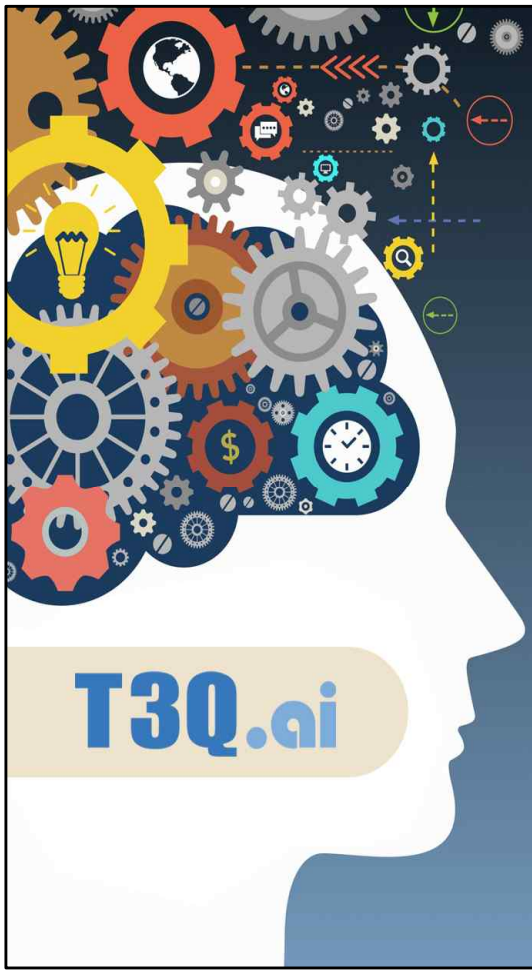


AI훈민정음_ Text_Generation

(RNN을 사용한 텍스트 생성 (AI 작가))

Copyright © 2022 T3Q Co., Ltd. All Rights Reserved.

목 차

I. 개요
1. 소개

II. 프로그래밍 가이드 문서
0_local_ text_generation_requirement
0_local_text_generation.ipynb
1_local_platform_text_generation.ipynb
2_platform_process

III. 수행 절차
01) T3Q.cep_데이터수집 파이프라인_text_generation
02) T3Q.cep_데이터변환 파이프라인_text_generation
03) T3Q.dl_프로젝트 설정_실행환경 관리_text_generation
04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_text_generation
05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_text_generation
06) T3Q.dl_학습플랫폼_데이터셋 관리_text_generation
07) T3Q.dl_학습플랫폼_전처리 모델 설계_text_generation
08) T3Q.dl_학습플랫폼_전처리 모델 관리_text_generation
09) T3Q.dl_학습플랫폼_학습모델 설계_text_generation
10) T3Q.dl_학습플랫폼_학습모델 관리_text_generation
11) T3Q.dl_추론플랫폼_추론모델 관리_text_generation
12) T3Q.dl_추론플랫폼_추론API관리_text_generation
13) T3Q.cep_실시간 추론 파이프라인_text_generation

1. 소개 : 텍스트 생성

세익스피어의 저작 텍스트 데이터셋을 사용하여 대사를 생성하는 예제

1. 데이터셋

shakespeare.txt :
세익스피어의 저작 텍스트 파일을
데이터셋으로 사용

저작 텍스트 파일 :
로미오와 줄리엣, 리처드 2세, 겨울 이야기, 헨리 6세 등

2. 전처리 및 학습

전처리:
워드 임베딩으로 텍스트 벡터화 (Vectorization)
슬라이딩 윈도우

학습:
GRU을 사용한 텍스트 생성

3. 추론 결과

입력 텍스트 이후의 텍스트 생성하여 이어지는 텍스트를 추론

4. 기대 효과

대사를 예측하여 대본 만들 수 있음
작가의 특성에 따른 시나리오 생성

데이터셋

- shakespeare.txt
- 로미오와 줄리엣, 리처드 2세, 겨울 이야기, 헨리 6세 등 세익스피어 저작 대사

ROMEO:
Not having that, which, having, makes them short.

BENVOLIO:
In love?

ROMEO:
Out--

BENVOLIO:
Of love?

로미오와 줄리엣

KING RICHARD II:
Should dying men flatter with those that live?

JOHN OF GAUNT:
No, no, men living flatter those that die.

KING RICHARD II:
Thou, now a-dying, say'st thou flatterest me.

리처드 2세

GRUMIO:
O this woodcock, what an ass it is!

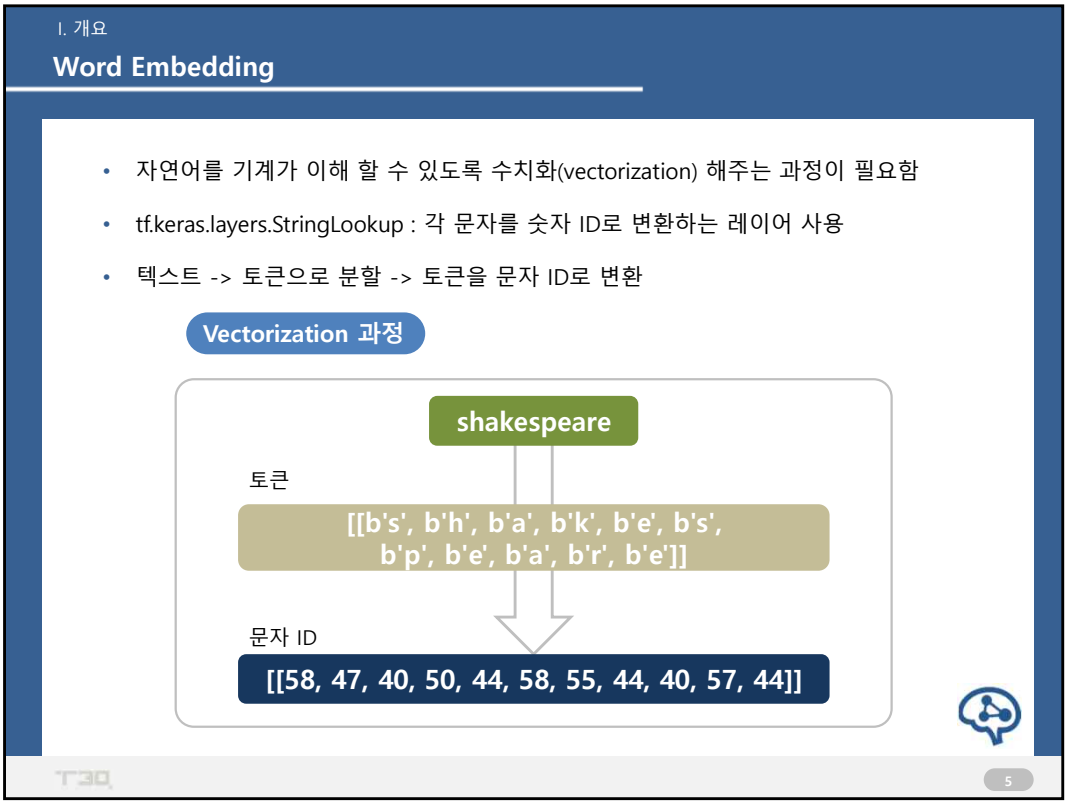
PETRUCHIO:
Peace, sirrah!

HORTENSIO:
Grumio, mum! God save you, Signior Gremio.

겨울 이야기

QUEEN ELIZABETH:
Come, come, we know your meaning, brother Gloucester;
You envy my advancement and my friends':
God grant we never may have need of you!

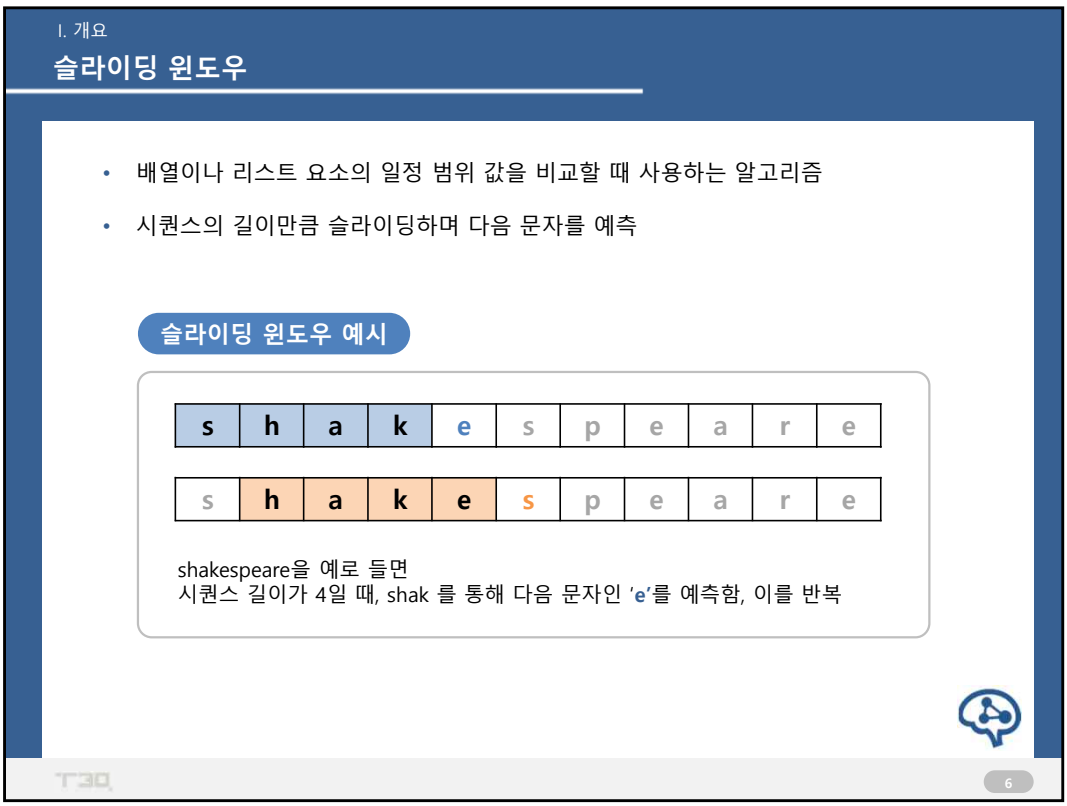
헨리 6세



- 이미지, 설명 참조 : https://www.tensorflow.org/text/guide/word_embeddings#encode_each_word_with_a_unique_number

