PyTorch 기반 ARIMA(p, d, q) 모델
    ARIMA(p,d,q) 파라미터 정의:
    - p (AR 차수): 과거 관측치 몇 개를 볼지, 직전 2 시점의 값을 반영
    - d (차분 차수): 몇 번 차분해 정상 시계열로 만들지, 1 차 차분 만으로 안정적 패턴
    - q (MA 차수): 과거 오차 몇 개를 볼지, 직전 2 시점의 오차를 반영
    """

    def __init__(self, p: int = 2, d: int = 1, q: int = 2):
        super().__init__()
        self.p, self.d, self.q = p, d, q
        # 학습 가능한 파라미터
        self.phi = nn.Parameter(torch.zeros(p))
        self.theta = nn.Parameter(torch.zeros(q))
        self.mu = nn.Parameter(torch.zeros(1))

    def forward(self, y: torch.Tensor) -> torch.Tensor:
        # y 가 1D 인 경우 배치 차원 추가
        if y.dim() == 1:
            y = y.unsqueeze(0)

        # 배치 크기(B)와 시퀀스 길이(Tseq)를 가져옴
        B, Tseq = y.shape

        # 모델의 차수 파라미터
        d, p, q = self.d, self.p, self.q

        # 실제 예측 가능한 시퀀스 길이
        T = Tseq - d

        # 잔차를 저장할 버퍼
        eps = y.new_zeros(B, T + q)

        # 차분된 시계열
        yd = y[:, d:] - y[:, :-d]

        # t 에 따라 AR+MA 계산, 잔차 저장
        for t in range(p, T):
            # ar, 과거 p 개 시점의 관측치에 대한 가중합
            ar = (self.phi * torch.flip(y[:, d + t - p:d + t], dims=[1])).sum(dim=1)

            # ma, 과거 q 개 시점의 잔차에 대한 가중합
            ma = (self.theta * torch.flip(eps[:, t - q:t], dims=[1])).sum(dim=1)

            # 예측값 계산
            pred = self.mu + ar + ma

```

```
106.             # 잔차 계산
107.             eps[:, t] = yd[:, t] - pred
108.         # 초기 p 스텝 제외 후 반환
109.         return eps[:, p:]
110.
111.
112. def get_device() -> tuple[torch.device, int]:
113.     """
114.     CUDA 사용 가능 시 ('cuda', GPU 개수), 아니면 ('cpu', 0) 반환
115.     """
116.     print("CUDA available:", torch.cuda.is_available())
117.     if torch.cuda.is_available():
118.         count = torch.cuda.device_count()
119.         [print(f"[{idx}] {torch.cuda.get_device_name(idx)}") for idx in range(count)]
120.         return torch.device("cuda"), torch.cuda.device_count()
121.     else:
122.         print("GPU 미감지: CPU로 실행")
123.         return torch.device("cpu"), 0
124.
```

B. train_arima.py

```python
"""
train_arima.py

■ 역할
  • 정상 데이터 CSV를 읽어 시계열 Tensor 생성
  • ARIMAModel(p=2, d=1, q=2) 객체 생성 후 GPU/CPU로 이동
  • DataParallel로 다중 GPU 사용 지원
  • MSE 손실을 최소화하도록 파라미터 학습
  • 학습 과정 시간 로깅(실제시각, 에폭별 소요)
  • 최종 파라미터를 models/arima_<TAG>.pt에 저장

■ 사용법
    python train_arima.py \
      --csv train.csv \
      --tag LIT101 \
      --epochs 100 \
      --lr 1e-2
"""

import argparse
import time
from datetime import datetime
from pathlib import Path

import torch
from torch.nn.parallel import DataParallel
from tqdm import trange

import sys

from common_utils import load_csv, ARIMAModel, get_device


def train_loop(model, data, epochs, optimizer):
    """
    • model    :  ARIMA 모델
    • data     : 입력 텐서
    • epochs   : 학습 반복 횟수
    • optimizer: 옵티마이저
    """
    # 로깅을 위한 타이머, 시간 객체
    start_wall = datetime.now()
    start_perf = time.perf_counter()
    print(f"학습 시작: {start_wall:%Y-%m-%d %H:%M:%S}")
    prev_perf = start_perf

    for epoch in trange(1, epochs + 1, desc="Epochs"):
        # 기울기 초기화
        optimizer.zero_grad()
        # forward() 호출 + 손실 계산  # 평균제곱오차
        loss = torch.mean((model(data)) ** 2)
        # 역전파
        loss.backward()
```

