[제주도 도로 교통량 예측 AI 경진대회]

배경: 제주도내 주민등록인구는 2022년 기준 약 68만명으로, 연평균 1.3%정도 매년 증가하고 있습니다. 또한 외국인과 관광객까지 고려하면 전체 상주인구는 90만명을 넘을 것으로 추정되며, 제주도민 증가와 외국인의 증가로 현재 제주도의 교통체증이 심각한 문제로 떠오르고 있습니다.

주제: 제주도의 교통 정보로부터 도로 교통량 회귀 예측

<코드 분석>

```
#개발환경: Ubuntu 16.04.7 LTS, Python 3.6.10, tensorflow 2.3.0
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import gc
```

```
def csv_to_parquet(csv_path, save_name):
  df = pd.read_csv(csv_path)
  df.to_parquet(f'./{save_name}.parquet')
  del df
  gc.collect()
  print(save_name, 'Done.')
```

csv_to_parquet 함수는 CSV 파일을 Parquet 파일로 변환하는 역할을 한다. Parquet 파일은 열(columnar) 기반 저장 형식으로, 데이터 크기를 줄이고 읽기 속도를 개선하는 데 유용하다.

데이터프레임을 삭제(del df)하고, Python의 gc.collect()를 호출해 사용하지 않는 메모리를 해제한다. 이는 대용량 데이터를 다룰 때 메모리 누수를 방지하기 위함이다.

마지막줄은 완료 메시지를 출력하는 것이다.

```
csv_to_parquet('./train.csv', 'train')
csv_to_parquet('./test.csv', 'test')
```

```
train = pd.read_parquet('./train.parquet')
test = pd.read_parquet('./test.parquet')
```

(뒤에 이어서...)

```
str_col = ['day_of_week',
           'base_hour',
           'lane_count',
           'maximum_speed_limit',
           'start_latitude',
           'start_longitude',
           'end_latitude',
           'end_longitude',
           'road_rating',
           'weight_restricted',
           'start_turn_restricted',
           'end_turn_restricted',
           'start_node_name',
           'end_node_name',
           'road_type',
           'road_name',
           'connect_code',
           'multi_linked']
for i in str_col:

    le = LabelEncoder()
    le=le.fit(train[i])
    train[i]=le.transform(train[i])

    for label in np.unique(test[i]):
        if label not in le.classes_:
            le.classes_ = np.append(le.classes_, label)
    test[i]=le.transform(test[i])
```

```
train['day_of_week'] = ['Monday', 'Tuesday']
test['day_of_week'] = ['Monday', 'Wednesday']
# fit 결과: {'Monday': 0, 'Tuesday': 1}
# 'Wednesday'는 train에 없으므로 classes_에 추가 후 변환
# transform 후: test['day_of_week'] = [0, 2]
```

LabelEncoder()를 사용해 str_col에 포함된 문자형 변수를 정수형으로 변환하는 과정

test[i]의 고유값(범주)을 순회하며, train[i]에서 학습되지 않은 새로운 값(범주)이 있으면 LabelEncoder에 추가합니다. 추가한 후 transform을 사용해 테스트 데이터도 동일하게 정수로 변환합니다.

```
y_train = train['target']

X_train = train.drop(['id', 'target', 'vehicle_restricted', 'height_restricted', 'road_in_use'], axis=1)

test = test.drop(['id', 'vehicle_restricted', 'height_restricted', 'road_in_use'], axis=1)

print(X_train.shape)
print(y_train.shape)
print(test.shape)
```

```
(4701217, 19)
(4701217,)
(291241, 19)
```

```
import glob
import pandas as pd
import numpy as np

import gc
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.utils import shuffle
from tqdm import tqdm

import tensorflow as tf
from tensorflow.keras import *
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  try:
    for gpu in gpus:
      tf.config.experimental.set_memory_growth(gpu, True)
    logical_gpus = tf.config.experimental.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    print(e)
```

러닝 작업을 위한 환경 설정으로, GPU를 사용할 수 있는지 확인하고, TensorFlow에서 GPU 메모리 관리를 설정하는 역할

```
X_train
```

	base_date	day_of_week	base_hour	lane_count	road_rating	road_name	multi_linked	connect_code	maximum_speed_limit	weight_restricted	road_type
0	20220623	1	17	0	1	46	0	0	3	1	1
1	20220728	1	21	1	0	34	0	0	3	0	0
2	20211010	4	7	1	0	36	0	0	5	0	0
3	20220311	0	13	1	2	57	0	0	2	0	0
4	20211005	6	8	1	0	35	0	0	5	0	0
...	...	...	...	...	...	...	...	...	...	...	...
4701212	20211104	1	16	0	2	0	0	0	2	0	0
4701213	20220331	1	2	1	2	0	0	0	5	2	1
4701214	20220613	3	22	1	0	35	0	0	3	0	0
4701215	20211020	2	2	1	0	37	0	0	5	0	0
4701216	20211019	6	6	1	2	1	0	0	3	0	0