- Word Embedding
  자연어의 특징을 뽑아내서 수치화 하는 과정

- 유사한 단어가, 유사한 인코딩을 갖는 효율적이고 조밀한 표현을 사용하는 방법을 제공



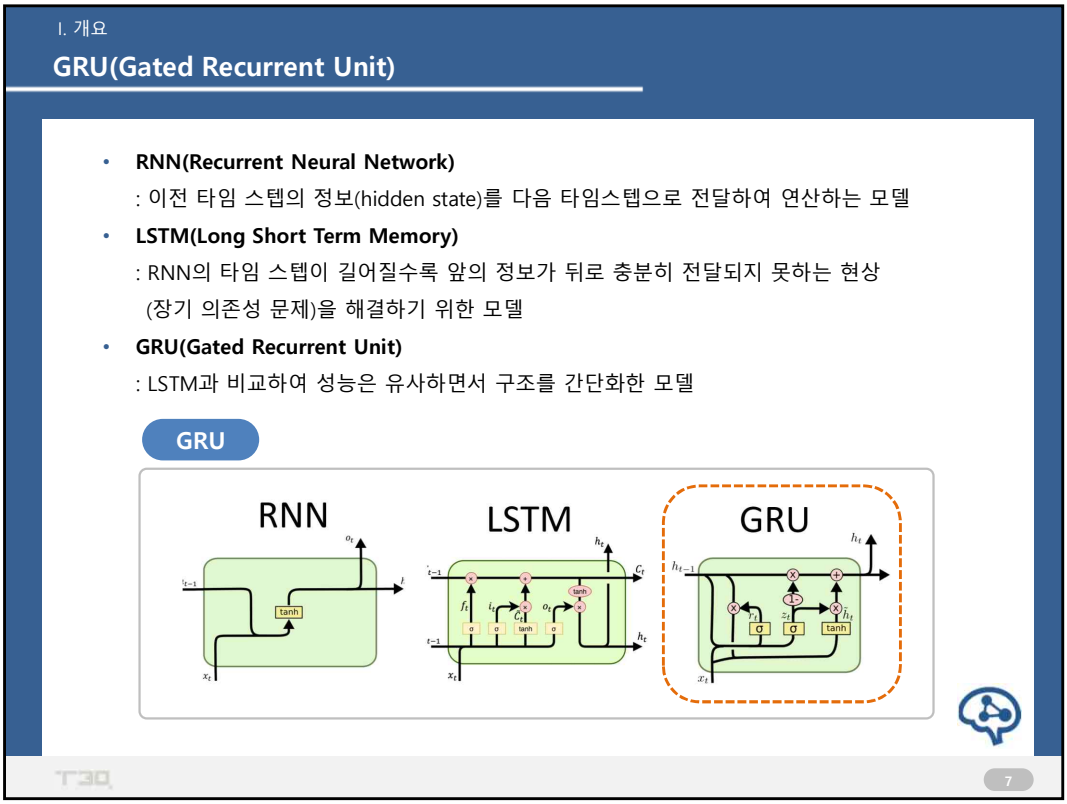
- 슬라이딩 윈도우 :

각 입력 시퀀스에 대해 해당 대상은 한 문자를 오른쪽으로 이동한 것을 제외하고 동일한 길이의 텍스트를 포함

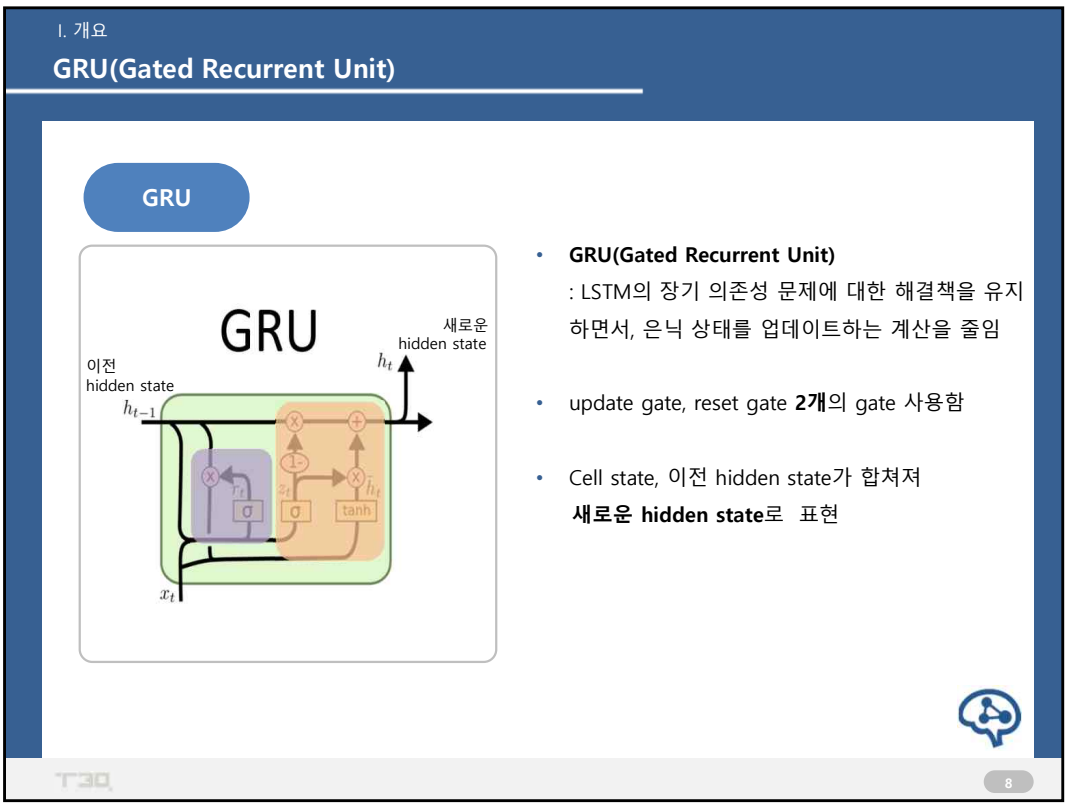
* 시퀀스란 연관된 연속의 데이터 (연속적인 시간 간격으로 배치된 데이터)

예시 : seq_length(시퀀스 길이)는 4이고 텍스트는 "shakespeare"인 경우
input 시퀀스는 "shak"이고, target 시퀀스는 "hake"

* 본 예제에서는 seq_length(시퀀스 길이)를 100으로 설정



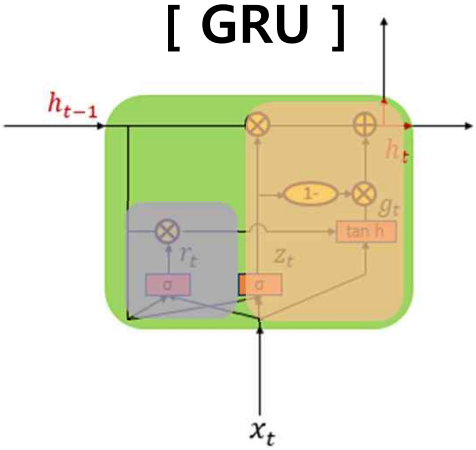
이미지, 설명 참조: RNN, https://wikidocs.net/22886 / LSTM, https://wikidocs.net/22888 / GRU, https://wikidocs.net/22889



이미지, 설명 참조: RNN, https://wikidocs.net/22886 / LSTM, https://wikidocs.net/22888 / GRU, https://wikidocs.net/22889

- Cell state : 장기간 메모리 역할 수행
- Gate : 연결의 강도를 조절

[GRU]

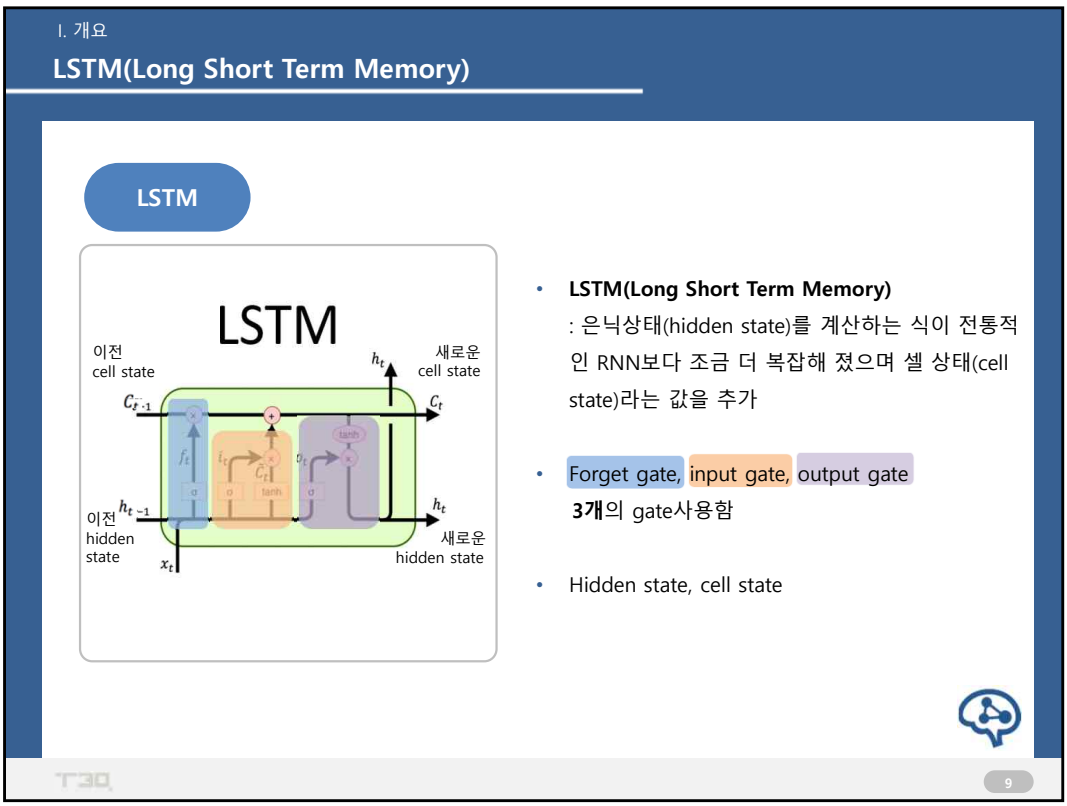


$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \quad \text{--- (1)}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \quad \text{--- (2)}$$
$$g_t = tanh(W_{hg}(r_t \circ h_{t-1}) + W_{xg}x_t + b_g) \quad \text{--- (3)}$$
$$h_t = (1 - z_t) \circ g_t + z_t \circ h_{t-1} \quad \text{--- (4)}$$