```
        # 파라미터 업데이트
        optimizer.step()

        # 로깅을 위한 시간 계산
        now_perf = time.perf_counter()
        delta_ms = (now_perf - prev_perf) * 1000
        prev_perf = now_perf

        # 상태 로깅
        print(f"[{epoch:03d}/{epochs}] loss={loss.item():.6f} | Δt={delta_ms:.1f} ms")

    total_time = time.perf_counter() - start_perf
    print(f"학습 종료: {datetime.now():%Y-%m-%d %H:%M:%S} (총 {total_time:.2f}s)")
    return total_time


def main():
    # 1) 인자 파싱
    parser = argparse.ArgumentParser()
    parser.add_argument("--csv", default="train.csv", help="정상 구간 CSV 경로")
    parser.add_argument("--tag", default="LIT101", help="센서 태그명")
    parser.add_argument("--epochs", type=int, default=100, help="전체 Epoch 수")
    parser.add_argument("--lr", type=float, default=1e-2, help="학습률")
    args = parser.parse_args()

    # 2) 디바이스 설정
    device, n_gpu = get_device()
    print(f"학습 디바이스: {device}  (GPU 개수={n_gpu})")

    # 3) 데이터 로드
    try:
        series, _ = load_csv(args.csv, args.tag)
    except Exception as e:
        print(f"데이터 로드 실패: {e}")
        sys.exit(1)
    else:
        # 성공 시에만
        y = series.unsqueeze(0).to(device)  # (1, T)
        if n_gpu > 1:
            y = y.repeat(n_gpu, 1)  # (B=n_gpu, T)

    # 4) 모델 생성
    base_model = ARIMAModel().to(device)
    model = DataParallel(base_model) if n_gpu > 1 else base_model
    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)

    # 5) 학습 수행
    train_loop(model, y, args.epochs, optimizer)

    # 6) 모델 저장
    Path("models").mkdir(exist_ok=True)
    torch.save(base_model.state_dict(), f"models/arima_{args.tag}.pt")
    print(f"모델 저장 완료: models/arima_{args.tag}.pt")


```

```
109. if __name__ == "__main__":
110.     main()
111.
```

C. predict_arima.py

```
"""
predict_arima.py

■ 역할
  • 학습된 ARIMA 파라미터(.pt)를 로드
  • 평가용 CSV 에서 동일 전처리 후 시계열 Tensor 생성
  • 1 단계앞 잔차 계산
  • 잔차를 Z-score 표준화, Z>3 인 포인트를 이상치로 분류
  • 이상치 개수 및 비율, 타임스탬프 출력
  • Matplotlib 으로 잔차 시계열과 이상치 시각화

■ 사용법
    python predict_arima.py \
      --csv test.csv \
      --tag LIT101 \
      --model models/arima_LIT101.pt
"""
import argparse
import sys

import torch
from scipy.stats import zscore
import matplotlib.pyplot as plt

from common_utils import load_csv, ARIMAModel, get_device


def main():
    # 1) 인자 파싱
    parser = argparse.ArgumentParser()
    parser.add_argument("--csv", default="test.csv", help="평가용 CSV 파일 경로")
    parser.add_argument("--tag", default="LIT101", help="예측할 센서 태그명")
    parser.add_argument("--model", default="models/arima_LIT101.pt", help="학습된
파라미터(.pt) 경로")
    args = parser.parse_args()

    # 2) 디바이스 확인
    device, _ = get_device()
    print(f"추론 디바이스: {device}")

    # 3) 테스트 데이터 로드 및 전처리
    try:
        series, ts_index = load_csv(args.csv, args.tag)
    except Exception as e:
        print(f"데이터 로드 실패: {e}")
        sys.exit(1)
    else:
        # 성공 시에만
        series = series.to(device)

