



III. AI훈민정음_4 Satellite_Regression (허리케인 위성 이미지의 풍속 예측)

T3Q.ai

Copyright © 2021 T3Q Co., Ltd. All Rights Reserved.

목 차

I. 개요

1. 소개

II. 프로그래밍 가이드 문서

0_local_satellite_regression_requirement

0_local_satellite_regression.ipynb

1_local_platform_satellite_regression.ipynb

2_platform_process

III. 수행 절차

01) T3Q.cep_데이터수집 파이프라인_satellite_regression

02) T3Q.cep_데이터변환 파이프라인_satellite_regression

03) T3Q.dl_프로젝트 설정_실행환경 관리_satellite_regression

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_satellite_regression

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_satellite_regression

06) T3Q.dl_학습플랫폼_데이터셋 관리_satellite_regression

07) T3Q.dl_학습플랫폼_전처리 모델 설계_satellite_regression

08) T3Q.dl_학습플랫폼_전처리 모델 관리_satellite_regression

09) T3Q.dl_학습플랫폼_학습모델 설계_satellite_regression

10) T3Q.dl_학습플랫폼_학습모델 관리_satellite_regression

11) T3Q.dl_추론플랫폼_추론모델 관리_satellite_regression

12) T3Q.dl_추론플랫폼_추론API관리_satellite_regression

13) T3Q.cep_실시간 추론 파이프라인_satellite_regression

1. 소개 : 허리케인 위성 이미지의 풍속 예측

'Wind-dependent Variables: Predict Wind Speeds of Tropical Storms' 대회에서 제공하는 위성 데이터를 이용하여 풍속을 예측하는 예제

1. 데이터셋

위성 이미지 데이터:
폭풍의 단일 대역 위성 이미지와 해당 이미지의 풍속으로 구성
데이터셋은 train, test 폴더와 해당 이미지의 풍속이 기록된 2개의 csv 파일로 구성

이 중 간단한 학습을 위해 [데이터셋](#)을 1/10 크기로 줄임.

2. 전처리 및 학습

전처리:
데이터 이미지 크기를 (64, 64)로 설정,
데이터셋 생성: image.jpg + label.csv,
픽셀 값: [정규화](#)

학습:
[ResNet50](#)을 활용한 단순 선형 회귀

3. 추론 결과

위성 이미지에 대한 풍속을 예측
(단위: Knot)

4. 기대 효과

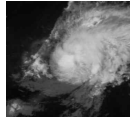
1. 폭풍의 풍속 예측을 통한 해안지대 피해 최소화
2. 재난안전관리에 활용 가능

- train(7,490개), test(6,898개) 폴더로 이루어져 있음
- 각각의 label.csv 파일에 해당 이미지의 풍속이 기록되어 있음

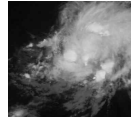
학습 데이터셋

train + label.csv
(7,490개)

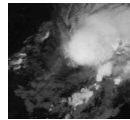
(단위 : Knot)



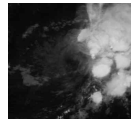
W.S : 53



W.S : 65



W.S : 50

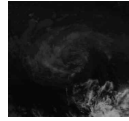


W.S : 30

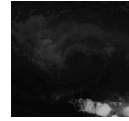
테스트 데이터셋

test + label.csv
(6,898개)

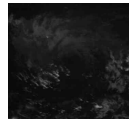
(단위 : Knot)



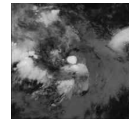
W.S : 30



W.S : 25



W.S : 21



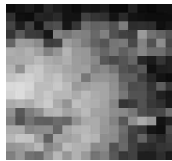
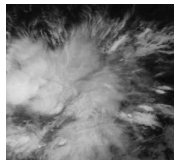
W.S : 45



데이터셋 생성

- 독립 변수 설정
 - 이미지가 저장된 폴더의 경로를 통해 이미지를 불러오기.
 - 불러온 이미지를 독립 변수로 설정
- 종속 변수 설정
 - label.csv 파일을 pandas DataFrame 형태로 불러오기
 - DataFrame의 'wind_speed' 속성을 종속 변수로 사용.

이미지 정규화



520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------

- 효율적인 모델 학습을 위해 각 픽셀을 0~1 사이의 값으로 만들어줌
- 한 픽셀이 0~255 사이의 값을 가짐으로 255로 나눠줌



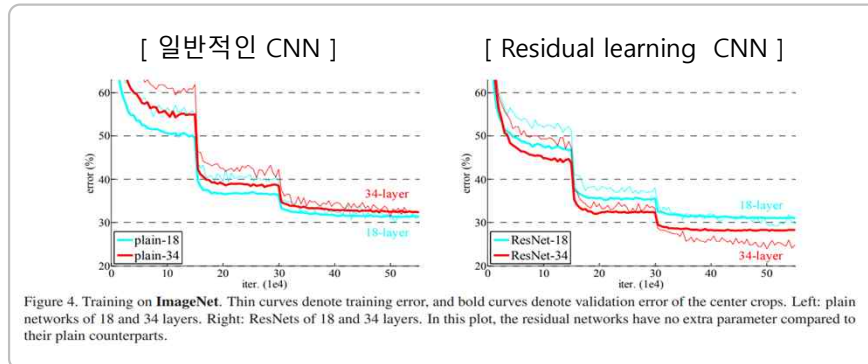
- 50개의 계층으로 구성된 컨벌루션 신경망으로 ImageNet 데이터베이스를 기반으로 사전 훈련된 신경망을 불러올 수 있음.
- ILSVRC 2015 분류 작업에서 3.57% 오류로 1위를 차지함

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



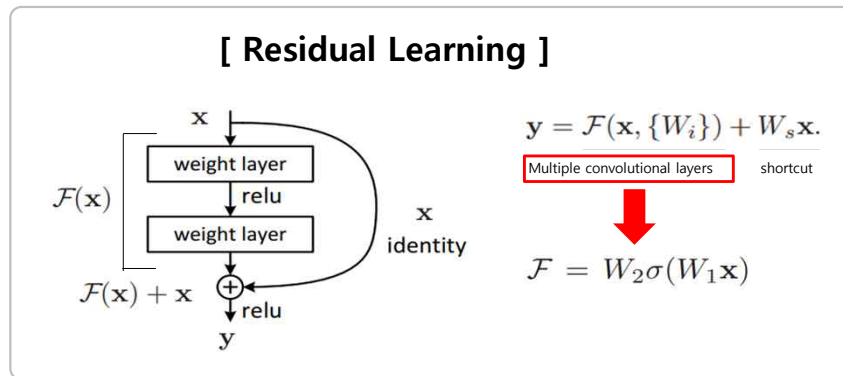
이미지 출처: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015), Deep Residual Learning for Image Recognition, Microsoft Research

- Layer를 깊게 쌓을수록 'gradient vanishing' 문제가 발생
- 이 문제를 해결하기 위해 Residual learning을 적용시킨 ResNet 모델 제안



이미지 출처: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015),
Deep Residual Learning for Image Recognition, Microsoft Research

- Skip connection을 이용한 Residual learning을 통해 gradient vanishing 문제 해결



설명 출처: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015),
Deep Residual Learning for Image Recognition, Microsoft Research

\mathbf{x} : input vector

\mathbf{y} : output vector

σ : ReLU Function

- $\mathcal{F}(x) + x$ 는 shortcut connection에서 수행

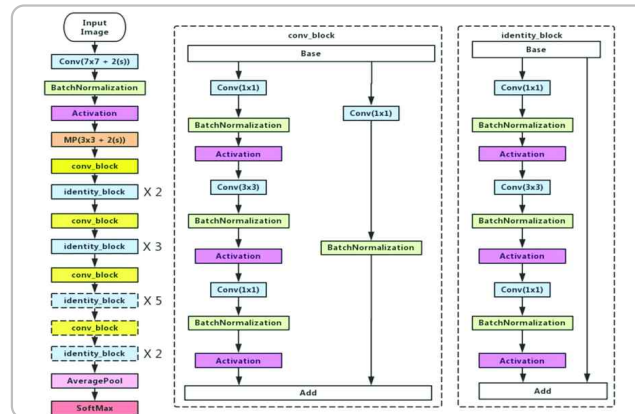
W : Convolution Layer

W_i : weight layer를 여러 번 사용 가능

W_s : Linear projection

- ResNet 모델의 한 종류로 총 50개의 층으로 구성되어있음
- ResNet50은 layer마다 다른 residual block 형태가 반복되어 학습되는 과정

ResNet50 구조



ResNet50의 Layer 살펴보기

코드 참조: <https://eremo2002.tistory.com/76>

Input) Input(shape=(224, 224, 3), dtype='float32', name='input')

Conv1_Layer(x)

1) Conv2D(64, (7, 7), strides=(2, 2))(x)

Conv2_Layer(x)

- 1) MaxPooling2D((3, 3), 2)(x)
 - 2) Conv2D(64, (1, 1), strides=(1, 1), padding='valid')(x)
 - 3) Conv2D(64, (3, 3), strides=(1, 1), padding='same')(x)
 - 4) Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
- 2)~4)을 총 3번 진행해 준다.

Conv3_Layer(x)

- 1) Conv2D(128, (1, 1), strides=(2, 2), padding='valid')(x)
- 2) Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
- 3) Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
- 4) Conv2D(128, (1, 1), strides=(1, 1), padding='valid')(x)
- 5) Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
- 6) Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)

- 4)~6)을 총 3번 진행해 준다.

Conv4_Layer(x)

- 1) Conv2D(256, (1, 1), strides=(2, 2), padding='valid')(x)
 - 2) Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
 - 3) Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
 - 4) Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
 - 5) Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
 - 6) Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
- 4)~6)을 총 5번 진행해 준다.