- Reset Gate ← (1) 식
: **과거의 정보를 얼마나 반영할지 결정,** 이전 hidden state와 현 시점의 x를 sigmoid 활성화 함수를 적용하여 구하는 식

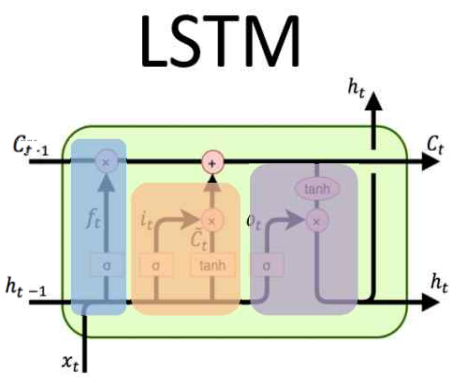
- Update Gate ← (2) ~ (4) 식
: **과거와 현재의 정보를 각각 얼마나 반영**할지에 대한 비율을 구하는 식

- z_t : 이전 정보 비율 결정 Forget gate역할
- 1-z_t : z_t에 대응되는 현재 정보의 비율을 결정 Input gate역할



이미지, 설명 참조: RNN, https://wikidocs.net/22886 / LSTM, https://wikidocs.net/22888 / GRU, https://wikidocs.net/22889

- Cell state : 장기간 메모리 역할 수행
- Gate : 연결의 강도를 조절



$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad \text{--- (1)}$$
$$g_t = tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad \text{--- (2)}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad \text{--- (3)}$$
$$h_t = o_t \circ tanh(c_t)$$

- Input Gate ← (1) 식
: sigmoid 함수를 지나 0과 1사이 값을 가지는 값과 thanh 함수를 지나 -1과 1사이의 값을 가지는 값이 두 개의 값으로 기억할 정보의 양을 (2)식에서 결정하게 됨

- Forget Gate ← (2) 식
: 현 시점 t의 x값과 이전 시점 t-1의 은닉상태가 sigmoid 함수를 지나면서 0에 가까우면 삭제, 1에 가까우면 기억

- Output Gate ← (3) 식
: 현 시점 t의 x값과 이전 시점 t-1의 은닉 상태가 sigmoid 함수를 지난 값으로 현재 시점 t의 은닉상태를 결정하고 출력층으로 향함

0_local_text_generation_requirement

- 로컬에 설치 되어야 할 패키지 버전
  - ✓ matplotlib 3.4.3
  - ✓ tensorflow 2.8.0
  - ✓ tensorflow-gpu 2.6.0

II. 프로그래밍 가이드 문서

0_local_text_generation.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_text_generation.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

11

0_local_text_generation.ipynb
==========

```
# imports
import tensorflow as tf
import numpy as np
import os
import zipfile
from glob import glob
import matplotlib.pyplot as plt
```

1. 데이터셋 준비(Data Setup)

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

# dataset.zip 파일을 dataset 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()

path_to_file = os.path.join(zip_target_path, 'dataset/shakespeare.txt')

# 데이터 불러오기
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
```

2. 데이터 전처리 (Data Preprocessing)

```
# 어휘묶음 생성
vocab = sorted(set(text))

# chars를 숫자id로 변경하는 StringLookup layer
ids_from_chars = tf.keras.layers.StringLookup(
    vocabulary=list(vocab), mask_token=None)

# 숫자id를 chars로 변경하는 StringLookup layer
chars_from_ids = tf.keras.layers.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(), invert=True, mask_token=None)


# train예제 및 target 만들기
# 각 입력 시퀀스에 대해 해당 대상은 한 문자를 오른쪽으로 이동한 것을 제외하고 동일한 길이의 텍스트를 포함한다.
# 예시 : seq_length는 4이고 텍스트는 "Hello"인 경우
#        입력 시퀀스는 "Hell"이고, target 시퀀스는 "ello"

all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))

ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)

# 여기서 시퀀스 길이는 100
seq_length = 100
# examples_per_epoch = len(text)//(seq_length+1)

sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)

# 시퀀스를 입력으로 받아 복제하고 입력과 라벨을 정렬하는 함수
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

# 입력시퀀스에 따른 타겟시퀀스 생성하고 이를 dataset에 저장
dataset = sequences.map(split_input_target)


# Batch size
BATCH_SIZE = 64

BUFFER_SIZE = 10000

dataset = (
    dataset
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE, drop_remainder=True)
    .prefetch(tf.data.experimental.AUTOTUNE))
```

3. 학습 모델 훈련(Train Model)

```
# Length of the vocabulary in chars
vocab_size = len(vocab)

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = 1024


class MyModel(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, rnn_units):
        super().__init__(self)
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units,
                                       return_sequences=True,
                                       return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, return_state=False, training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states, training=training)
        x = self.dense(x, training=training)

        if return_state:
            return x, states
        else:
            return x


model = MyModel(
    # Be sure the vocabulary size matches the `StringLookup` layers.
    vocab_size=len(ids_from_chars.get_vocabulary()),
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)

loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
```

모델 컴파일 및 학습
모델 컴파일(Compile Model)

```
model.compile(optimizer='adam', loss=loss)

# Directory where the checkpoints will be saved
checkpoint_dir = './meta_data/training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "last_ckpt")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    monitor='loss',
    save_best_only=True,
    save_weights_only=True)
```

모델 학습(Train Model)

```
EPOCHS = 50

history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

모델 평가(Evaluate Model)

```
#Plot accuracy and loss curves for both training and validation data
loss = history.history['loss']

plt.plot(loss, label='Loss')
plt.title("Loss")
plt.legend()
plt.show()
```

4. 추론 (Inference)

```
class OneStep(tf.keras.Model):
    def __init__(self, model, chars_from_ids, ids_from_chars, temperature=1.0):
        super().__init__()
        self.temperature = temperature
        self.model = model
        self.chars_from_ids = chars_from_ids
        self.ids_from_chars = ids_from_chars

        # Create a mask to prevent "[UNK]" from being generated.
        skip_ids = self.ids_from_chars(['[UNK]'])[:, None]
        sparse_mask = tf.SparseTensor(
            # Put a -inf at each bad index.
            values=[-float('inf')]*len(skip_ids),
            indices=skip_ids,
            # Match the shape to the vocabulary
            dense_shape=[len(ids_from_chars.get_vocabulary())])
        self.prediction_mask = tf.sparse.to_dense(sparse_mask)
```

```
    @tf.function
    def generate_one_step(self, inputs, states=None):
        # Convert strings to token IDs.
        input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
        input_ids = self.ids_from_chars(input_chars).to_tensor()

        # Run the model.
        # predicted_logits.shape is [batch, char, next_char_logits]
        predicted_logits, states = self.model(inputs=input_ids, states=states,
                                              return_state=True)
        # Only use the last prediction.
        predicted_logits = predicted_logits[:, -1, :]
        predicted_logits = predicted_logits/self.temperature
        # Apply the prediction mask: prevent "[UNK]" from being generated.
        predicted_logits = predicted_logits + self.prediction_mask

        # Sample the output logits to generate token IDs.
        predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
        predicted_ids = tf.squeeze(predicted_ids, axis=-1)

        # Convert from token ids to characters
        predicted_chars = self.chars_from_ids(predicted_ids)

        # Return the characters and model state.
        return predicted_chars, states


chars_from_ids =
tf.keras.layers.experimental.preprocessing.StringLookup(vocabulary=ids_from_chars.get_vocabulary(),
invert=True, mask_token=None)

one_step_model = OneStep(model, chars_from_ids, ids_from_chars)

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

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

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

extract_zip_file.close()

# laod test data
test_files = glob(os.path.join(zip_test_target_path, 'test_dataset/*.txt'))


test_data = []
for test_file in test_files:
    with open(test_file, 'r') as f:
        test_data.append(f.read())
```

```
def inference(next_char):
    states = None

    result = [next_char]

    for n in range(100):
        next_char, states = one_step_model.generate_one_step(next_char, states=states)
        result.append(next_char)

    return tf.strings.join(result)[0].numpy().decode("utf-8")

#결과 확인


for data in test_data:
    print('############ inference ############')
    next_char = tf.constant([data])
    print(inference(next_char), '₩n₩n')
```

1_local_platform_text_generation.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 1(파일명): 1_local_platform_text_generation.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.

II. 프로그래밍 가이드 문서

1_local_platform_text_generation.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.