    # 4) 모델 초기화 및 가중치 로드
    model = ARIMAModel().to(device)
```

```
52.
53.     # 저장한 pt 파일을 불러와, 모델에 매핑
54.     state = torch.load(args.model, map_location=device)
55.     model.load_state_dict(state)
56.     model.eval()
57.
58.     # 5) 잔차 계산
59.     with torch.no_grad():
60.         # forward()가 호출됨, 1 단계앞 예측 후 실제값과의 차이가 잔차
61.         residuals = model(series).squeeze(0)
62.
63.     # 잔차의 총 길이, 너무 짧으면 계산 못함.
64.     res_len = residuals.numel()
65.     if res_len == 0:
66.         print("!! 입력 시계열이 너무 짧아 잔차 계산 불가 !!")
67.         sys.exit(1)
68.
69.     # 6) Z-score 로 이상치 판정
70.     # 잔차를 정규분포의 표준정규(z)값 표준화
71.     z_scores = zscore(residuals.cpu().numpy(), nan_policy="omit")
72.
73.     # 3 시그마로 이상치 판정
74.     anoms = abs(z_scores) > 3
75.
76.     # 7) 잔차 길이만큼 타임스탬프 정렬
77.     aligned_ts = ts_index[-res_len:]
78.
79.     # 8) 결과 출력
80.     count = int(anoms.sum())
81.     ratio = count / res_len * 100
82.     print(f"전체 {res_len} 스텝 중 이상치 {count}개 ({ratio:.2f}%)")
83.     print("이상치 타임스탬프 (10 개만):")
84.     for t in aligned_ts[anoms][:10]:
85.         print(" * ", t)
86.
87.     # 9) 시각화
88.     plt.figure(figsize=(12, 4))
89.     plt.plot(aligned_ts, residuals.cpu().numpy(), label="Residuals")
90.     plt.scatter(aligned_ts[anoms], residuals.cpu().numpy()[anoms],
91.                 color="red", label=f"Anomalies ({count})", zorder=5)
92.     plt.title(f"Residuals & Anomalies for {args.tag}")
93.     plt.xlabel("Timestamp")
94.     plt.ylabel("Residual")
95.     plt.legend()
96.     plt.xticks(rotation=30)
97.     plt.tight_layout()
98.     plt.show()
99.
100.
101. if __name__ == "__main__":
102.     main()
103.
```

D. 훈련 데이터

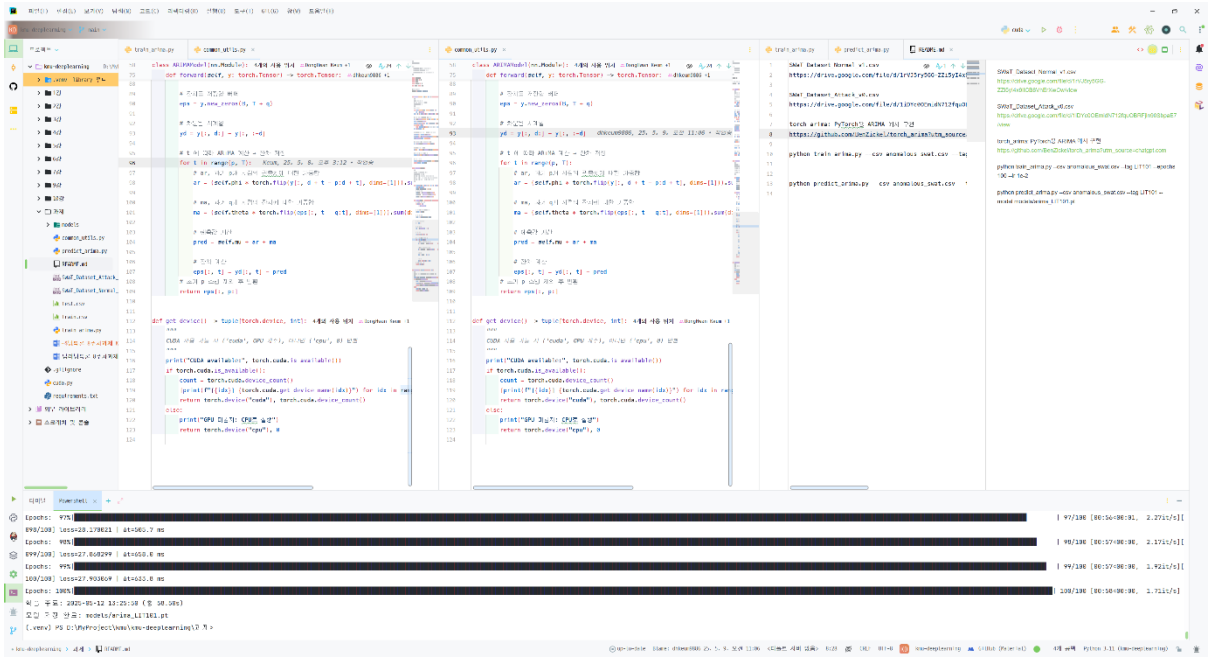
```