Conv5_Layer(x)

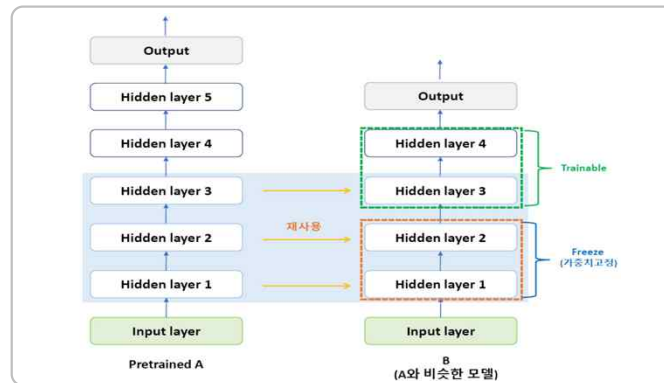
- 1) Conv2D(512, (1, 1), strides=(2, 2), padding='valid')(x)
 - 2) Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
 - 3) Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
 - 4) Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
 - 5) Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
 - 6) Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
- 4)~6)을 총 2번 진행해 준다.
- 7) GlobalAveragePooling2D(1000)(x)

Output) Dense(K, activation='softmax')(x)

전이학습

- 사전 훈련된 모델의 일부를 가져와 새 모델에서 재사용
- 전이학습을 통해 빠른 모델 학습 속도를 얻을 수 있음

전이학습 구조



이미지 출처: 강수철. (2020). 인공지능 전이학습과 응용 분야 동향. 정보통신 기획평가원(IITP). <https://t1.daumcdn.net/cfile/tistory/9972414A5E527AC226>

0_local_satellite_regression_requirement

- 로컬에 설치 되어야 할 패키지 버전
 - ✓ matplotlib 3.5.2
 - ✓ tensorflow 2.6.0
 - ✓ tensorflow-gpu 2.6.0
 - ✓ tensorflow-datasets 4.2.0
 - ✓ numpy 1.22.3
 - ✓ pandas 1.4.2
 - ✓ pillow 9.2.0
 - ✓ keras 2.6.0

0_local_satellite_regression.ipynb

- 로컬 개발 코드

- ✓ 로컬에서 주피터 노트북(Jupyter Notebook), 주피터 랩(JupyterLab) 또는 파이썬(Python)을 이용한다.
- ✓ 사이킷 런(scikit-learn), 텐서플로우(tensorflow), 파이토치(pytorch)를 사용하여 딥러닝 프로그램을 개발한다.
- ✓ 파일명: 0_local_satellite_regression.ipynb

- 로컬 개발 워크플로우(workflow)

- ✓ 로컬 개발 워크플로우를 다음의 4단계로 분리한다.

1. 데이터셋 준비(Data Setup)

- 로컬 저장소에서 전처리 및 학습에 필요한 학습 데이터셋을 준비한다.

2. 데이터 전처리(Data Preprocessing)

- 데이터셋의 분석 및 정규화(Normalization)등의 전처리를 수행한다.
- 데이터를 모델 학습에 사용할 수 있도록 가공한다.
- 추론과정에서 필요한 경우, 데이터 전처리에 사용된 객체를 meta_data 폴더 아래에 저장한다.

3. 학습 모델 훈련(Train Model)

- 데이터를 훈련에 사용할 수 있도록 가공한 뒤에 학습 모델을 구성한다.
- 학습 모델을 준비된 데이터셋으로 훈련시킨다.
- 정확도(Accuracy)나 손실(Loss)등 학습 모델의 성능을 검증한다.
- 학습 모델의 성능 검증 후, 학습 모델을 배포한다.
- 배포할 학습 모델을 meta_data 폴더 아래에 저장한다.

4. 추론(Inference)

- 저장된 전처리 객체나 학습 모델 객체를 준비한다.
- 추론에 필요한 테스트 데이터셋을 준비한다.
- 배포된 학습 모델을 통해 테스트 데이터에 대한 추론을 진행한다

0_local_satellite_regression.ipynb

IMPORTS

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from PIL import Image
import zipfile
```

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

```
from tensorflow.keras.applications.resnet import ResNet50
```

1. 데이터셋 준비(Data Setup)

```
zip_target_path = './meta_data'
os.makedirs(zip_target_path, exist_ok=True)

# dataset.zip 파일을 meta_data 폴더에 압축을 풀어준다.
zip_source_path = './dataset.zip'

extract_zip_file = zipfile.ZipFile(zip_source_path)
extract_zip_file.extractall(zip_target_path)

extract_zip_file.close()
```

2. 데이터 전처리(Data Preprocessing)

```
# 압축이 잘 풀렸다면 meta_data 폴더에 dataset 폴더가 생성이 되고
# train, test 폴더와 train_labels.csv, test_labels.csv 파일이 존재한다.
data_dir = './meta_data'
train_source_dir = os.path.join(data_dir, 'dataset/train')
test_source_dir = os.path.join(data_dir, 'dataset/test')

train_label_dir = os.path.join(data_dir, 'dataset/train_labels.csv')
test_label_dir = os.path.join(data_dir, 'dataset/test_labels.csv')

# label data를 출력하기 위해 label.csv를 pandas dataframe으로 가져온다.
train_df = pd.read_csv(train_label_dir)
test_df = pd.read_csv(test_label_dir)

# csv의 불필요한 인덱스 부분을 제거(DataFrame에 출력되기 때문에)
train_df = train_df.drop('Unnamed: 0', axis=1)
test_df = test_df.drop('Unnamed: 0', axis=1)

print(f"Total images for training: {len(train_df)}")
print(f"Total images for testing: {len(test_df)}")

train_df.sample(10)

# 학습 데이터셋의 wind_speed column의 정보를 한눈에 확인하기
train_df.describe()

test_df.describe()

# 이미지 샘플 보여주기

train_df_list = train_df['FileName'].tolist()

storm_ids = []
for storm_id in train_df_list:
    storm_ids.append(storm_id[-7:])

# train_df에 폭풍의 이름을 'image_id' column으로 추가
train_df['image_id'] = storm_ids
train_df.sample(10)

# 데이터셋에 비어있는 값이 있는지 확인하기
print('<train_df 살펴보기.>\n', train_df.isna().sum())
print('\n<test_df 살펴보기.>\n', test_df.isna().sum())
```



```

# 학습 데이터셋에 있는 폭풍 이미지와 실제 풍속을 출력
sample_images = []
train_imgs = [file for file in os.listdir(train_source_dir)]
for idx in np.random.randint(0, len(train_imgs), 5):
    sample_images.append(train_source_dir + '/' + train_imgs[idx])

fig, axes = plt.subplots(1, 5, figsize=(17, 8))

for idx, img in enumerate(sample_images):
    image = cv2.imread(os.path.join(img, 'image.jpg'))
    name = img.split('/')[-1][-7:]
    wind_s = train_df.loc[train_df['image_id'] == name]['wind_speed'].values[0]
    axes[idx].set_title('Pic: {} | Speed: {}'.format(name, wind_s))
    axes[idx].imshow(image)
    axes[idx].axis("off")

# 본 예제는 풍속을 예측한다. 학습을 위해 dataframe에서 'wind_speed'를 label로 지정하기.
train_labels = train_df['wind_speed'].values.tolist()
test_labels = test_df['wind_speed'].values.tolist()

image_size = (64, 64)
batch_size = 16

# 허리케인 image.jpg 파일에 해당되는 풍속을 label로 지정하기
# 학습 데이터셋 생성
# train 폴더의 80% 사용
print('Training Set:')
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_source_dir,
    labels = train_labels,
    validation_split=0.2,
    subset="training",
    seed = 123,
    image_size = image_size,
    color_mode = 'rgb',
    batch_size = batch_size
)
# 학습에 사용되는 검증 데이터셋 생성
# train 폴더의 20% 사용
print('Validation Set:')
validation_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_source_dir,
    labels = train_labels,
    validation_split=0.2,
    subset="validation",
    seed = 123,
    image_size = image_size,
    color_mode = 'rgb',
    batch_size = batch_size
)
# 테스트 데이터셋 생성
print('Test Set:')
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_source_dir,
    labels = test_labels,
    image_size = image_size,

```

```
    color_mode = 'rgb',  
    batch_size = batch_size  
)
```

```
# 이미지 정규화 전 상태
for image, lable in train_ds:
    print(image[0][0][0])
    break

# dataset의 이미지 정규화하기
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

# 학습에 필요한 train과 validation 데이터셋의 정규화를 진행한다.
train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)

# 이미지 정규화 후 상태
for image, lable in train_ds:
    print(image[0][0][0])
    break
```

3. 학습 모델 훈련(Train Model)

```
# ResNet50 모델 : VGGNet, GoogLeNet과 같은 모델들 보다 학습시키기 더 쉬운 모델
base_model = ResNet50(include_top=False)

# ResNet50 모델 전이학습하기.
for layer in base_model.layers[:40]:
    layer.trainable = False

for layer in base_model.layers[40:]:
    layer.trainable = True

base_model.summary()

# ResNet50 모델을 이용하여 Regression 하기 위해 모델을 만들어 준다.
# ResNet50의 기본 input_shape (224, 224, 3)를 빠른 학습을 위해 (64, 64, 3)으로 reshape한다.
def model_build_and_compile(base_model):
    inputs = tf.keras.Input(shape=(64, 64, 3))
    x = base_model(inputs, training=True)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.3)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation='linear')(x)
    model = tf.keras.Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mae')
    return model

model = model_build_and_compile(base_model)

model.summary()

# 모델 학습 (Train Model)
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
history = model.fit(train_ds,
                    epochs=20,
                    validation_data=validation_ds,
                    callbacks=[early_stop])
```

시각화

```
# loss의 시각화
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.plot(loss, label='Loss')
plt.plot(val_loss, label='val_loss')
plt.legend()
```

검증(Validation)

```
# 테스트 데이터 셋에서 배치사이즈 만큼의 이미지를 추론한다.
# 각각의 이미지에 대한 예측값과 실제값을 pred_y와 labels에 저장한다.
pred_y = []
labels = []
for img, label in test_ds:
    pred = model.predict(img)
    for i in range(batch_size):
        pred_y.append(pred[i][0])

    for i in range(len(label)):
        labels.append(label[i])

    break

# 저장된 예측값과 실제값을 이미지와 함께 출력해서 보여준다.
fig, axes = plt.subplots(1, 4, figsize=(17,8))

for idx, image in enumerate(img):
    if idx < 4:
        np_array = image.numpy().astype(np.uint8)
        image = Image.fromarray(np_array)
        predict = pred_y[idx]
        actual = labels[idx]
        axes[idx].set_title('Predict: {:.2f} | Actual: {}'.format(predict,actual))
        axes[idx].imshow(image)
        axes[idx].axis("off")
```

4. 추론(Inference)

```
# test_dataset.zip 파일을 meta_data 폴더에 압축을 풀어준다.
zip_test_source_path = './test_dataset.zip'

extract_zip_file = zipfile.ZipFile(zip_test_source_path)
extract_zip_file.extractall(zip_target_path)

extract_zip_file.close()