18

```
# 파일명: text_generation_preprocess.py

from text_generation_preprocess_sub import exec_process

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):
    return {}


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
```

```
# 파일명: text_generation_preprocess_sub.py

import os
import numpy as np
import pandas as pd
import zipfile
import logging


def exec_process(pm):

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

    logging.info('[hunmin log] pm.source_path : {}'.format(pm.source_path))

    # 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]')


# 저장 파일 확인
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))
```

```
# 파일명: text_generation_train.py

from text_generation_train_sub import exec_train, exec_init_svc, exec_inference

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 }
```

```
# 파일명: text_generation_train_sub.py

# Imports
import tensorflow as tf
import numpy as np
import os
import pickle
import logging

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

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

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. 데이터 세트 준비(Data Setup)
    ##################################################################

    logging.info('[hunmin log] data load')

    path_to_file = os.path.join(tm.train_data_path, 'dataset/shakespeare.txt')
    logging.info('[hunmin log] file path : {}'.format(path_to_file))

    text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
    logging.info('[hunmin log] loaded data check (text[:100]) : {}'.format(text[:100]))
```

```
    ################################################################
    ## 2. 데이터 전처리(Data Preprocessing)
    ################################################################

    vocab = sorted(set(text))

    # 추론에 사용할 vocab데이터 저장
    with open(os.path.join(tm.model_path, 'vocabulary.p'), 'wb') as f:
        pickle.dump(vocab, f)

    # 문자를 id로 변환
    ids_from_chars =
tf.keras.layers.experimental.preprocessing.StringLookup(vocabulary=list(vocab),
mask_token=None)
    all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
    ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)

    seq_length = 100
    sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)
    dataset = sequences.map(split_input_target)

    # Batch size
    BATCH_SIZE = 64
    BUFFER_SIZE = 10000
```

```
dataset = (dataset
        .shuffle(BUFFER_SIZE)
        .batch(BATCH_SIZE, drop_remainder=True)
        .prefetch(tf.data.experimental.AUTOTUNE))

#################################################################
## 3. 학습 모델 훈련(Train Model)
#################################################################

# 모델 구축 (Build Model)
# The embedding dimension
embedding_dim = 256
# Number of RNN units
rnn_units = 1024

model = MyModel(
        # Be sure the vocabulary size matches the `StringLookup` layers.
        vocab_size=len(ids_from_chars.get_vocabulary()),
        embedding_dim=embedding_dim,
        rnn_units=rnn_units)

# 입력 텍스트 다음에 올 문자 중 확률이 가장 높은 문자를 추출해야 하므로
# 다중분류에 사용되는 loss를 사용한다.
loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam', loss=loss)

# 모델 학습
# Directory where the checkpoints will be saved
checkpoint_dir = os.path.join(tm.model_path, 'training_checkpoints')
```

```
  # 체크포인트 콜백
    checkpoint_prefix = os.path.join(checkpoint_dir, "last_ckpt")
    checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
                                    filepath=checkpoint_prefix,
                                    save_weights_only=True)

    EPOCHS = 50
    history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

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

    ##################################################################
    ## 플랫폼 시각화
    ##################################################################
    plot_metrics(tm, history)

    # 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):

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

```
    # 저장 파일 확인
    list_files_directories(im.model_path)

    ##################################################################
    ## 학습 모델 준비
    ##################################################################

    with open(os.path.join(im.model_path, 'vocabulary.p'), 'rb') as f:
        vocab = pickle.load(f)

    # rebuild model
    ids_from_chars =
tf.keras.layers.experimental.preprocessing.StringLookup(vocabulary=list(vocab),
mask_token=None)
    # The embedding dimension
    embedding_dim = 256
    # Number of RNN units
    rnn_units = 1024

    loaded_model = MyModel(
            # Be sure the vocabulary size matches the `StringLookup` layers.
            vocab_size=len(ids_from_chars.get_vocabulary()),
            embedding_dim=embedding_dim,
            rnn_units=rnn_units)
    loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
    loaded_model.compile(optimizer='adam', loss=loss)
```

```
  # 가장 최근 체크포인트를 호출
    latest = tf.train.latest_checkpoint(os.path.join(im.model_path, 'training_checkpoints'))
    loaded_model.load_weights(latest)

    chars_from_ids =
tf.keras.layers.experimental.preprocessing.StringLookup(vocabulary=ids_from_chars.get_voca
bulary(), invert=True, mask_token=None)

    # rebuild한 모델을 이용하여 입력 텍스트에 이어지는 텍스트를 예측하는 모델을 반환한다.
    loaded_one_step_model = OneStep(loaded_model, chars_from_ids, ids_from_chars)

    return {'model' : loaded_one_step_model}



def exec_inference(df, params, batch_id):

    ##################################################################
    ## 4. 추론(Inference)
    ##################################################################

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

    ## 학습 모델 준비
    model = params['model']
```

```
    origin_data = df.iloc[0, 0]
    input_data = tf.constant([origin_data])

    logging.info('[hunmin log] data predict')
    # data predict
    # 상태 초기값 : None
    states = None
    prediction = [input_data]
    # 입력 이후 100자 예측
    for n in range(100):
        input_data, states = model.generate_one_step(input_data, states=states)
        prediction.append(input_data)

    inference = tf.strings.join(prediction)[0].numpy().decode("utf-8")
    logging.info('[hunmin log] inference : {}'.format(inference))

    # inverse transform
    result = {'inference' : inference}
    logging.info('[hunmin log] result : {}'.format(result))

    return result


# 저장 파일 확인
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))
```

```
#####################################################################
## exec_train(tm) 호출 함수
#####################################################################

def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

# 모델 객체 정의
class MyModel(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, rnn_units):
        super().__init__(self)
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units,
                                       return_sequences=True,
                                       return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, return_state=False, training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states, training=training)
        x = self.dense(x, training=training)
```

```
        if return_state:
            return x, states
        else:
            return x

class OneStep(tf.keras.Model):
    def __init__(self, model, chars_from_ids, ids_from_chars, temperature=1.0):
        super().__init__()
        self.temperature = temperature
        self.model = model
        self.chars_from_ids = chars_from_ids
        self.ids_from_chars = ids_from_chars

        # Create a mask to prevent "[UNK]" from being generated.
        skip_ids = self.ids_from_chars(['[UNK]'])[:, None]
        sparse_mask = tf.SparseTensor(
            # Put a -inf at each bad index.
            values=[-float('inf')]*len(skip_ids),
            indices=skip_ids,
            # Match the shape to the vocabulary
            dense_shape=[len(ids_from_chars.get_vocabulary())])
        self.prediction_mask = tf.sparse.to_dense(sparse_mask)
```

```
@tf.function
    def generate_one_step(self, inputs, states=None):
        # Convert strings to token IDs.
        input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
        input_ids = self.ids_from_chars(input_chars).to_tensor()

        # Run the model.
        # predicted_logits.shape is [batch, char, next_char_logits]
        predicted_logits, states = self.model(inputs=input_ids, states=states,
                                              return_state=True)
        # Only use the last prediction.
        predicted_logits = predicted_logits[:, -1, :]
        predicted_logits = predicted_logits/self.temperature
        # Apply the prediction mask: prevent "[UNK]" from being generated.
        predicted_logits = predicted_logits + self.prediction_mask

        # Sample the output logits to generate token IDs.
        predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
        predicted_ids = tf.squeeze(predicted_ids, axis=-1)

        # Convert from token ids to characters
        predicted_chars = self.chars_from_ids(predicted_ids)

        # Return the characters and model state.
        return predicted_chars, states
```

```
# 시각화
def plot_metrics(tm, history):

    # accuracy_list = history.history['accuracy']
    loss_list = history.history['loss']

    for step, loss in enumerate(loss_list):
        metric={}
        metric['accuracy'] = 0
        metric['loss'] = loss
        metric['step'] = step
        tm.save_stat_metrics(metric)

    logging.info('[hunmin log] accuracy and loss curve plot for platform')
```

```
@tf.function
  def generate_one_step(self, inputs, states=None):
    # Convert strings to token IDs.
    input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
    input_ids = self.ids_from_chars(input_chars).to_tensor()

    # Run the model.
    # predicted_logits.shape is [batch, char, next_char_logits]
    predicted_logits, states = self.model(inputs=input_ids, states=states,
                                          return_state=True)
    # Only use the last prediction.
    predicted_logits = predicted_logits[:, -1, :]
    predicted_logits = predicted_logits/self.temperature
    # Apply the prediction mask: prevent "[UNK]" from being generated.
    predicted_logits = predicted_logits + self.prediction_mask

    # Sample the output logits to generate token IDs.
    predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)

    # Convert from token ids to characters
    predicted_chars = self.chars_from_ids(predicted_ids)

    # Return the characters and model state.
    return predicted_chars, states
```

```
# 시각화
def plot_metrics(tm, history):

    # accuracy_list = history.history['accuracy']
    loss_list = history.history['loss']

    for step, loss in enumerate(loss_list):
        metric={}
        metric['accuracy'] = 0
        metric['loss'] = loss
        metric['step'] = step
        tm.save_stat_metrics(metric)

    logging.info('[hunmin log] accuracy and 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:
    param_info = {}
    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)