Timestamp,FIT101,LIT101,MV101,P101,P102,AIT201,AIT202,AIT203,FIT201,MV201,P201,P202,P203,P204,P2
05,P206,DPIT301,FIT301,LIT301,MV301,MV302,MV303,MV304,P301,P302,AIT401,AIT402,FIT401,LIT401,P401
,P402,P403,P404,UV401,AIT501,AIT502,AIT503,AIT504,FIT501,FIT502,FIT503,FIT504,P501,P502,PIT501,P
IT502,PIT503,FIT601,P601,P602,P603,Normal/Attack
22/12/2015 04:30:00
PM,"102,483571","106,996777","96,624109","90,460962","95,682532","97,881202","94,429593","103,92
5925","99,834874","103,827011","96,607526","94,109102","105,874070","100,058682","98,878795","99
,282884","104,396536","98,300894","101,609176","101,325535","101,741431","112,555663","93,211838
","110,994185","96,643132","100,854368","99,293284","96,081199","108,827963","102,973991","90,09
7140","93,402585","103,660071","97,403275","110,851274","104,570489","102,832979","94,035316","1
01,999294","101,869903","97,640712","97,141042","97,688142","96,386469","103,062803","105,248912
","102,560151","94,719208","103,037855","112,071239","100,491031",Normal
22/12/2015 04:30:01
PM,"99,308678","104,623168","99,277407","95,698075","99,843983","97,732929","96,845346","91,1115
95","97,481749","105,367064","98,472503","102,164252","90,605095","102,637193","98,945826","99,8
36720","100,196546","100,910674","96,093212","97,697383","101,416618","99,900342","99,366054","1
01,716777","96,431003","100,061277","94,087400","105,820172","103,402753","102,767448","94,72507
2","104,579814","91,942172","103,516005","99,095129","108,122436","99,197090","102,052599","105,
296224","100,822990","105,063512","101,343738","104,340210","93,906067","106,959858","96,097336"
,"100,707861","103,818832","96,355764","96,817187","99,679459",Normal
…
```

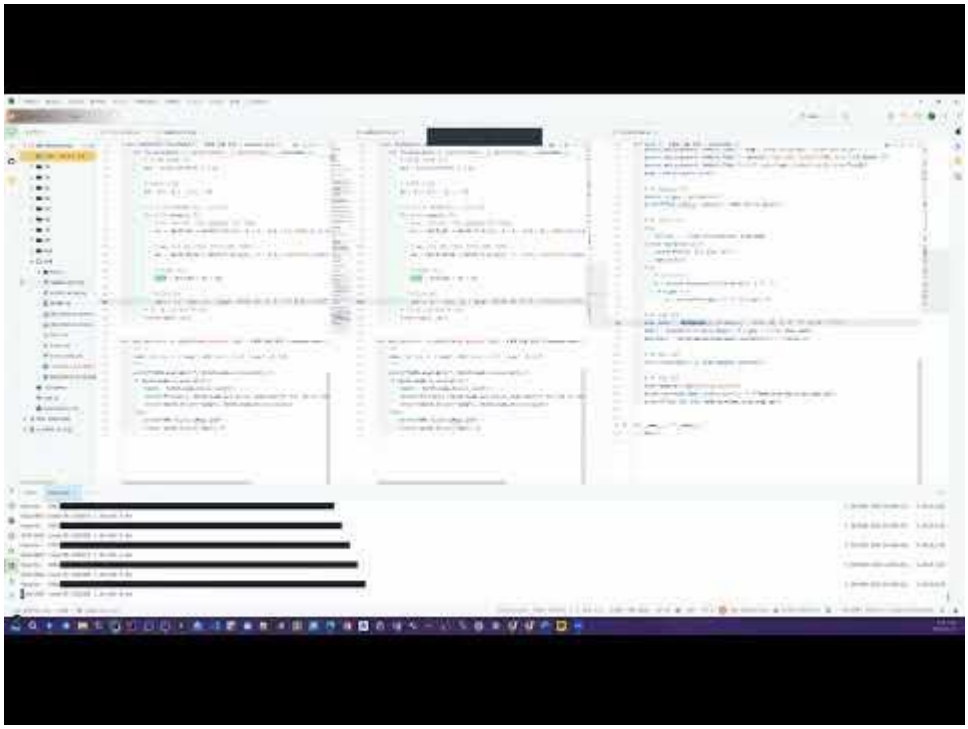
E. 테스트 데이터