# test_dataset 폴더에 있는 이미지를 이용하여 추론한다.
inference_source_dir = os.path.join(data_dir, 'test_dataset')

inference_images = []
inference_imgs = [file for file in os.listdir(inference_source_dir)]
for idx in range(len(inference_imgs)):
    inference_images.append(inference_source_dir+'/'+inference_imgs[idx])

fig, axes = plt.subplots(1, 4, figsize=(17,8))

for idx,path in enumerate(inference_images):
    image = cv2.imread(path)
    reshaped_img = cv2.resize(image, (64, 64))
    pred_image = reshaped_img.reshape(-1, 64, 64, 3)
    pred_ws = model.predict(pred_image)
    name = path.split('/')[-1][-11:-4]
    axes[idx].set_title('Pic: {} | Speed: {:.2f}'.format(name,pred_ws[0][0]))
    axes[idx].imshow(image)
    axes[idx].axis("off")
```

1_local_platform_satellite_regression.ipynb

- ◆ 플랫폼 업로드를 쉽게하기 위한 로컬 개발 코드
 - ✓ T3Q.ai(T3Q.cep + T3Q.dl): 빅데이터/인공지능 통합 플랫폼
 - ✓ 플랫폼 업로드를 쉽게하기 위하여 로컬에서 아래의 코드(파일1)를 개발한다.
 - ✓ 파일(파일명): 1_local_platform_satellite_regression.ipynb
- 전처리 객체 또는 학습모델 객체
 - ✓ 전처리 객체나 학습모델 객체는 meta_data 폴더 아래에 저장한다.
- 데이터셋(학습 데이터/테스트 데이터)
 - ✓ 학습과 테스트에 사용되는 데이터를 나누어 관리한다.
 - ✓ 학습 데이터: dataset 폴더 아래에 저장하거나 dataset.zip 파일 형태로 저장한다.
 - ✓ 테스트 데이터: test_dataset 폴더 아래에 저장하거나 test_dataset.zip 파일 형태로 저장한다.

◆ 로컬 개발 워크플로우(workflow)

- ✓ 로컬 개발 워크플로우를 다음의 4단계로 분리한다.

1.데이터셋 준비(Data Setup)

- ✓ 로컬 저장소에서 전처리 및 학습에 필요한 학습 데이터셋을 준비한다.

2.데이터 전처리(Data Preprocessing)

- ✓ 데이터셋의 분석 및 정규화(Normalization)등의 전처리를 수행한다.
- ✓ 데이터를 모델 학습에 사용할 수 있도록 가공한다.
- ✓ 추론과정에서 필요한 경우, 데이터 전처리에 사용된 객체를 meta_data 폴더 아래에 저장한다.

3.학습 모델 훈련(Train Model)

- ✓ 데이터를 훈련에 사용할 수 있도록 가공한 뒤에 학습 모델을 구성한다.
- ✓ 학습 모델을 준비된 데이터셋으로 훈련시킨다.
- ✓ 정확도(Accuracy)나 손실(Loss)등 학습 모델의 성능을 검증한다.
- ✓ 학습 모델의 성능 검증 후, 학습 모델을 배포한다.
- ✓ 배포할 학습 모델을 meta_data 폴더 아래에 저장한다.

4.추론(Inference)

- ✓ 저장된 전처리 객체나 학습 모델 객체를 준비한다.
- ✓ 추론에 필요한 테스트 데이터셋을 준비한다.
- ✓ 배포된 학습 모델을 통해 테스트 데이터에 대한 추론을 진행한다.

```
# satellite_regression_preprocess.py
```

```
import logging
```

```
logging.basicConfig(level=logging.INFO)
```

```
def process_for_train(pm):
```

```
    exec_process(pm)
```

```
    logging.info('[hunmin log] the end line of the function [process_for_train]')
```

```
def init_svc(im, rule):
```

```
    meta_path = im.meta_path
```

```
    logging.info('[hunmin log] the end line of the function [init_svc]')
```

```
    return {"meta_path": meta_path, "rule": rule}
```

```
def transform(df, params, batch_id):
```

```
    logging.info('[hunmin log] df : {}'.format(df))
```

```
    logging.info('[hunmin log] df.shape : {}'.format(df.shape))
```

```
    logging.info('[hunmin log] type(df) : {}'.format(type(df)))
```

```
    logging.info('[hunmin log] the end line of the function [transform]')
```

```
    return df
```



```

# satellite_regression_preprocess_sub.py

import zipfile
import logging
import os

def exec_process(pm):

    logging.info('[hunmin log] the start line of the function [exec_process]')

    # 저장 파일 확인
    list_files_directories(pm.source_path)

    # pm.source_path의 dataset.zip 파일을
    # pm.target_path의 dataset 폴더에 압축을 풀어준다.
    my_zip_path = os.path.join(pm.source_path, 'dataset.zip')
    extract_zip_file = zipfile.ZipFile(my_zip_path)
    extract_zip_file.extractall(pm.target_path)
    extract_zip_file.close()

    # 저장 파일 확인
    list_files_directories(pm.target_path)

    logging.info('[hunmin log] the finish line of the function [exec_process]')

def list_files_directories(path):
    # Get the list of all files and directories in current working directory
    dir_list = os.listdir(path)

    logging.info('[hunmin log] Files and directories in {}'.format(path))
    logging.info('[hunmin log] dir_list : {}'.format(dir_list))

# satellite_regression_train.py

import logging

def train(tm):
    exec_train(tm)
    logging.info('[hunmin log] the end line of the function [train]')

def init_svc(im):
    params = exec_init_svc(im)
    logging.info('[hunmin log] the end line of the function [init_svc]')
    return { **params }

def inference(df, params, batch_id):
    result = exec_inference(df, params, batch_id)
    logging.info('[hunmin log] the end line of the function [inference]')
    return { **result }

```

```

# satellite_regression_train_sub.py

#IMPORTS
import numpy as np
import pandas as pd
import os
import base64
import matplotlib.pyplot as plt
import cv2
from PIL import Image

import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import logging

from tensorflow.keras.applications.resnet import ResNet50

logging.info(f'[hunmin log] tensorflow ver : {tf.__version__}')

# 사용할 gpu 번호를 적는다.
os.environ["CUDA_VISIBLE_DEVICES"]='0,1'

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus, 'GPU')
        logging.info('[hunmin log] gpu set complete')
        logging.info('[hunmin log] num of gpu: {}'.format(len(gpus)))

    except RuntimeError as e:
        logging.info('[hunmin log] gpu set failed')
        logging.info(e)

```

```

def exec_train(tm):

    logging.info('[hunmin log] the start line of the function [exec_train]')
    logging.info('[hunmin log] tm.train_data_path : {}'.format(tm.train_data_path))

    # 저장 파일 확인
    list_files_directories(tm.train_data_path)

    #####
    ## 1. 데이터셋 준비(Data Setup)
    #####
    # 데이터셋 경로 지정
    data_dir = tm.train_data_path
    train_source_dir = os.path.join(data_dir, 'dataset/train')
    test_source_dir = os.path.join(data_dir, 'dataset/test')

    train_label_dir = os.path.join(data_dir, 'dataset/train_labels.csv')
    test_label_dir = os.path.join(data_dir, 'dataset/test_labels.csv')

    # pandas dataframe 만들기
    train_df = pd.read_csv(train_label_dir)
    test_df = pd.read_csv(test_label_dir)

    train_labels = train_df['wind_speed'].values.tolist()
    test_labels = test_df['wind_speed'].values.tolist()

    image_size = (64, 64)
    batch_size = 16

    # 허리케인 image.jpg 파일에 해당되는 풍속을 label로 지정하기
    logging.info('[hunmin log] train dataset process start')
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        train_source_dir,
        labels = train_labels,
        validation_split=0.2,
        subset="training",
        seed = 123,
        image_size = image_size,
        color_mode = 'rgb',
        batch_size = batch_size
    )

    logging.info('[hunmin log] validation dataset process start')
    validation_ds = tf.keras.preprocessing.image_dataset_from_directory(
        train_source_dir,
        labels = train_labels,
        validation_split=0.2,
        subset="validation",
        seed = 123,
        image_size = image_size,
        color_mode = 'rgb',
        batch_size = batch_size
    )

```

```

logging.info('[hunmin log] test dataset process start')
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_source_dir,
    labels = test_labels,
    image_size = image_size,
    color_mode = 'rgb',
    batch_size = batch_size
)

# 학습에 필요한 train과 validation 데이터셋의 정규화를 진행한다.
logging.info('[hunmin log] Dataset normalization start')
train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)

#####
## 3. 학습 모델 훈련(Train Model)
#####
logging.info('=== modeling start ===')

# ResNet50 모델 import 하기
base_model = ResNet50(include_top=False)

for layer in base_model.layers[:40]:
    layer.trainable = False

for layer in base_model.layers[40:]:
    layer.trainable = True

# 모델 생성 & 컴파일 (Model Build and Compile)
# 단일 gpu 혹은 cpu학습
if len(gpus) < 2:
    model = model_build_and_compile(base_model)
# multi-gpu
else:
    strategy = tf.distribute.MirroredStrategy()
    logging.info('[hunmin log] gpu devices num {}'.format(strategy.num_replicas_in_sync))
    with strategy.scope():
        model = model_build_and_compile(base_model)

# 모델 학습 (Train Model)
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
history = model.fit(train_ds,
                    epochs=20,
                    validation_data=validation_ds,
                    callbacks=[early_stop])

```

```

#####
## 플랫폼 시각화
#####
## : 이 부분은 로컬에서 plotting하는 부분이지만
## 로컬환경과 플랫폼환경에서의 코드가 많이 달라 로컬코드에서는 생략되었습니다.
## 이 부분을 주석처리하고 실행하여도 플랫폼에서의 오류는 없습니다.
#####
#plot_metrics(tm, history)

#####
## 학습 모델 저장
#####

model.save(os.path.join(tm.model_path, 'regression_resnet_model.h5'))

# 저장 파일 확인
list_files_directories(tm.model_path)
logging.info('[hunmin log] the finish line of the function [exec_train]')

def exec_init_svc(im):
    #####
    ## 학습 모델 준비
    #####
    # load the model
    model = load_model(os.path.join(im.model_path, 'regression_resnet_model.h5'))
    return {'model':model}

def exec_inference(df, params, batch_id):
    logging.info('[hunmin log] the start line of the function [exec_inference]')
    inference = []
    # 학습 모델 준비
    model = params['model']
    logging.info('[hunmin log] model.summary() : {}'.format(model.summary()))

    logging.info('[hunmin log] data transform')
    for i in range(len(df)):
        data = df.iloc[i, 0]
        image_bytes = io.BytesIO(base64.b64decode(data))
        image = Image.open(image_bytes)
        numpy_image = np.array(image)
        numpy_image = numpy_image.astype(np.float32)/255.
        opencv_image = cv2.cvtColor(numpy_image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(opencv_image, (64,64))
        image = image.reshape(-1, 64, 64, 3)

```