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

import io
import pandas as pd

# base64 encoded image
data = [['ROMEO : ']]
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)
```

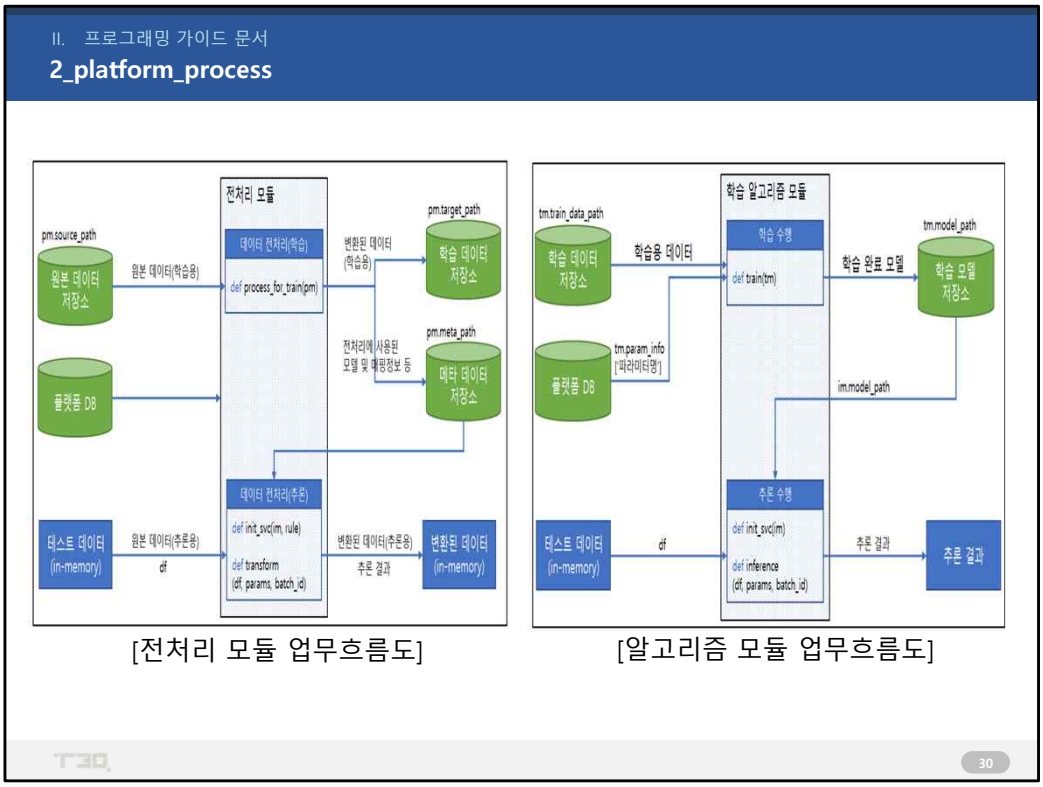
2_platform_process

- 파일명 : text_generation_preprocess.py
  - ✓ from text_generation_preprocess_sub import exec_process
  - ✓ def process_for_train(pm):
  - ✓ def init_svc(im, rule):
  - ✓ def transform(df, params, batch_id):

- 파일명: text_generation_preprocess_sub.py
  - ✓ def exec_process(pm):

- 파일명: text_generation_train.py
  - ✓ from text_generation_train_sub import exec_train, exec_init_svc, exec_inference
  - ✓ def train(tm):
  - ✓ def init_svc(im):
  - ✓ def inference(df, params, batch_id):

- 파일명: text_generation_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)



[전처리 모듈 업무흐름도] [알고리즘 모듈 업무흐름도]

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 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
    """

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
```

2) 프로젝트 설정/ 학습 알고리즘 관리 함수(AI 훈민정음 프로젝트)

```
import logging

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 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
    """

    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}
```

수행절차 소개

01) T3Q.cep_데이터수집 파이프라인_text_generation : 해당 예제에서는 수행 절차 없음

02) T3Q.cep_데이터변환 파이프라인_text_generation : 해당 예제에서는 수행 절차 없음

03) T3Q.dl_프로젝트 설정_실행환경 관리_text_generation

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_text_generation

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_text_generation

06) T3Q.dl_학습플랫폼_데이터셋 관리_text_generation

07) T3Q.dl_학습플랫폼_전처리 모델 설계_text_generation

08) T3Q.dl_학습플랫폼_전처리 모델 관리_text_generation

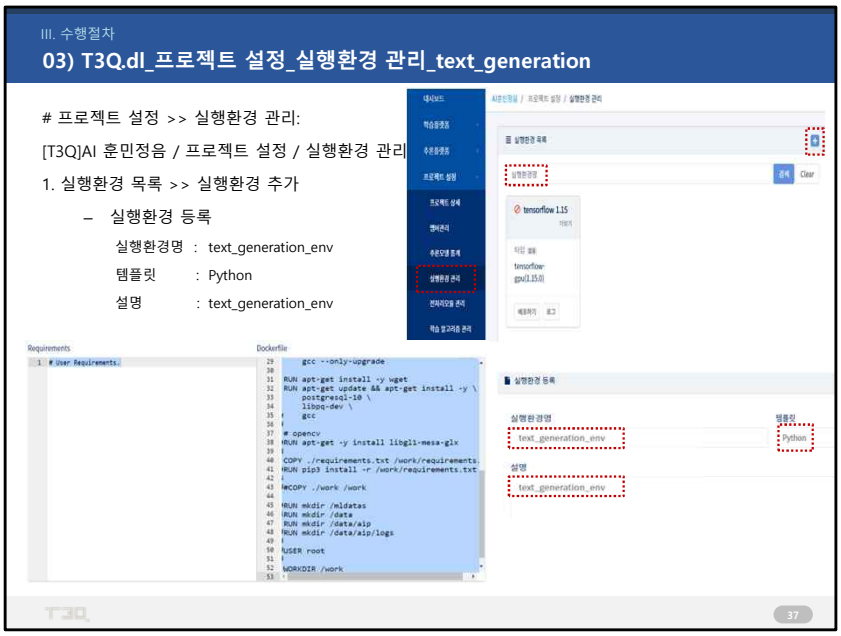
09) T3Q.dl_학습플랫폼_학습모델 설계_text_generation

10) T3Q.dl_학습플랫폼_학습모델 관리_text_generation

11) T3Q.dl_추론플랫폼_추론모델 관리_text_generation

12) T3Q.dl_추론플랫폼_추론API관리_text_generation

13) T3Q.cep_실시간 추론 파이프라인_text_generation



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
```

```
==================================================
2) Dockerfile-계속
==================================================
# 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 psycopg2 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

III. 수행절차

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_text_generation

- 프로젝트 설정 >>전처리모듈관리
  - 전처리모듈 관리>> [전처리 모듈 추가] 실행
  - 전처리모듈 등록 시 순서와 입력 정보

① 기본정보
- 전처리명: text_generation_premodule
- 실행환경 선택 : text_generation_env
- GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default

④ 파라미터

⑤ 소스 코드 : T3Q.ai_platform_text_generation_preprocess.zip [파일업로드]
- LICENSE.txt
- README.txt
- text_generation_preprocess.py (실행모듈 선택)
- text_generation_preprocess_sub.py
- platform_process.txt

39

- 전처리모듈 등록 시 순서와 입력 정보

① 기본정보

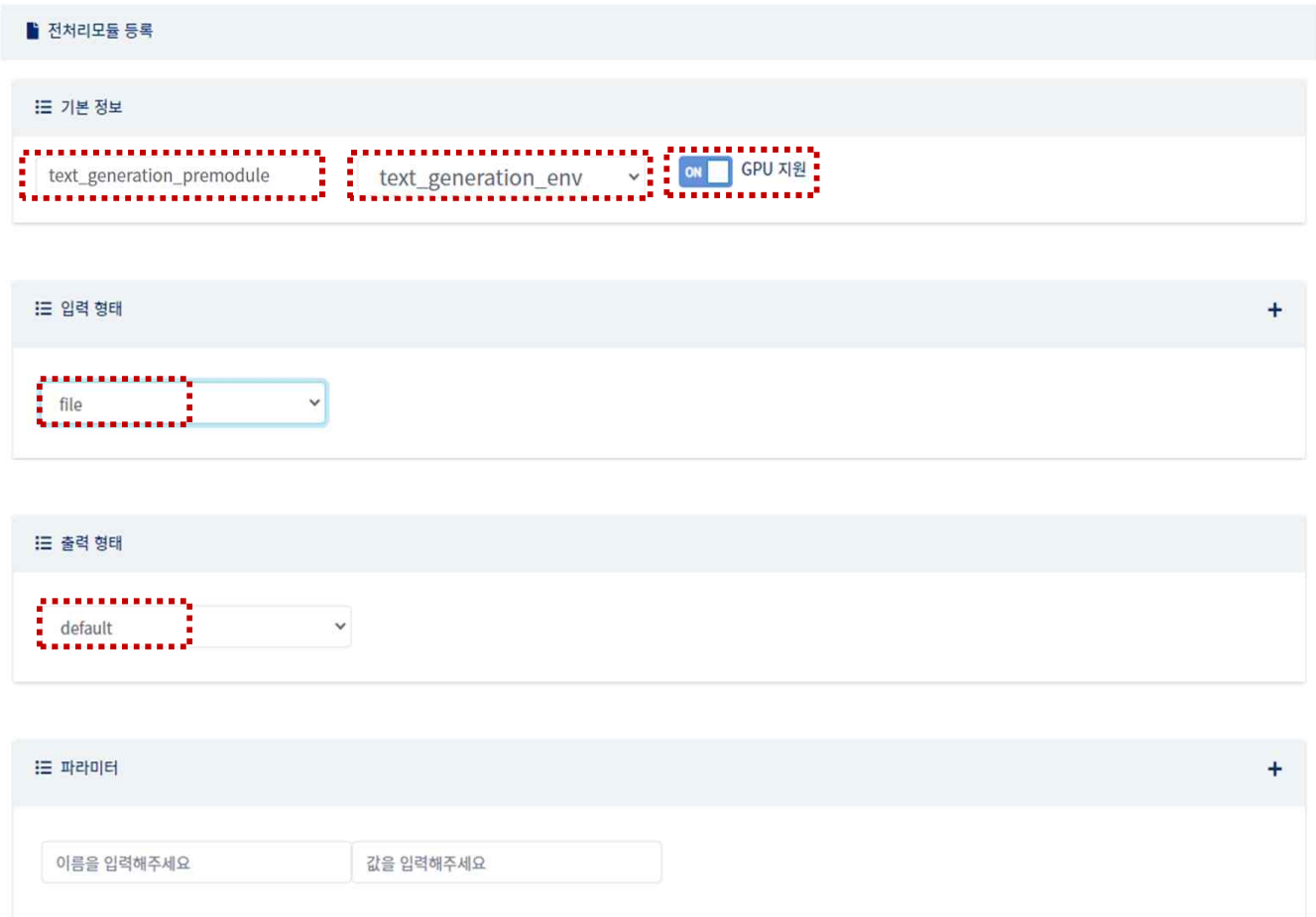
전처리명: text_generation_premodule

실행환경 선택 : text_generation_env

GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default



III. 수행절차

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_text_generation

- 프로젝트 설정 >>전처리모듈관리
  - 전처리모듈 관리>> [전처리 모듈 추가] 실행
  - 전처리모듈 등록 시 순서와 입력 정보

① 기본정보
- 전처리명: text_generation_premodule
- 실행환경 선택 : text_generation_env
- GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default