```
Timestamp,FIT101,LIT101,MV101,P101,P102,AIT201,AIT202,AIT203,FIT201,MV201,P201,P202,P203,P204,P2
05,P206,DPIT301,FIT301,LIT301,MV301,MV302,MV303,MV304,P301,P302,AIT401,AIT402,FIT401,LIT401,P401
,P402,P403,P404,UV401,AIT501,AIT502,AIT503,AIT504,FIT501,FIT502,FIT503,FIT504,P501,P502,PIT501,P
IT502,PIT503,FIT601,P601,P602,P603,Normal/Attack
22/12/2015 04:30:00
PM,"94,571847","96,255863","91,128882","97,747007","97,643622","101,593818","101,134534","105,58
9245","98,385400","105,657692","93,795152","105,687452","93,180742","99,260779","100,505800","10
4,537821","101,458562","95,640633","110,426860","99,881584","105,157967","103,980952","94,163678
","96,993231","89,753369","106,599264","103,669781","91,523672","99,131889","106,174774","94,729
148","103,592040","102,094690","97,725440","103,593832","94,636496","106,398991","105,260018","1
04,393012","99,503554","109,561249","93,424874","104,020495","101,032893","98,441578","96,220214
","99,289750","97,017091","109,241432","105,859692","109,725588",Normal