```

#####
## 4. 추론(Inference)
#####
pred = model.predict(image)
prediction = pred.tolist()
inference.append(prediction[0][0])

result = {'inference':inference}
return result

def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

def model_build_and_compile(base_model):
    inputs = tf.keras.Input(shape=(64, 64, 3))
    x = base_model(inputs, training=True)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.3)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation='linear')(x)
    model = tf.keras.Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mae')

    logging.info('[hunmin log] model.summary() : ')
    model.summary(print_fn = logging.info)
    return model

def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

def model_build_and_compile(base_model):
    inputs = tf.keras.Input(shape=(64, 64, 3))
    x = base_model(inputs, training=True)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.3)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation='linear')(x)
    model = tf.keras.Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mae')

    logging.info('[hunmin log] model.summary() : ')
    model.summary(print_fn = logging.info)
    return model

def list_files_directories(path):
    # Get the list of all files and directories in current working directory
    dir_list = os.listdir(path)

    logging.info('[hunmin log] Files and directories in {} :'.format(path))
    logging.info('[hunmin log] dir_list : {}'.format(dir_list))

```

```
def plot_metrics(tm, history):
    for i in range(len(history.history['loss'])):
        metrix={}
        metrix['loss'] = history.history['loss'][i]
        metrix['accuracy'] = 0
        metrix['step'] = i
        tm.save_stat_metrics(metrix)

    eval_results={}
    eval_results['loss']= history.history['val_loss'][-1]

    tm.save_result_metrics(eval_results)
    logging.info('[hunmin log] loss curve plot for platform')
```

```

# PM 클래스: pm 객체
class PM:
    def __init__(self):
        self.source_path = './'
        self.target_path = './meta_data'

# TM 클래스: tm 객체
class TM:
    def __init__(self):
        self.train_data_path = './meta_data'
        self.model_path = './meta_data'

# IM 클래스: im 객체
class IM:
    def __init__(self):
        self.model_path = './meta_data'

# pm 객체
pm = PM()
print('pm.source_path:', pm.source_path)
print('pm.target_path: ', pm.target_path)

# tm 객체
tm = TM()
print('tm.train_data_path: ', tm.train_data_path)
print('tm.model_path: ', tm.model_path)

# im 객체
im = IM()
print('im.model_path: ', im.model_path)

# inferencne(df, params, batch_id) 함수 입력
params = {}
batch_id = 0

import io
import pandas as pd

# base64 encoded image
data= # '(11_1) Request_satellite_regression.txt' 파일 안에 있는 base64 형식의 이미지 이용

data = io.StringIO(data)
df = pd.DataFrame(data)
print('df: ', df)
print('df.dtypes:', df.dtypes)
df.columns

%%time
process_for_train(pm)

train(tm)

transform(df, params, batch_id)

params = init_svc(im)

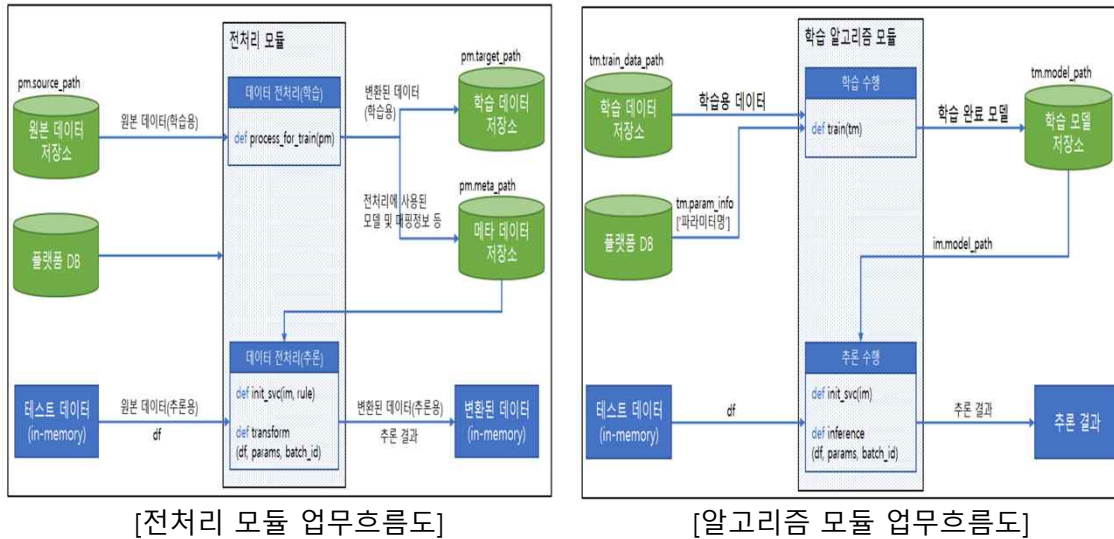
inference(df, params, batch_id)

```


2_platform_process

- 파일명 : satellite_regression_preprocess.py
 - ✓ from satellite_regression_preprocess_sub import exec_process
 - ✓ def process_for_train(pm):
 - ✓ def init_svc(im, rule):
 - ✓ def transform(df, params, batch_id):
- 파일명: satellite_regression_preprocess_sub.py
 - ✓ def exec_process(pm):
- 파일명: satellite_regression_train.py
 - ✓ from satellite_regression_train_sub import exec_train, exec_init_svc, exec_inference
 - ✓ def train(tm):
 - ✓ def init_svc(im):
 - ✓ def inference(df, params, batch_id):
- 파일명: satellite_regression_train_sub.py
 - ✓ def exec_train(tm):
 - ✓ def exec_init_svc(im):
 - ✓ def exec_inference(df, params, batch_id):
 1. 데이터셋 준비(Data Setup)
 2. 데이터 전처리(Data Preprocessing)
 3. 학습 모델 훈련(Train Model)
 4. 추론(Inference)

2_platform_process



0. 빅데이터/인공지능 통합 플랫폼 [T3Q.ai]

- 빅데이터 플랫폼 [T3Q.cep]
- 인공지능 플랫폼 [T3Q.dl]
- 빅데이터/인공지능 통합 플랫폼 [T3Q.ai (T3Q.cep + T3Q.dl)]

1. 머신러닝(Machine Learning)과 딥러닝(Deep Learning) 프로그래밍 패턴

- (1) 데이터셋 불러오기(Dataset Loading)
- (2) 데이터 전처리(Data Preprocessing)
 - 데이터 정규화(Normalization)
 - 학습과 테스트 데이터 분할(Train/Test Data Split) 등
- (3) 학습 모델 구성(Train Model Build)
- (4) 학습(Model Training)
- (5) 학습 모델 성능 검증(Model Performance Validation)
- (6) 학습 모델 저장(배포) 하기(Model Save)
- (7) 추론 데이터 전처리(Data Preprocessing)
- (8) 추론(Inference) 또는 예측(Prediction)
- (9) 추론 결과 데이터 후처리(Data Postprocessing)

2. 빅데이터/인공지능 통합 플랫폼[T3Q.ai]에서 딥러닝 프로그래밍 하고 인공지능 서비스 실시간 운용하기

- 6개의 함수로 딥러닝 프로그래밍 하고 인공지능 서비스 실시간 운용하기

- (1) process_for_train(pm) 함수
 - 데이터셋 준비(Dataset Setup)에 필요한 코드 작성
- (2) init_svc(im, rule) 함수
 - 전처리 객체 불러오기 에 필요한 코드 작성(생략 가능)
- (3) transform(df, params, batch_id) 함수
 - 추론 데이터 전처리(Data Preprocessing) 에 필요한 코드 작성(생략 가능)

- (4) train(tm) 함수
 - 데이터셋 불러오기(Dataset Loading)
 - 데이터 전처리(Data Preprocessing)
 - 학습 모델 구성(Train Model Build)
 - 학습(Model Training)
 - 학습 모델 성능 검증(Model Performance Validation)
 - 전처리 객체 저장
 - 학습 모델 저장(배포) 하기에 필요한 코드 작성
 - (5) init_svc(im) 함수
 - 전처리 객체 불러오기
 - 학습모델 객체 불러오기에 필요한 코드 작성
 - (6) inference(df, params, batch_id) 함수
 - 추론 데이터 전처리(Data Preprocessing)
 - 추론(Inference) 또는 예측(Prediction)
 - 추론 결과 데이터 후처리(Data Postprocessing)에 필요한 코드 작성
- =====

3. 전처리 모듈 관리, 학습 알고리즘 관리 함수 설명

1) 프로젝트 설정/전처리모듈 관리 함수

```
def process_for_train(pm):
```

```
    """
```

```
    (1) 입력: pm
```

```
        # pm.source_path: 학습플랫폼/데이터셋 관리 메뉴에서 저장한 데이터를 불러오는 경로
```

```
        # pm.target_path: 처리 완료된 데이터를 저장하는 경로
```

```
    (2) 출력: None
```

```
    (3) 설명:
```

```
        # 데이터셋 관리 메뉴에서 저장한 데이터를 불러와서 필요한 처리를 수행
```

```
        # 처리 완료된 데이터를 저장하는 기능, pm.target_path에 저장
```

```
        # train(tm) 함수의 tm.train_data_path를 통해 데이터를 불러와서 전처리와 학습을 수행
```

```
    """
```

```
def init_svc(im, rule):
```

```
    """
```

```
    (1) 입력: im, rule
```

```
    (2) 출력: None
```

```
    (3) 설명:
```

```
        # process_for_train(pm) 함수에서 저장한 전처리 객체와 데이터에 적용된 룰(rule)을  
        불러오는 기능
```

```
        # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
```

```
    """
```

```
    return {}
```

```
def transform(df, params, batch_id):
```

```
    """
```

```
    (1) 입력: df, params, batch_id
```

```
        # df: 추론모델관리와 추론API관리, 실시간 추론을 통해 전달되는 추론 입력 데이터  
        (dataframe 형태)
```

```
        # params: init_svc(im, rule) 함수의 리턴(return) 값을 params 변수로 전달
```

```
    (2) 출력: df
```

```
    (3) 설명:
```

```
        # df(추론 입력 데이터)에 대한 전처리를 수행한 후 전처리 된 데이터를
```

```
        inference(df, ...) 함수의 입력 df에 전달하는 기능
```

```
        # df(추론 입력 데이터)를 전처리 없이 inference(df, params, batch_id) 함수의 입력
```

```
df
```

```
        에 리턴(return)
```

```
    """
```

```
    return df
```

2) 프로젝트 설정/학습 알고리즘 관리 함수

```
def train(tm):
```

```
    """
```

```
    (1) 입력: tm
```

```
        # tm.train_data_path: pm.target_path에 저장한 데이터를 불러오는 경로
```

```
        # tm.model_path: 전처리 객체와 학습 모델 객체를 저장하는 경로
```

```
    (2) 출력: None
```

```
    (3) 설명:
```

```
        # pm.target_path에 저장한 데이터를 tm.train_data_path를 통해 데이터를 불러오는 기
```

```
        # 데이터 전처리와 학습 모델을 구성하고 모델 학습을 수행
```

```
        # 학습 모델의 성능을 검증하고 배포할 학습 모델을 저장
```

```
        # 전처리 객체와 학습 모델 객체를 저장, tm.model_path에 저장
```

```
        # init_svc(im) 함수의 im.model_path를 통해 전처리 객체와 학습 모델 객체를 준비
```

```
    """
```

```
def init_svc(im):
```

```
    """
```

```
    (1) 입력: im
```

```
        # im.model_path: tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을  
        불러오는 경로
```