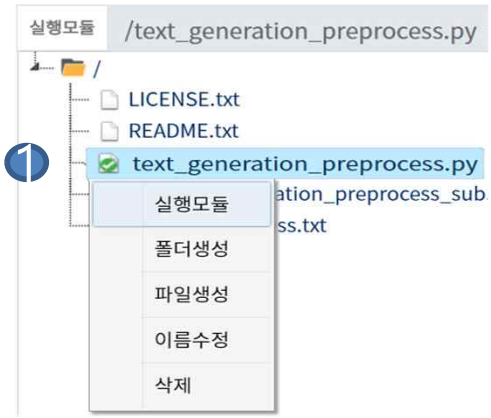
④ 파라미터

⑤ 소스 코드 : T3Q.ai_platform_text_generation_preprocess.zip [파일업로드]
- LICENSE.txt
- README.txt
- text_generation_preprocess.py (실행모듈 선택)
- text_generation_preprocess_sub.py
- platform_process.txt

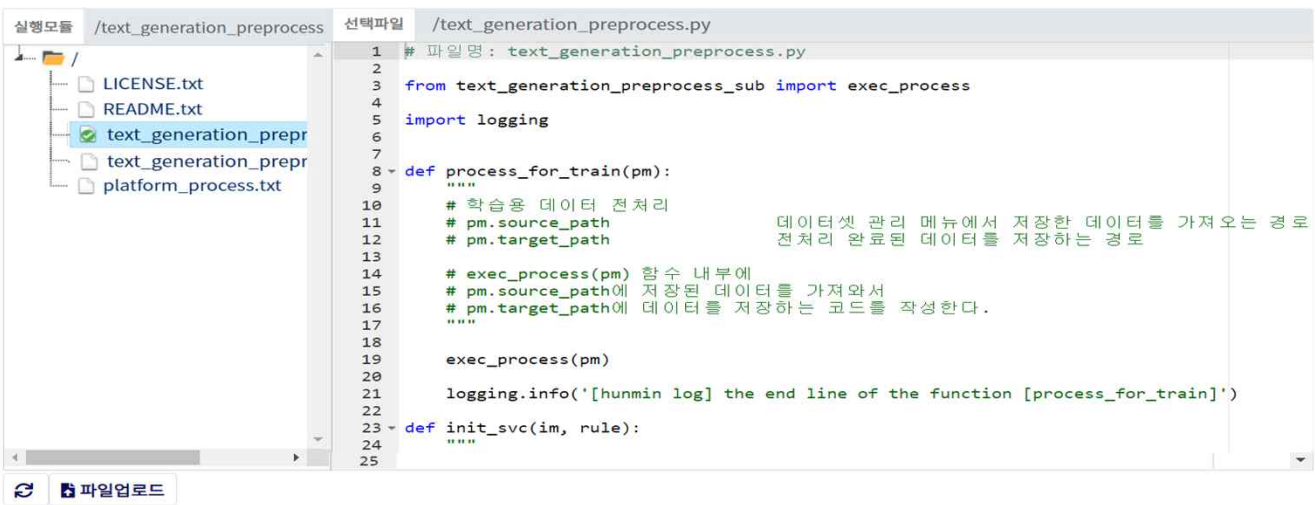
40

전처리모듈 등록 시 순서와 입력 정보

⑤ 소스 코드 : T3Q.ai_platform_text_generation_preprocess.zip [파일업로드] 누름
- LICENSE.txt
- README.txt
- text_generation_preprocess.py (실행모듈 선택)
- text_generation_preprocess_sub.py
- platform_process.txt