22/12/2015 04:30:01
PM,"104,986727","102,837974","93,993113","103,047952","105,420362","107,555519","88,007232","99,
971367","102,505801","98,392449","98,435266","101,685785","97,026186","94,321452","103,113864","
108,426086","101,300476","99,688567","101,556456","98,050747","106,659724","105,929789","107,333
883","95,086666","94,627212","96,389191","97,371165","98,055226","97,572111","95,586200","96,084
943","111,665480","103,641377","102,186517","108,797350","97,497525","102,051817","95,526880","1
06,370112","93,125188","98,658643","114,850873","89,452013","101,878468","105,619463","100,86172
2","104,584032","95,429671","104,015153","91,493462","102,039913",Normal
…
```

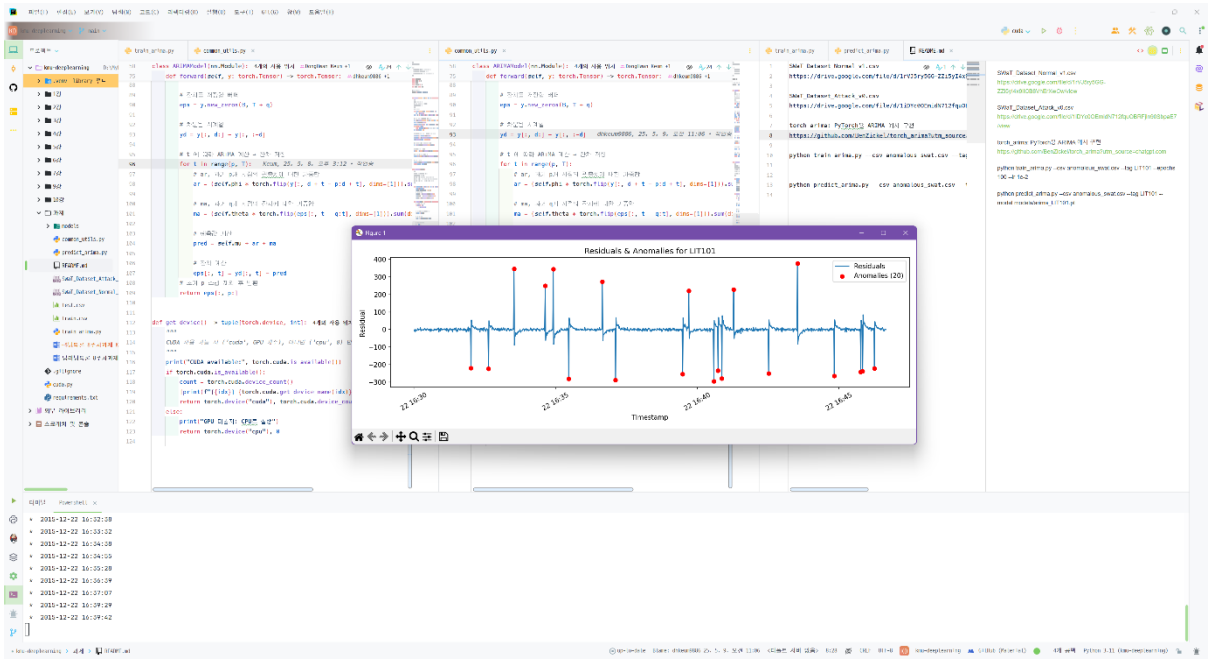
3. 실행 화면

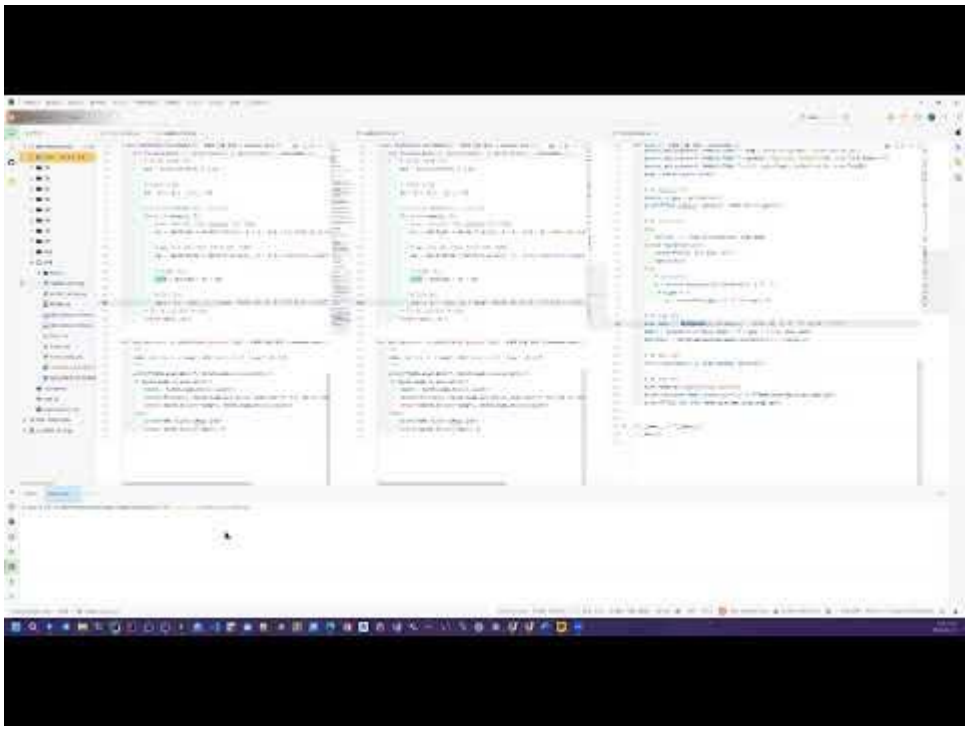
A. 훈련





B. 예측


Residuals & Anomalies for LIT101
Residuals
Anomalies (20)
Residual
Timestamp



4. 참고

A. Pytorch 로 구현한 ARIMA 모델 클래스 참고

https://github.com/BenZickel/torch_arima