```
    (2) 출력: 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
```

```
    (3) 설명:
```

```
        # tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는 기능
```

```
        # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
```

```
        # 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
```

```
        # 리턴(return) 값을 inference(df, params, batch_id) 함수의 입력 params 변수로 전달
```

```
    """
```

```
    return { "model": model, "param": param }
```

```
def inference(df, params, batch_id):
```

```
    """
```

```
    (1) 입력: df, params, batch_id
```

```
        # df: transform(df, params, batch_id)함수의 리턴(return) 값으로 전달된 df, 추론 입력  
        데이터 (dataframe 형태)
```

```
        # params init_svc(im) 함수의 return 값을 params 변수로 전달
```

```
            ## 학습 모델 객체 사용 예시    model=params['model']
```

```
            ## 전처리(pca) 객체 사용 예시    pca=params['pca']
```

```
    (2) 출력: 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
```

```
    (3) 설명:
```

```
        # 전처리 객체를 사용하여 df(추론 입력 데이터)에 대한 전처리 수행
```

```
        # 배포된 학습 모델(model)을 사용하여 df(추론 입력 데이터)에 추론(예측)을 수행
```

```
        # 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
```

```
    """
```

```
    return {"inference": result}
```

```
=====
```

4. 전처리 모듈 관리, 학습 알고리즘 관리 함수 설명(AI 훈민정음 프로젝트)

1) 프로젝트 설정/전처리모듈 관리 함수(AI 훈민정음 프로젝트)

```
import logging
```

```
def process_for_train(pm):
```

```
    """
```

```
    (1) 입력: pm
```

```
        # pm.source_path: 학습플랫폼/데이터셋 관리 메뉴에서 저장한 데이터를 불러오는 경로
```

```
        # pm.target_path: 처리 완료된 데이터를 저장하는 경로
```

```
    (2) 출력: None
```

```
    (3) 설명:
```

```
        # 데이터셋 관리 메뉴에서 저장한 데이터를 불러와서 필요한 처리를 수행
```

```
        # 처리 완료된 데이터를 저장하는 기능, pm.target_path에 저장
```

```
        # train(tm) 함수의 tm.train_data_path를 통해 데이터를 불러와서 전처리와 학습을 수행
```

```
    (4) 추가 설명:
```

```
        # 함수 구조는 원형대로 유지
```

```
        # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_process)로 정의하여 사용함
```

```
        # 함수명                                서브함수명
```

```
        # process_for_train(pm)                exec_process(pm)
```

```
        # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
```

```
    """
```

```
    exec_process(pm)
```

```
    logging.info('[hunmin log] the end line of the function [process_for_train]')
```

```
def init_svc(im, rule):
```

```
    """
```

```
    (1) 입력: im, rule
```

```
    (2) 출력: None
```

```
    (3) 설명:
```

```
        # process_for_train(pm) 함수에서 저장한 전처리 객체와 데이터에 적용된 룰(rule)을  
        불러오는 기능
```

```
        # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
```

```
    """
```

```
    return {}
```

```
def transform(df, params, batch_id):
```

```
    """
```

```
    (1) 입력: df, params, batch_id
```

```
        # df: 추론모델관리와 추론API관리, 실시간 추론을 통해 전달되는 추론 입력 데이터  
        (dataframe 형태)
```

```
        # params: init_svc(im, rule) 함수의 리턴(return) 값을 params 변수로 전달
```

```
    (2) 출력: df
```

```
    (3) 설명:
```

```
        # df(추론 입력 데이터)에 대한 전처리를 수행한 후 전처리 된 데이터를 inference(df, ...)  
        함수의 입력 df에 전달하는 기능
```

```
        # df(추론 입력 데이터)를 전처리 없이 inference(df, params, batch_id) 함수의 입력 df에  
        리턴(return)
```

```
    (4) 추가 설명:
```

```
        # 함수 구조는 원형대로 유지
```

```
        # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
```

```
    """
```

```
    logging.info('[hunmin log] the end line of the function [transform]')
```

```
    return df
```

```
import logging
```

```
def train(tm):
```

///

(1) 입력: t_m

tm.train_data_path: pm.target_path에 저장한 데이터를 불러오는 경로

tm.model_path: 전처리 객체와 학습 모델 객체를 저장하는 경로

(2) 출력: None

(3) 설명:

pm.target_path에 저장한 데이터를 tm.train_data_path를 통해 데이터를 불러오는 기능

데이터 전처리와 학습 모델을 구성하고 모델 학습을 수행

학습 모델의 성능을 검증하고 배포할 학습 모델을 저장

```
# 전처리 객체와 학습 모델 객체를 저장, tm.model_path에 저장
```

init_svc(im) 함수의 im.model_path를 통해 전처리 객체와 학습 모델 객체를 준비

(4) 추가 설명:

함수 구조는 원형대로 유지

실질적인 기능을 하는 함수를 서브모듈 함수(exec_train)로 정의하여 사용함

함수명

서브함수명

```
# train(tm)                                exec_train(tm)
```

함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행

III III III

```
exec_train(pm)
```

```
logging.info('[hunmin log] the end line of the function [train]')
```

```
def init_svc(im):
```

■■ ■■ ■■

(1) 입력: im

im.model_path: tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는 경로

(2) 출력: 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)

(3) 설명:

tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는 기능

전처리 객체, 룰(rule) 불러오기 기능 없이 처리

전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)

리턴(return) 값을 inference(df, params, batch_id) 함수의 입력 params 변수로 전달

(4) 추가 설명:

함수 구조는 원형대로 유지

실질적인 기능을 하는 함수를 서브모듈 함수(exec_init_svc)로 정의하여 사용함

함수명

서브함수명

```
# init_svc(im)                                exec_init_svc(im)
```

함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행

|| || ||

```
params = exec init svc(im)
```

```
logging.info('[hunmin log] the end line of the function [init_svc]')
```

```
return {**params}
```

```
def inference(df, params, batch_id):
```

```
    """
```

```
    (1) 입력: df, params, batch_id
```

```
    # df: transform(df, params, batch_id)함수의 리턴(return) 값으로 전달된 df, 추론 입력  
    데이터(dataframe 형태)
```

```
    # params: init_svc(im) 함수의 리턴(return) 값을 params 변수로 전달
```

```
    ## 학습 모델 객체 사용 예시    model=params['model']
```

```
    ## 전처리(pca) 객체 사용 예시    pca=params['pca']
```

```
    (2) 출력: 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
```

```
    (3) 설명:
```

```
    # 전처리 객체를 사용하여 df(추론 입력 데이터)에 대한 전처리 수행
```

```
    # 배포된 학습 모델(model)을 사용하여 df(추론 입력 데이터)에 추론(예측)을 수행
```

```
    # 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
```

```
    (4) 추가 설명:
```

```
    # 함수 구조는 원형대로 유지
```

```
    # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_inference)로 정의하여 사용함
```

```
    # 함수명                                서브함수명
```

```
    # inference(df, params, batch_id)      exec_inference(df, params, batch_id)
```

```
    # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
```

```
    """
```

```
    result = exec_inference(df, params, batch_id)
```

```
    logging.info('[hunmin log] the end line    of the function [inference]')
```

```
    return {**result}
```

III. 수행절차

수행절차 소개

- 01) T3Q.cep_데이터수집 파이프라인_satellite_regression : 해당 예제에서는 수행 절차 없음
- 02) T3Q.cep_데이터변환 파이프라인_satellite_regression : 해당 예제에서는 수행 절차 없음
- 03) T3Q.dl_프로젝트 설정_실행환경 관리_satellite_regression
- 04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_satellite_regression
- 05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_satellite_regression
- 06) T3Q.dl_학습플랫폼_데이터셋 관리_satellite_regression
- 07) T3Q.dl_학습플랫폼_전처리 모델 설계_satellite_regression
- 08) T3Q.dl_학습플랫폼_전처리 모델 관리_satellite_regression
- 09) T3Q.dl_학습플랫폼_학습모델 설계_satellite_regression
- 10) T3Q.dl_학습플랫폼_학습모델 관리_satellite_regression
- 11) T3Q.dl_추론플랫폼_추론모델 관리_satellite_regression
- 12) T3Q.dl_추론플랫폼_추론API관리_satellite_regression
- 13) T3Q.cep_실시간 추론 파이프라인_satellite_regression



실행환경 추가 내용 및 절차

1) Requirements

```
# User Requirements.
```

2) Dockerfile

```
FROM tensorflow/tensorflow:2.4.1-gpu
```

```
ARG DEBIAN_FRONTEND=noninteractive
```

```
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
A4B469963BF863CC
```

```
RUN apt-get update && apt-get install -y wget &
```

```
python3.8 &
```

```
python3-pip &
```

```
python3-dev &
```

```
python3.8-dev &
```

```
postgresql &
```

```
libpq-dev
```

```
RUN pip3 install --upgrade pip
```

```
# libraries for operservice
```

```
RUN pip install --no-input kubernetes pygresql pyjwt pyarrow pandas &  
flask flask-sqlalchemy flask-cors flask-bcrypt flask-migrate flask-  
restful flask-rest-jsonapi
```

```
# opencv
```

```
RUN apt-get -y install libgl1-mesa-glx
```

```
# generic libraries
RUN pip install --no-input numpy==1.19.5 ₩
    torch scikit-learn imbalanced-learn xgboost ₩
    fastai keras keras-preprocessing keras-vis ₩
    matplotlib pillow nltk ₩
    opencv-contrib-python opencv-python openpyxl imageio
pretty_midi ₩
pickleshare pip-tools protobuf psutil pycopg2 PyYAML ₩
pathlib requests requests-oauthlib rsa ₩
tensorboard tensorboard-data-server tensorboard-plugin-wit ₩
sqlalchemy grpcio tqdm urllib3 xlrd xlwt lightgbm
```

```
# libraries for operservice
RUN pip install --no-input category_encoders
```

```
# prevent making cache not to preserve previous train code
ENV PYTHONDONTWRITEBYTECODE 1
```

```
# For User
ADD ./requirements.txt /work/requirements.txt
RUN pip install -r /work/requirements.txt
```