2 소스 코드




```
# 파일명 : text_generation_preprocess.py

from text_generation_preprocess_sub import exec_process

import logging


def process_for_train(pm):
    """
    # 학습용 데이터 전처리
    # pm.source_path                데이터셋 관리 메뉴에서 저장한 데이터를 가져오는 경로
    # pm.target_path                전처리 완료된 데이터를 저장하는 경로

    # exec_process(pm) 함수 내부에
    # pm.source_path에 저장된 데이터를 가져와서
    # pm.target_path에 데이터를 저장하는 코드를 작성한다.
    """

    exec_process(pm)

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

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

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_text_generation

- 프로젝트 설정 >> 학습 알고리즘 관리
  - 학습 알고리즘 관리>>
    - [알고리즘 추가] 실행

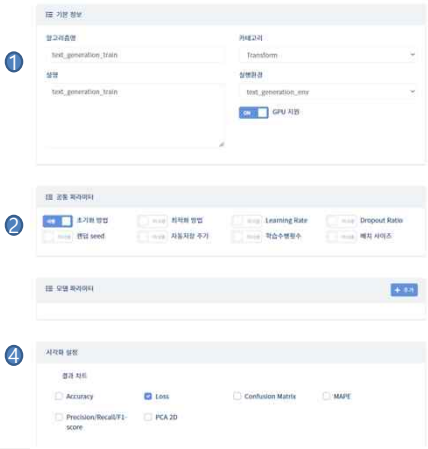
- 학습 알고리즘 등록 시 순서와 입력 정보

① 기본정보: 다음과 같이 입력
-알고리즘명: text_generation_train
-설명 : text_generation_train
-카테고리 : Transform
-실행환경 : text_generation_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



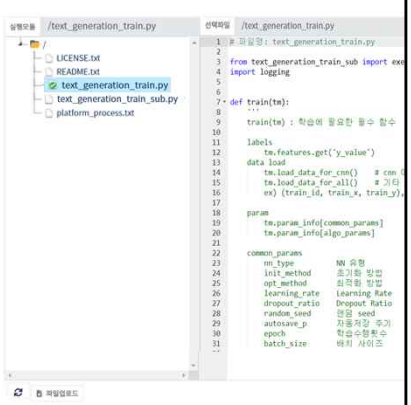
05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_text_generation

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

⑤ 소스코드 업로드 : [파일업로드]

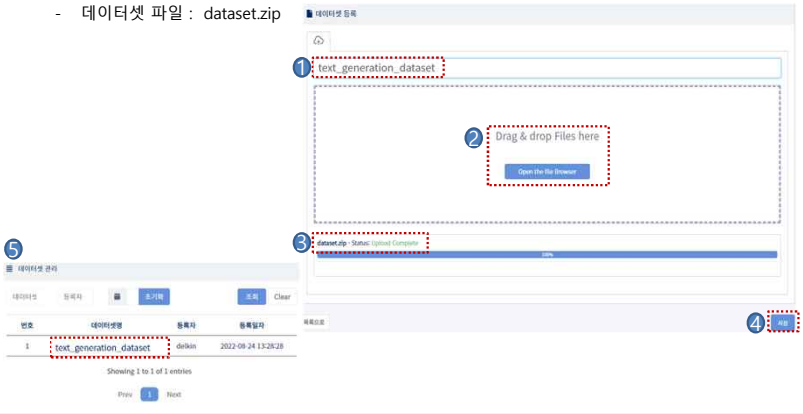
: T3Q.ai_platform_text_generation_train.zip
- LICENSE.txt
- README.txt
- text_generation_train.py
- text_generation_train_sub.py
- platform_process.txt

: 실행 모듈 설정 >> [저장]
- /text_generation_train.py 실행 모듈 지정
- [저장]을 누른다.



06) T3Q.dl_학습플랫폼_데이터셋 관리_text_generation

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



07) T3Q.dl_학습플랫폼_전처리 모델 설계_text_generation

- 학습플랫폼 >> 전처리 모델 설계
- 전처리 모델 설계 절차

① 학습플랫폼 >> 데이터셋 관리
  - 데이터셋 명 : text_generation_dataset 선택

② text_generation_dataset 아래의 [전처리 모델 설계] 선택

07) T3Q.dl_학습플랫폼_전처리 모델 설계_text_generation

- 학습플랫폼 >> 전처리 모델 설계

❷ 전처리 모델 설계 절차

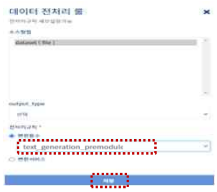
Step1. 기본 정보
- 모델명: text_generation_premodel

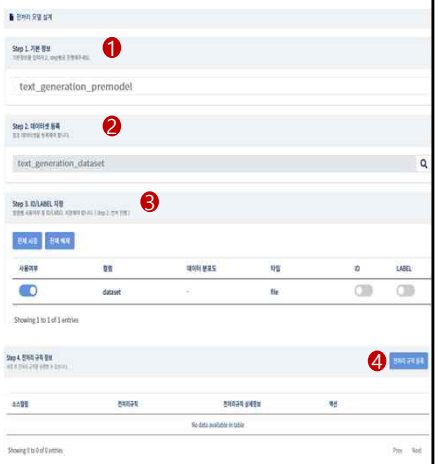
Step2. 데이터셋 등록:
- 참조데이터셋 : text_generation_dataset

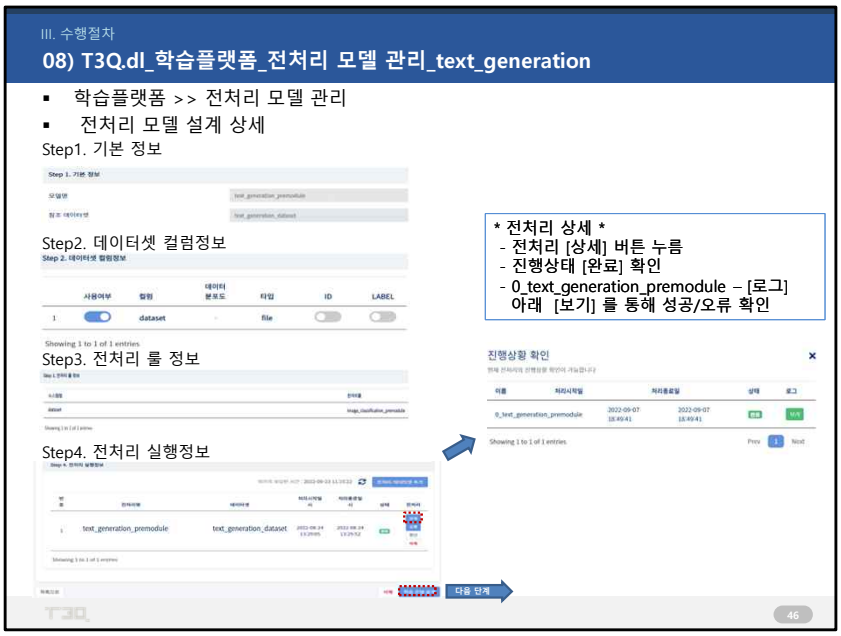
Step3. ID/LABEL 지정

Step4. 전처리 규칙 정보
- [전처리 규칙 등록] 선택

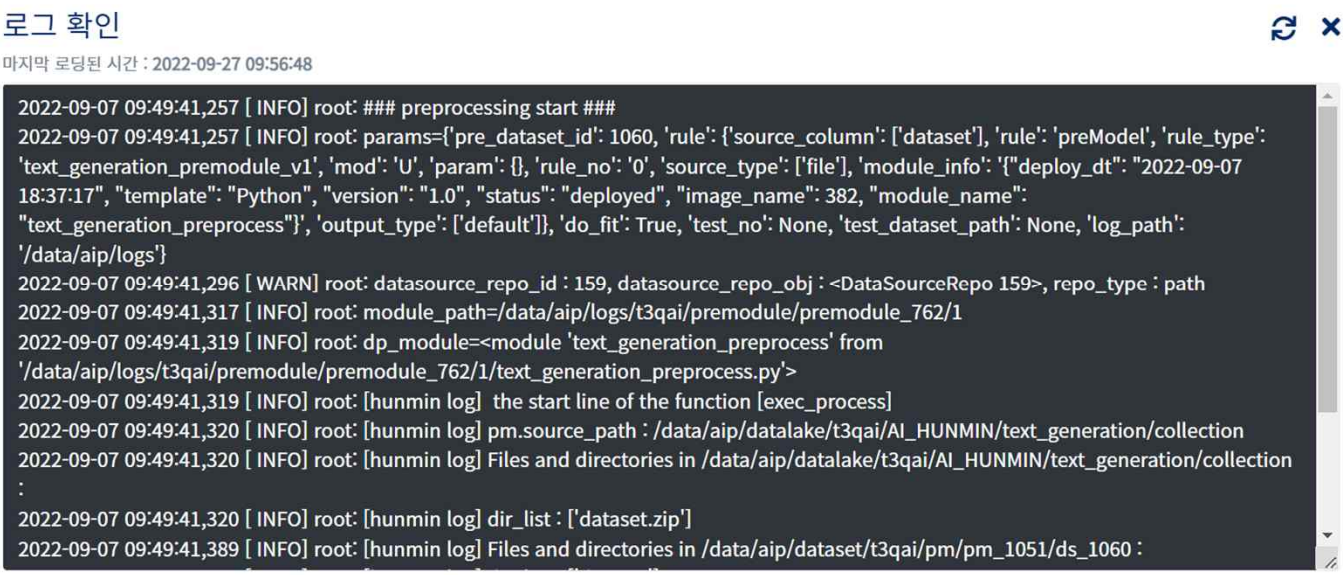
❸ 전처리 규칙 등록 시 절차
- 소스컬럼 : dataset(file) 선택
- 변환 함수 : text_generation_premodule 선택 후 [저장]