```
#COPY ./work /work
USER root
```

```
RUN mkdir /mldatas
RUN mkdir /data
RUN mkdir /data/aip
RUN mkdir /data/aip/logs
```

```
WORKDIR /work
```

=====

추가된 실행 환경 [저장] 하고, [배포하기] 시작 , 완료 후 [로그] 에서 확인

≡ 실행환경 목록

실행환경명

✔
satellite_regression_env
더보기

타입 Python

satellite_regression_env

배포하기

로그

로그 확인 ↺ 2022-09-23 10:39:29 ✕

```
2022-09-23 00:11:25,588 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"d5f0eff44d91"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"38482f47bc58"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"daf57e1d9792"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"53194dce1444"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"ef8330bcc944"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"964ee116c0c0"}
2022-09-23 00:11:25,593 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"7a694df0ad6c"}
2022-09-23 00:11:25,594 [ INFO] root: {"status":"Preparing","progressDetail":{},"id":"3fd9df553184"}
```

04) T3Q.ai 프로젝트 설정_전처리 모듈 관리_satellite_regression

- 프로젝트 설정>>전처리모듈관리
 - 전처리모듈 관리>>  [전처리 모듈 추가] 실행
 - 전처리모듈 등록 시 순서와 입력 정보
 - ① 기본정보
 - 전처리명: satellite_regression_premodule
 - 실행환경 선택 : satellite_regression_env
 - GPU지원 : 체크
 - ② 입력 형태: file
 - ③ 출력 형태: default
 - ④ 파라미터
 - ⑤ 소스 코드 : T3Q.ai_platform_satellite_regression_preprocess.zip [파일업로드]
 - satellite_regression_preprocess.py (실행모듈 선택)
 - satellite_regression_preprocess_sub.py
 - platform_process.txt
 - LICENSE.txt
 - README.txt

T3Q

41

- 전처리모듈 등록 시 순서와 입력 정보

① 기본정보

전처리명: satellite_regression_premodule

실행환경 선택 : satellite_regression_env

GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default

전처리모듈 등록

기본 정보

satellite_regression_premodule

satellite_regression_env

ON ☐ GPU 지원

입력 형태

file

출력 형태

default

파라미터

이름을 입력해주세요

값을 입력해주세요

04) T3Q.ai 프로젝트 설정_전처리 모듈 관리_satellite_regression

- 프로젝트 설정>>전처리모듈관리
 - 전처리모듈 관리>> [전처리 모듈 추가] 실행
 - 전처리모듈 등록 시 순서와 입력 정보
 - ① 기본정보
 - 전처리명: satellite_regression_premodule
 - 실행환경 선택 : satellite_regression_env
 - GPU지원 : 체크
 - ② 입력 형태: file
 - ③ 출력 형태: default
 - ④ 파라미터
 - ⑤ 소스 코드 : T3Q.ai_platform_satellite_regression_preprocess.zip [파일업로드]
 - satellite_regression_preprocess.py (실행모듈 선택)
 - satellite_regression_preprocess_sub.py
 - platform_process.txt
 - LICENSE.txt
 - README.txt

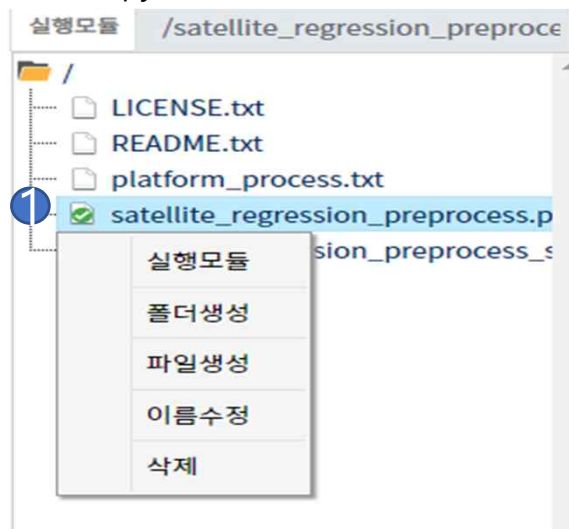
T3Q

42

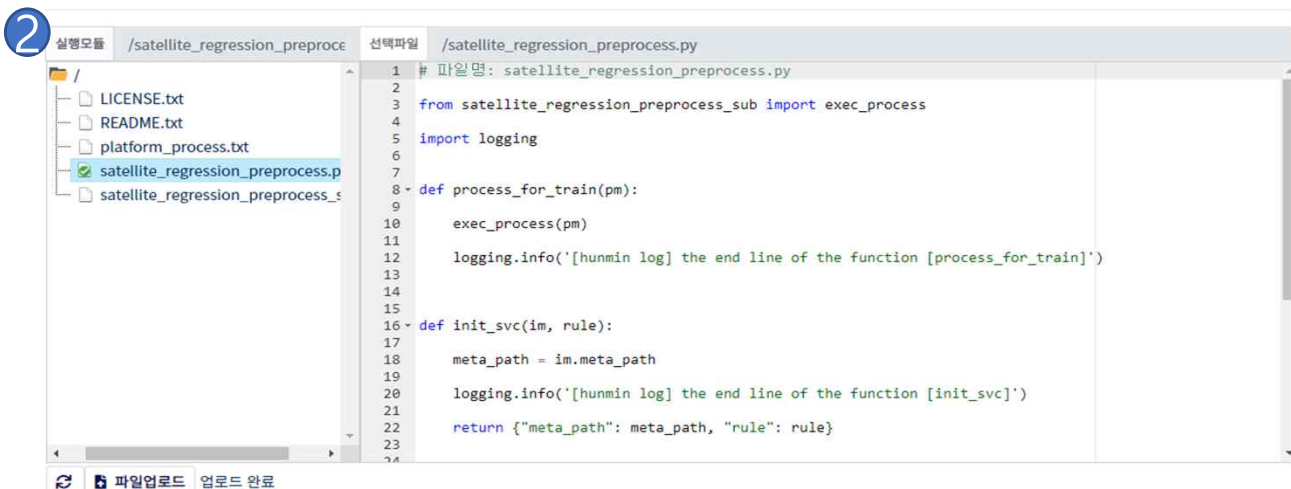
전처리모듈 등록 시 순서와 입력 정보

- ⑤ 소스 코드 : T3Q.ai_platform_satellite_regression_preprocess.zip [파일업로드] 누름

- satellite_regression_preprocess.py ([실행모듈] 선택)
- satellite_regression_preprocess_sub.py
- platform_process.txt
- LICENSE.txt
- README.txt



소스 코드



05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_satellite_regression

■ 프로젝트 설정 >> 학습 알고리즘 관리

- 학습 알고리즘 관리>>

[+ \[알고리즘 추가\] 실행](#)

- 학습 알고리즘 등록 시 순서와 입력 정보

- ① 기본정보: 다음과 같이 입력
 - 알고리즘명: satellite_regression_train
 - 설명 :satellite_regression_train
 - 카테고리 : Regression
 - 실행환경 : satellite_regression_env
 - GPU 지원 : 체크

- ② 공통 파라미터: 아래 1가지 항목 사용
 - 초기화 방법 : 사용

- ③ 모델 파라미터

- ④ 시각화 설정: 아래 1개 항목만 체크
 - Loss : 체크

1. 학습 알고리즘 등록

1. 기본 정보

알고리즘명: satellite_regression_train 카테고리: Regression

설명: satellite_regression_train 실행환경: satellite_regression_env

☒ GPU 지원

2. 공통 파라미터

☒ 초기화 방법 ☐ 학습률 ☐ Dropout Rate ☐ 배치 크기

☐ 랜덤 seed ☐ 랜덤 초기화 ☐ 학습률 스케줄러 ☐ 배치 크기

4. 시각화 설정

결과 차트

☐ Accuracy ☒ Loss ☐ Confusion Matrix ☐ MAPE

☐ Precision/Recall/F1-score ☐ PCA 2D

05) T3Q.ai 프로젝트 설정_학습 알고리즘 관리_satellite_regression

- 프로젝트 설정 >> 학습 알고리즘 관리
 - 학습 알고리즘 관리>> [알고리즘 추가] 실행
 - 학습 알고리즘 등록 시 순서와 입력 정보

⑤ 소스코드 업로드 : [파일업로드]

: T3Q.ai_platform_satellite_regression_train.zip

- satellite_regression_train.py
- satellite_regression_train_sub.py
- platform_process.txt
- LICENSE.txt
- README.txt

: 실행 모듈 설정 >> [저장]

- /satellite_regression_train.py 실행 모듈 지정
- [저장]을 누른다.

■ 학습 알고리즘 관리

학습알고리즘명, 문제유형, 실행환경 검색

satellite_regression_train

더보기

satellite_regression_train

Regression

satellite_regression_env

2022-09-02 14:11:05

소스 코드

```

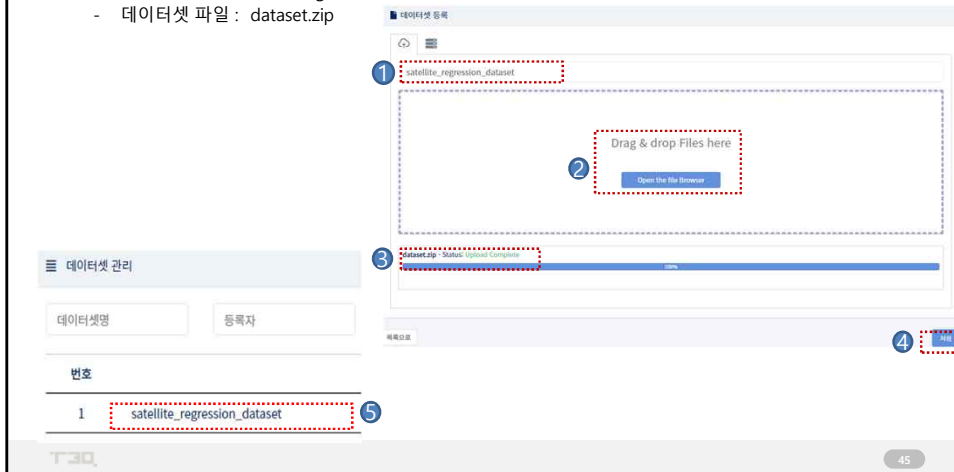
1 #!/usr/bin/env python
2
3 from satellite_regression_train_sub import exec_train, exec_init_svc, exec_inference
4 import logging
5
6
7 def train(train):
8     exec_train(train)
9
10
11     logging.info('human log the end line of the function [train]')
12
13
14
15 def init_svc(in):
16     params = exec_init_svc(in)
17
18     logging.info('human log the end line of the function [init_svc]')
19
20     return { "params" }
21
22
23
24
25 def inference(of, params, batch_id):
26     result = exec_inference(of, params, batch_id)
27
28     logging.info('human log the end line of the function [inference]')
29
30     return { "result" }
31
32
33
34

```

파일업로드 | 항상 최신으로 업로드되었습니다

06) T3Q.dl_학습플랫폼_데이터셋 관리_satellite_regression

- 학습플랫폼 >> 데이터셋 관리
- 데이터셋 등록 절차
 - 데이터셋 관리>> **등록** 실행
 - 데이터셋 명 : satellite_regression_dataset
 - 데이터셋 파일 : dataset.zip



07) T3Q.dl_학습플랫폼_전처리 모델 설계_satellite_regression

- 학습플랫폼 >> 전처리 모델 설계
- 전처리 모델 설계 절차
 - ① 학습플랫폼 >> 데이터셋 관리
 - 데이터셋 명 : satellite_regression_dataset 선택
 - ② satellite_regression_dataset 아래의 [전처리 모델 설계] 선택

데이터셋 관리

데이터셋명

등록자

번호

1 satellite_regression_dataset

1

satellite_regression_dataset

2022-09-02 14:11:58 다운로드

File explorer

281.28 MB

/ (1파일, 0 디렉토리)

dataset.zip

dataset.zip
281.28 MB

목록으로

삭제

전처리 모델 설계

2

T3Q

46

07) T3Q.dl_학습플랫폼_전처리 모델 설계_satellite_regression

■ 학습플랫폼 >> 전처리 모델 설계

② 전처리 모델 설계 절차

Step1. 기본 정보

- 모델명: satellite_regression_premodel

Step2. 데이터셋 등록:

- 참조데이터셋 : satellite_regression_dataset

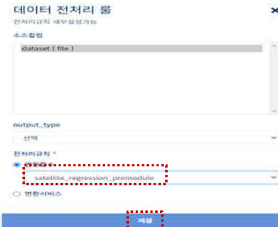
Step3. ID/LABEL 지정

Step4. 전처리 규칙 정보

- [전처리 규칙 등록] 선택

③ 전처리 규칙 등록 시 절차

- 소스컬럼 : dataset(file) 선택
- 변환 함수 : satellite_regression_premodule 선택 후 [저장]



1 전처리 모델 설계

Step 1. 기본 정보
모델명을 입력하고, 모델명 8 문자 이하로 설정합니다.

satellite_regression_premodel

Step 2. 데이터셋 등록
참조 데이터셋을 등록하고, 모델명 8 문자 이하로 설정합니다.

satellite_regression_dataset

Step 3. ID/LABEL 지정
모델명, 데이터셋명, ID/LABEL, 변환 함수를 각각 Step 1, 2, 3, 4 단계에서 지정합니다.

원래 사용 | 현재 새로

사용처	원형	데이터 분포도	시각화	ID	LABEL
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

dataset

file

Showing 1 to 1 of 1 entries

Step 4. 전처리 규칙 등록
데이터 전처리 규칙을 등록하고, 모델명 8 문자 이하로 설정합니다.

소스컬럼 | 전처리 규칙 | 전처리 규칙 상세정보 | 확인

No data available in table

Showing 0 to 0 of 0 entries

Prev Next

III. 수행절차

08) T3Q.dl_학습플랫폼_전처리 모델 관리_satellite_regression

- 학습플랫폼 >> 전처리 모델 관리
- 전처리 모델 설계 상세

Step1. 기본 정보

Step 1. 기본 정보

모델명: satellite_regression_premodel

참조 데이터셋: satellite_regression_dataset

Step2. 데이터셋 컬럼정보

Step 2. 데이터셋 컬럼정보

데이터셋: dataset

Step3. 전처리 룰 정보

Step 3. 전처리 룰 정보

전처리룰: satellite_regression_preprocess

Step4. 전처리 실행정보

Step 4. 전처리 실행정보

전처리 실행정보

이름: satellite_regression_preprocess

처리시작일: 2022-09-02 14:14:28

처리종료일: 2022-09-02 14:15:16

상태: 실행 완료

로그: [로그]

전처리 상세 *

- 전처리 [상세] 버튼 누름
- 진행상태 [완료] 확인
- 0_satellite_regression_preprocess - [로그] 아래 [보기]를 통해 성공/오류 확인

진행상황 확인

현재 전처리의 진행상황 확인이 가능합니다.

이름: satellite_regression_preprocess

처리시작일: 2022-09-02 14:14:28

처리종료일: 2022-09-02 14:15:16

상태: 실행 완료

로그: [로그]

다음 단계

- 학습플랫폼 >> 전처리 모델 관리
 - 전처리 모델 설계 상세
- Step1. 기본 정보
- 모델명: satellite_regression_premodel
 - 참조데이터셋 : satellite_regression_dataset
- Step2. 데이터셋 컬럼정보
- Step3. 전처리 룰 정보
- Step4. 전처리 실행정보
- 전처리 상세
- 전처리 상세 버튼 누름
 - 진행상황 확인
 - 0_satellite_regression_preprocess - [로그] 아래 [보기] 누름
- [학습 모델 설계] 선택하여 다음 단계 진행

로그 확인

마지막 로딩된 시간 : 2022-09-23 14:08:17

```
2022-09-02 05:14:28,702 [ INFO] root: ### preprocessing start ###
2022-09-02 05:14:28,702 [ INFO] root: params=[{'pre_dataset_id': 1044, 'rule': {'source_column': ['dataset'], 'rule': 'preModel', 'rule_type': 'satellite_regression_preprocess', 'mod': 'U', 'param': {}, 'rule_no': '0', 'source_type': 'file', 'module_info': {'deploy_dt': '2022-09-02 14:09:51', 'template': 'Python', 'version': '1.0', 'status': 'deployed', 'image_name': 369, 'module_name': 'satellite_regression_preprocess'}}, 'output_type': ['default']}, {'do_fit': True, 'test_no': None, 'test_dataset_path': None, 'log_path': '/data/aip/logs'}]
2022-09-02 05:14:28,738 [ WARN] root: datasource_repo_id : 157, datasource_repo_obj : <DataSourceRepo 157>, repo_type : file
2022-09-02 05:14:28,754 [ INFO] root: module_path=/data/aip/logs/t3qai/premodule/premodule_511/1
2022-09-02 05:14:28,790 [ INFO] root: dp_module=<module 'satellite_regression_preprocess' from '/data/aip/logs/t3qai/premodule/premodule_511/1/satellite_regression_preprocess.py'>
2022-09-02 05:14:28,790 [ INFO] root: [hunmin log] the start line of the function [exec_process]
2022-09-02 05:14:28,790 [ INFO] root: [hunmin log] pm.source_path : /data/aip/file_group/t3qai/777
2022-09-02 05:14:28,792 [ INFO] root: [hunmin log] Files and directories in /data/aip/file_group/t3qai/777 :
2022-09-02 05:14:28,792 [ INFO] root: [hunmin log] dir_list : ['dataset.zip']
2022-09-02 05:15:16,882 [ INFO] root: [hunmin log] Files and directories in /data/aip/dataset/t3qai/pm/pm_1035/ds_1044 :
2022-09-02 05:15:16,882 [ INFO] root: [hunmin log] dir_list : ['dataset']
```

III. 수행절차

09) T3Q.dl_학습플랫폼_학습모델 설계_satellite_regression

- 학습플랫폼 >> 학습모델 설계
- 학습모델 설계 상세 과정

Step1. 기본 정보

Step 1. 기본 정보

학습 모델 이름:

전처리 모델:

평가 방법:

Step2. 모델 설계

Step 2. 모델 설계

문제유형: 알고리즘:

평가방법:

Step3. 상세 설계

Step 3. 상세 설계

알고리즘 선택 시 상세 설계가 가능합니다. (step 2. 먼저 진행)

공통 파라미터

초기화방법:

목록으로

T3Q 49

학습플랫폼 >> 학습모델 설계

1. AI 훈민정음 >> 학습플랫폼 >> 학습모델 설계 상세 과정

1) Step 1. 기본 정보

학습모델명: satellite_regression_trainmodel

전처리모델: satellite_regression_premodel

[사용] 체크: satellite_regression_premodel

2) Step 2. 모델 설계

문제유형: Regression

알고리즘: satellite_regression_train

평가방법: #Train-Test Split : 80

None

3) Step 3. 상세 설계

알고리즘 선택 시 상세 설계가 가능합니다. (step 2. 먼저 진행)

(1) 공통 파라미터

초기화방법 : Xavier uniform

4) [저장] 누름

2. AI 훈민정음 >> 학습플랫폼 >> [학습모델 관리] 에서 등록된 학습 모델 확인

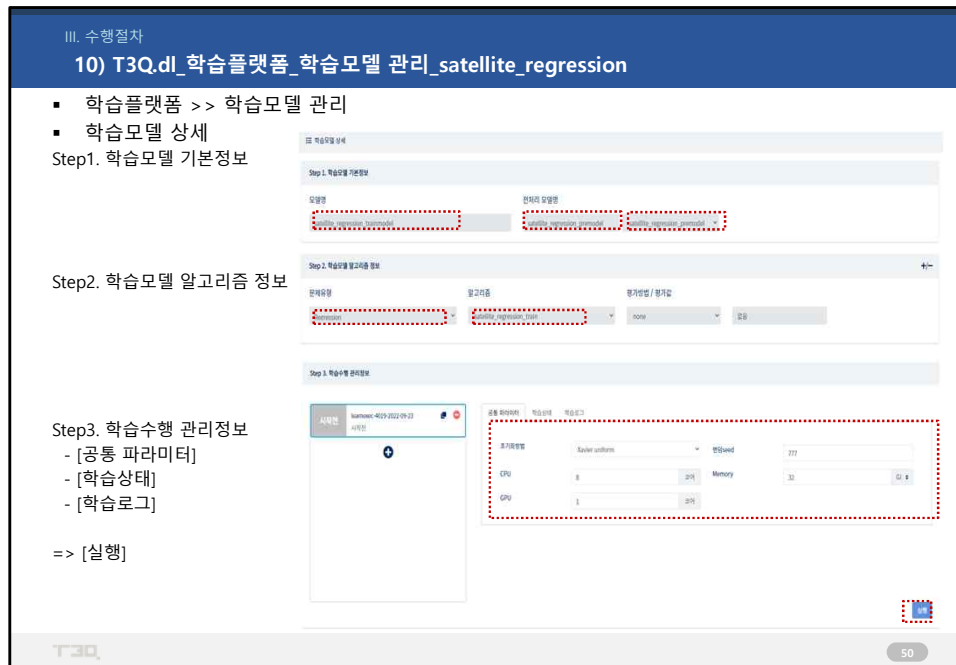
학습 모델 관리

카테고리 선택: 알고리즘 선택: 학습명 검색: 등록자:

번호	카테고리	알고리즘	학습명	등록자	실행정보			학습상태
					수행시작일시	수행종료일시	결과(Accuracy)	
1	Regression	satellite_regression_train	satellite_regression_trainmodel	kimyj6032	-	-	-	<input type="button" value="시작"/>

Showing 1 to 1 of 1 entries

Prev **1** Next



학습플랫폼 >> 학습모델 관리

1. 학습플랫폼 >> 학습모델 관리

학습 모델 관리								
카테고리 선택		알고리즘 선택		학습명 검색	등록자	조회 Clear		
번호	카테고리	알고리즘	학습명	등록자	실행정보			학습상태
					수행시작일시	수행종료일시	결과(Accuracy)	
1	Regression	satellite_regression_train	satellite_regression_trainmodel	kimy6032	-	-	-	시작전
Showing 1 to 1 of 1 entries								
								Prev 1 Next

2 학습모델 상세

1) Step 1. 학습모델 기본정보

모델명 satellite_regression_trainmodel

전처리 모델명 satellite_regression_premodel satellite_regression_premodel

2) Step 2. 학습모델 알고리즘 정보

문제유형 Regression

알고리즘 satellite_regression_train

평가방법 / 평가값 none 없음

3) Step 3. 학습수행 관리정보

(1) 공통 파라미터

초기화방법

Xavier uniform

랜덤seed

777

CPU

8 코어

Memory

32 Gi

GPU

1 코어

(2) 학습상태

학습상태 시작전 [- ~ -]

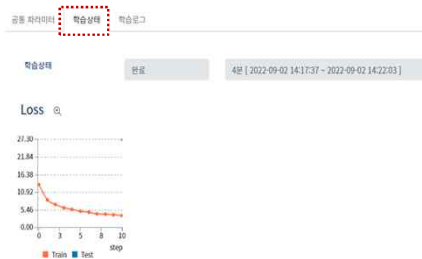
Loss

[실행] 버튼 누름

10) T3Q.dl_학습플랫폼_학습모델 관리_satellite_regression

- 학습플랫폼 >> 학습모델 관리
- 학습모델 상세
- [실행] 완료 후
 - [학습상태]

- [학습로그]



- 모델 배포 : [모델 배포] 버튼 누르고 [배포] 누름

학습모델 배포

모델명을 입력하고 추론모델로 배포가 가능합니다.(50자 이내)

satellite_regression_trainmodel-2022-09-23

[illegible]

圖 4 右側開閉器

9024 • J. Neurosci., September 24, 2008 • 28(39):9018–9028

SYM

48

632

Showing 1 to 1 of 1 entries

Free Get

III. 수행절차

11) T3Q.dl_추론플랫폼_추론모델 관리_satellite_regression

- 추론플랫폼 >> 추론모델관리
- 1. 추론플랫폼 >> 추론모델관리 - 서비스 시작
- 2. 추론플랫폼 >> 추론모델관리 - 추론모델 정보 확인
- 3. 추론플랫폼 >> 추론모델관리 - 추론모델 검증

추론플랫폼 >> 추론모델관리

1. 추론플랫폼 >> 추론모델관리 - 서비스 시작
2. 추론플랫폼 >> 추론모델관리 - 추론모델 정보 확인

운영중 satellite_regression_trainmodel-2022-09-02

배포일 : 2022-09-02 14:23:01

자동업데이트