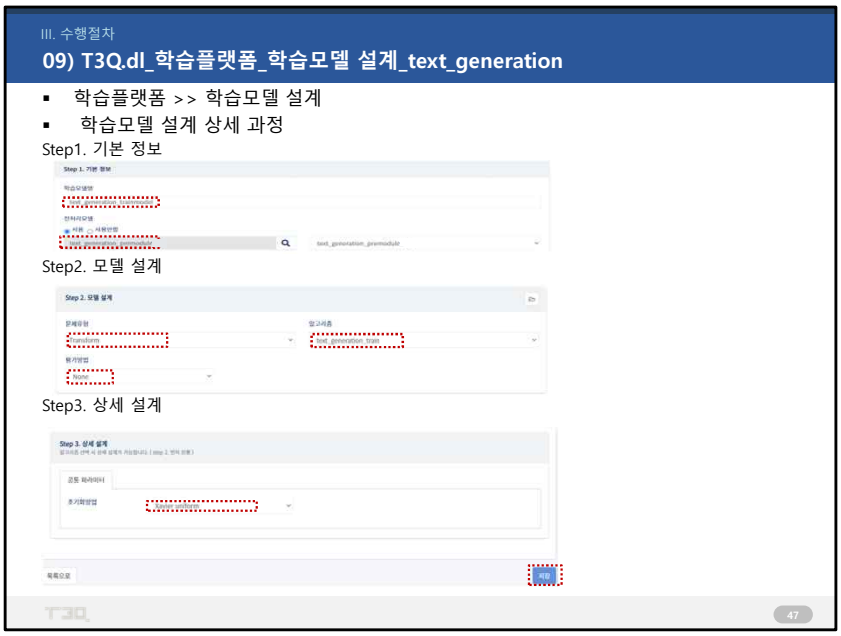






▪ 학습플랫폼 >> 전처리 모델 관리
▪ 전처리 모델 설계 상세
Step1. 기본 정보
- 모델명: text_generation_premodel
- 참조데이터셋 : text_generation_dataset
Step2. 데이터셋 컬럼정보
Step3. 전처리 룰 정보
Step4. 전처리 실행정보
전처리 상세
- 전처리 상세 버튼 누름
- 진행상황 확인
- 0_text_generation_premodule – [로그] 아래 [보기] 누름
[학습 모델 설계] 선택하여 다음 단계 진행





학습플랫폼 >> 학습모델 설계

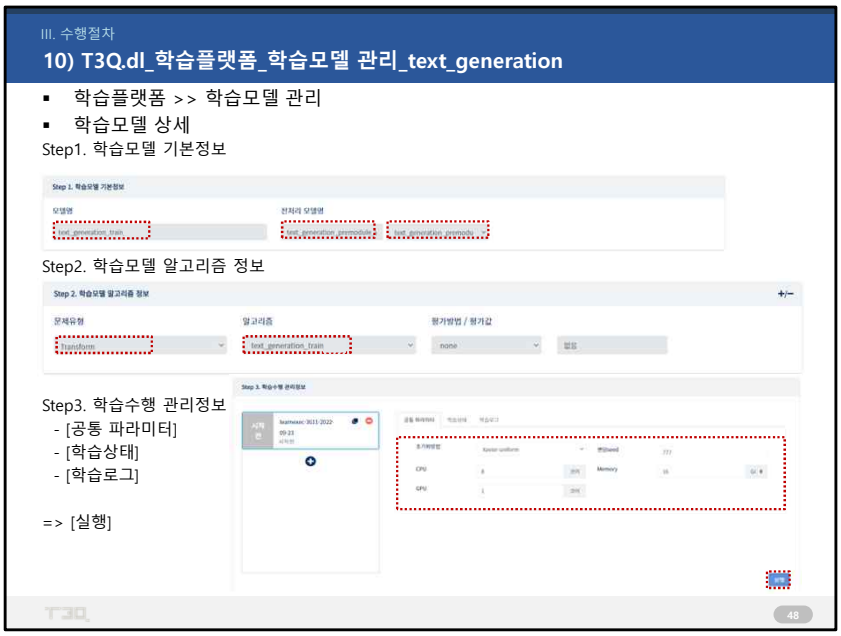
1. AI 훈민정음 >> 학습플랫폼 >> 학습모델 설계 상세 과정
   1) Step 1. 기본 정보
      학습모델명 text_generation_trainmodel
      전처리모델
        [사용] 체크
        text_generation_premodel text_generation_premodel

   2) Step 2. 모델 설계
      문제유형 Transform
      알고리즘 text_generation_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



학습플랫폼 >> 학습모델 관리

1. 학습플랫폼 >> 학습모델 관리

2 학습모델 상세

1) Step 1. 학습모델 기본정보
모델명 text_generation_trainmodel
전처리 모델명 text_generation_premodel text_generation_premodel

2) Step 2. 학습모델 알고리즘 정보
문제유형 Transform
알고리즘 text_generation_train
평가방법 / 평가값 none 없음

3) Step 3. 학습수행 관리정보
(1) 공통 파라미터
랜덤seed 777
CPU 8 코어
Memory 16 Gi
GPU 1 코어

(2) 학습상태
학습상태 시작전 [- ~ -]
Loss

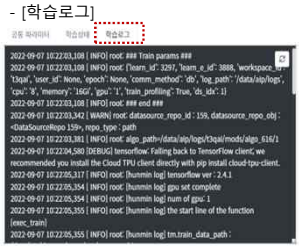
[실행] 버튼 누름

10) T3Q.dl_프로젝트 설정_학습모델 관리_text_generation

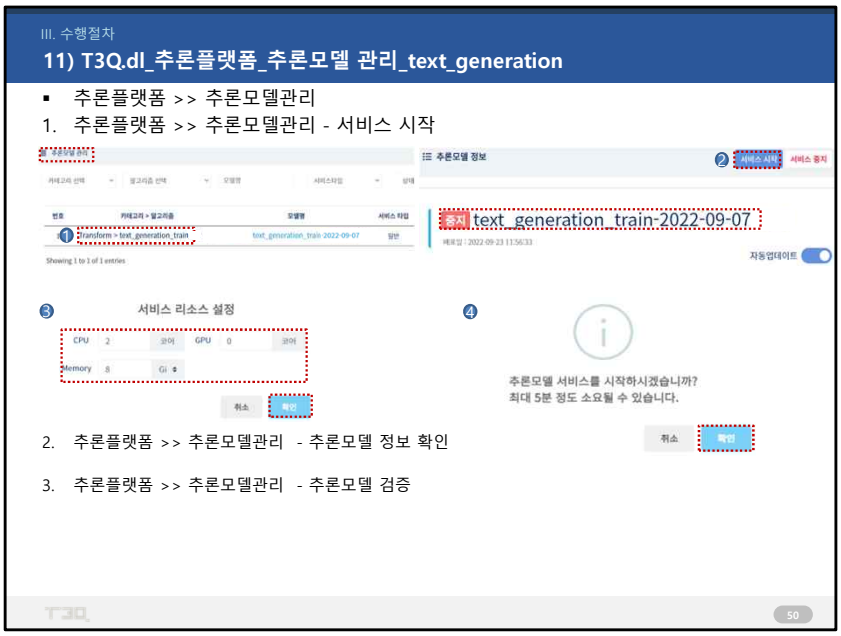
- 학습플랫폼 >> 학습모델 관리
- 학습모델 상세
- [실행] 완료 후

- [학습상태]

- [학습로그]

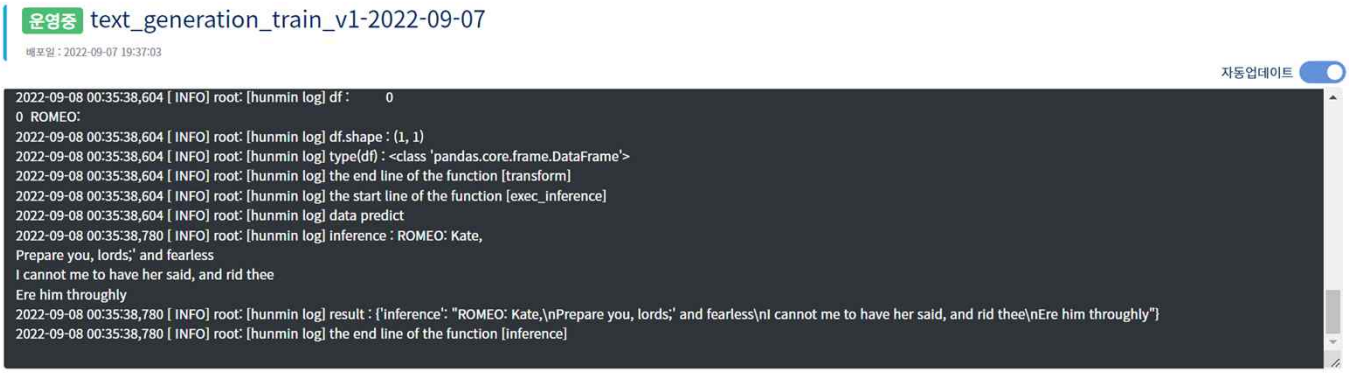


- 모델 배포 : [모델 배포] 버튼 누르고 [배포] 누름

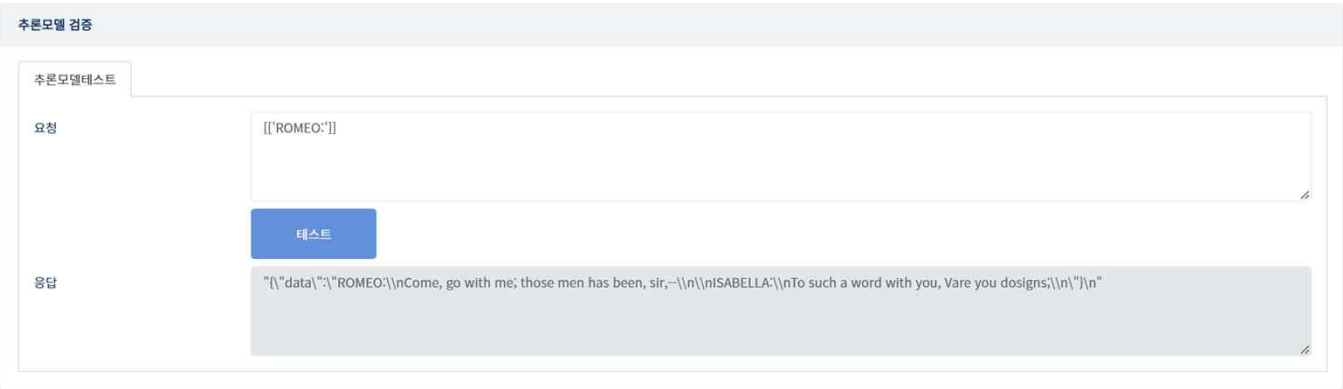


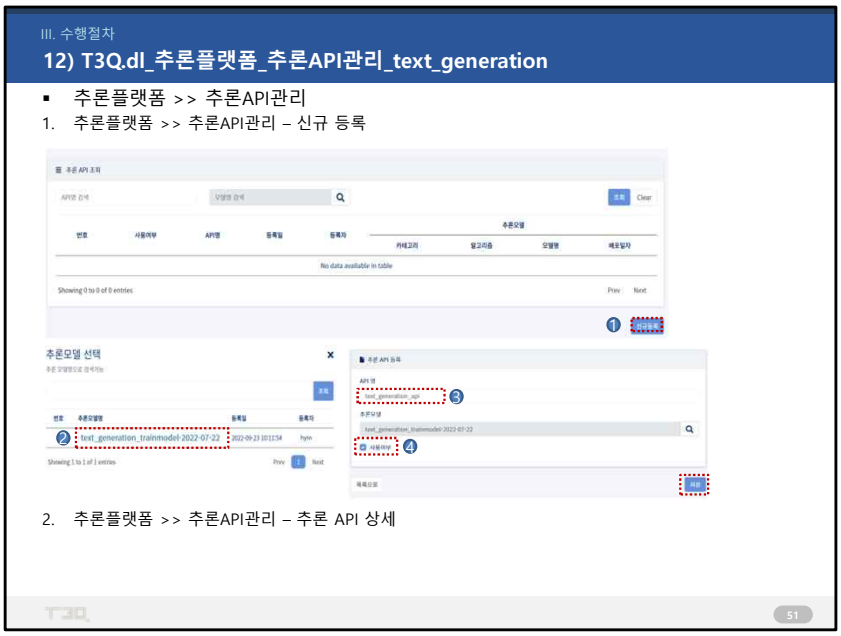
추론플랫폼 >> 추론모델관리

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



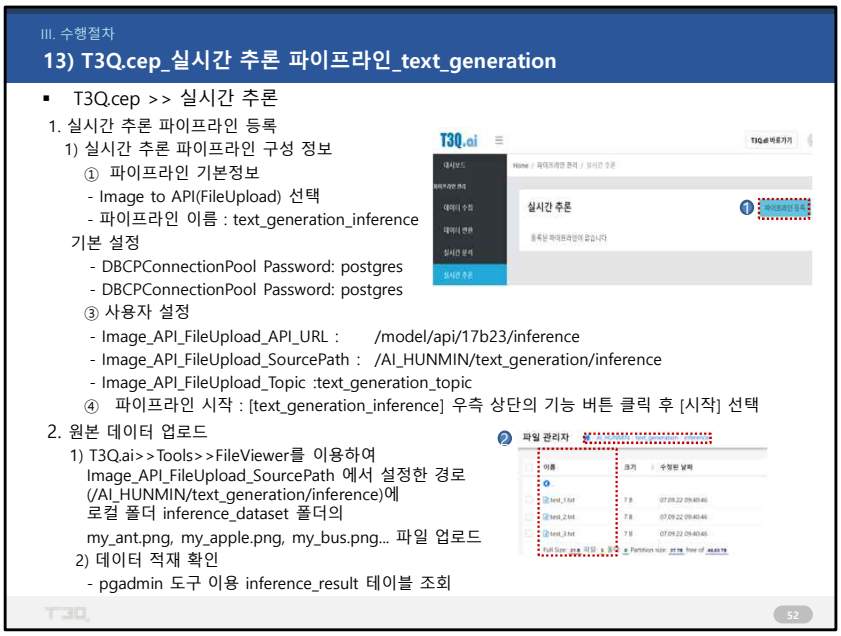
3. 추론플랫폼 >> 추론모델관리 - 추론모델 검증
[추론모델테스트]-[요청] 에 아래의 값을 넣고, [테스트] 눌러 수행
요청 : 입력 예시 : [['/data/aip/file_group/pm/pm_334/ds_441/image/1/1230.png']]
요청 : [['ROMEO:']]
응답 : "{₩"data₩":₩"ROMEO:₩₩nCome, go with me; those men has been, sir,--₩₩n₩₩nISABELLA:₩₩nTo such a word with you, Vare you dosigns;₩₩n₩"}₩n"





▪추론플랫폼 >> 추론API관리
1.추론플랫폼 >> 추론API관리 – 신규 등록
2.추론플랫폼 >> 추론API관리 – 추론 API 상세

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



(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 start_time like '%Mon, 13 Jun 2022%'
order by start_time desc

SELECT * FROM public.inference_result
where url like '%/model/api/17b23/inference%'
order by start_time desc