```

msNtQVEMiRrCyznDg/NWldG1xp4CblZidZ1H61ZUDD*0N8JV5P7rGodbug11C+ESRQCQD1P4VOqewJdSDaz9gDivryiivH4wnHxY1o/wDXD/0RHXOw606xiBrelqOMgYNT/wBpMCyrEF7daqNdSXUy15IGccG
ugk8O29va+aZXcnsQKzW0yJQcknPIVOSErQUByKZa2xknCggZNdTaabKsedyYHGM/WrRis35HfG3nAP/ANarcOkNdsGDqGB4yTWvpsl5aOlvJcuV7bXDBXUWcvBAESzE9ya1Y33ACmTqClYN9AsrHLMbnpUEVU
qKVHTfRqA3yipphiQuCWLSolBqSNDt++apXsRRRtcl8KpToDFzXPXaBp2B5qKGACQY/nXZ6J+5IVcASHWuoBGACBSIQTOFQXPC2H27JJLJO0A+tc1qNpSeolSQY/TFejeG0I0IL5J09elXlnCmrbRuzTZIEygNKY54
rCldTbXEUsYYnv/APWqza3skkux/TsaZeXjQwsQuCbdydkKkLZGyQW+6TU+uplzaQE9Mbe1eZ3VwsEkiMk7V2P6VztzdN5/mioQ/nRaKbq4QE/OTJPaut8xRG3WAHgjByOM81//Z"]
0
2022-09-02 05:31:11.367 [INFO] root: [hunmin log] df.shape : (1, 1)
2022-09-02 05:31:11.367 [INFO] root: [hunmin log] type(df) : <class 'pandas.core.frame.DataFrame'>
2022-09-02 05:31:11.367 [INFO] root: [hunmin log] the end line of the function [transform]
2022-09-02 05:31:11.367 [INFO] root: [hunmin log] the start line of the function [exec_inference]
2022-09-02 05:31:11.378 [INFO] root: [hunmin log] model.summary() : None
2022-09-02 05:31:11.379 [INFO] root: [hunmin log] data transform
2022-09-02 05:31:11.444 [INFO] root: [hunmin log] the end line of the function [inference]
  
```

CPU 2 GPU 0 MEMORY 8GI

전처리 정보 상세보기 학습 정보 상세보기

3. 추론플랫폼 >> 추론모델관리 - 추론모델 검증
- [추론모델테스트]-[요청] 에 아래의 값을 넣고, [테스트] 눌러 수행
- 요청 : 입력 예시 : ['/'data/aip/file_group/pm/pm_334/ds_441/image/1/1230.png']]
- 요청 : '(11_1) Request_satellite_clustering.txt' 파일 안에 있는 base64 형식의
이미지의 내용을 복사 붙여넣기 한다
- 응답 : "{w"inferenceW":[52.50049591064453]}wn"

추론모델 검증

추론모델테스트

요청

```

00ajl8cmXraU/Vo/8AWQ/nqnn02uJgVtFALV/1XpIDBLf/qnyIGNhuUUAUNNU/AAIT3a1/0/WiigA+un+(*tALOJ+U9aKKAH/AG+Kt0V9aD9r/WbJ9aKKAEN1/STKcPp9Z9
aKKAFF2R/B+tL9sP9z9aKKAcm7J/h/Wk+0n+7+tFFAB9pP939aXUfTv60UAH2o/3f1oN0T/AfrRRQAfaJ/AHP1pPtJ/u/rRRQAfaT/AHf1o+0n+7+tFFAB9pP939aPtJ/u/r
RRQAfa9j9akW+Kj/Vj86KKAHHUCf+WY/OKN/kf6sfRRQBf9pP939aPtJ/u/rRRQAfaD/AHf1o+0H+7+tFFAB9oP939aPtB/u/rRRQAfaD/d/Wk+0H+7+tFFAB9oP939aPtB/
u/rRRQAfaD/d/Wi+0H+7+tFFACl6x/B+tH2jTn60UUAJ9pP939aDc5/h/WiigA+0H+7+tH2g/3f1ooaPtJ/u/rS/aj/c/WiigBwvCP4P1pftP/55/rRRQAftB/AM8x+0AvSP8Aln+
tFFADhqJH/LlfnTxqZByYQf8AgVFFACnVSf8AlIP+qBqphWEH/gVFFAEg1oJ/3H/H/ANaiigD/9k="]]
  
```

테스트

응답

```

{"inference": [52.50049591064453]}
  
```

III. 수행절차

12) T3Q.dl_추론플랫폼_추론API관리_satellite_regression

- 추론플랫폼 >> 추론API관리

- 추론플랫폼 >> 추론API관리 - 신규 등록

- 추론플랫폼 >> 추론API관리 - 추론 API 상세

- 추론플랫폼 >> 추론API관리
- 추론플랫폼 >> 추론API관리 - 신규 등록
 - 추론플랫폼 >> 추론API관리 - 추론 API 상세

추론 API 조회

API명 검색 모델명 검색 조회 Clear

번호	사용여부	API명	등록일	등록자	추론모델		
					카테고리	알고리즘	모델명
1	사용중	satellite_regression_api	2022-09-02 14:28:13	kimyj6032	Regression	satellite_regression_train	satellite_regression_trainmodel-2022-09-02

Showing 1 to 1 of 1 entries Prev 1 Next

=> 요청 : {"data": "[테스트 데이터 값 입력="]} => [API 호출] 클릭
=> 응답 : {"data": "[결과 "]}]

2 추론 API 상세 상세보기

테스트

API URL http://dro3yub.dl.nhnes.net/model/api/4ae3b/inference

METHOD POST

요청 {"data": "[테스트 데이터 값 입력="]}]

API 호출 초기화

응답 3 [{"inference": [52.50049591064453]}]

III. 수행절차
13) T3Q.cep_실시간 추론 파이프라인_satellite_regression

- T3Q.cep >> 실시간 추론
 1. 실시간 추론 파이프라인 등록
 - 1) 실시간 추론 파이프라인 구성 정보
 - ① 파이프라인 기본정보
 - Image to API(FileUpload) 선택
 - 파이프라인 이름 : satellite_regression_inference
 - 파이프라인 설명 : satellite_regression_inference
 - ② 기본 설정
 - DBCPConnectionPool Password: postgres
 - ③ 사용자 설정
 - Image_API_FileUpload_API_URL : /model/api/4ae3b/inference
 - Image_API_FileUpload_SourcePath : /AI_HUNMIN/satellite_regression/inference
 - Image_API_FileUpload_Topic : satellite_regression_topic
 - ④ 파이프라인 시작 : [satellite_regression_inference] 우측 상단의 기능 버튼 클릭 후 [시작] 선택
 2. 원본 데이터 업로드
 - 1) T3Q.ai>> Tools>> FileViewer를 이용하여 Image_API_FileUpload_SourcePath 에서 설정한 경로 (/AI_HUNMIN/satellite_regression/inference)에 원본 파일 업로드
 - 2) 데이터 적재 확인
 - pgadmin 도구 이용 inference_result 테이블 조회

(2) 데이터 적재 확인

T3Q.ai >> Tools>> PgAdmin

#inference_origin

inference_result

테이블 조회

#select * from inference_origin;

select * from inference_result;

데이터 저장 확인

```
=====
SELECT * FROM public.inference_result
where url like '%/model/api/4ae3b/inference%'
order by start_time desc
```