4701217 rows × 19 columns

```
X_train = np.array(X_train)
y_train = np.array(y_train)
test = np.array(test)
```

```
X_train.shape
```

(4701217, 19)

```
for t in range(19):
    print(np.max(X_train[:, t]))
```

```
max_temp = np.array([8, 9, 12, 17, 21, 24, 29, 29, 25, 21, 16, 10])
# https://www.google.com/search?q=%EC%A0%9C%EC%A3%BC+%EC%9B%94%EB%B3%84+%EA%B8%B0%EC%98%A8&oq=%EC%A0%9C%EC%A
# NOAA(미국 해양대기청)에서 제공하는 제주특별자치도 제주시 월별 평년값중 최고기온
#가장 최근 NOAA(미국 해양대기청)에서 업데이트한 평년값 범위 (1991년 - 2020년)

max_temp_max = np.max(max_temp)
max_temp = max_temp / max_temp_max #maximum값으로 나눠 normalize

holiday = np.array([20210920,
                    20210921,
                    20210922,
                    20211003,
                    20211004,
                    20211009,
                    20211011,
                    20211225,
                    20220101,
                    20220131,
                    20220201,
                    20220202,
                    20220301,
                    20220309,
                    20220505,
                    20220508,
                    20220601,
                    20220606,
                    20220815
                   ])
```

```
#https://search.naver.com/search.naver?sm=tab_hty.top&where=nexearch&query=2021+%EA%B3%B5%ED%9C%B4%EC%9D%BC&oquery=2022+%
#https://search.naver.com/search.naver?sm=tab_hty.top&where=nexearch&query=2022+%EA%B3%B5%ED%9C%B4%EC%9D%BC&oquery=2021+%
#2021, 2022 공휴일

x_train = []
for k, d in tqdm(enumerate(X_train)):
    month_i = np.int((d[0]%10000)/100) - 1
    d = np.concatenate([
        d[1:], [np.minimum(3,np.min(np.abs(holiday - d[0]))), max_temp[month_i]] #input data에 휴무일과의 차이(0일, 1일, 2일, 3일 이상), 최고기온에 대한
        ])
    x_train.append([d, y_train[k]])
x_test = []
for d in tqdm(test):
    month_i = np.int((d[0]%10000)/100) - 1

    d = np.concatenate([
        d[1:], [np.minimum(3, np.min(np.abs(holiday - d[0]))), max_temp[month_i]]
        ])
    x_test.append(d)
```

날씨와 공휴일 정보를 활용해 학습 데이터(x_train)와 테스트 데이터(x_test)를 확장하는 과정을 보여줌. 주어진 기온과 공휴일 데이터를 기반으로 입력 데이터를 보강하여 모델이 휴일 및 계절성을 반영할 수 있도록 설계

max_temp는 NOAA(미국 해양대기청)에서 제공한 제주시 월별 최고 기온 데이터를 나타냄

데이터는 np.max()를 통해 최대값으로 정규화(normalize)되어, 모든 값이 0과 1 사이로 변환됨 //

holiday는 2021년과 2022년의 공휴일을 YYYYMMDD 형식으로 배열에 저장한 것

휴일 데이터는 학습 및 테스트 데이터에서 날짜 정보를 비교하기 위해 사용 //

d[0]은 날짜를 나타내며, YYYYMMDD 형식임

이를 통해 날짜에서 월 정보를 추출하고, 배열 인덱스를 위해 1을 뺌

- 예: 20210920 → month_i = 8 (9월) //

holiday 배열과 d[0] 간의 절대 차이를 계산하여 공휴일과의 최소 일수를 구함

최소 거리가 3일보다 크면 값을 3으로 제한

- 예: d[0] = 20210921, holiday = [20210920, 20210921, ...] → 공휴일 거리 = 0 //

month_i를 기반으로 정규화된 월별 최고 기온 값을 가져옴

- 예: month_i = 8 (9월) → max_temp[8] = 0.8621 //

기존 데이터에서 첫 번째 열(d[0], 즉 날짜)을 제외하고, 새로 계산한 두 정보를 추가

최종적으로 확장된 데이터를 x_train에 추가 //

x_train과 동일한 논리로 테스트 데이터를 확장

단, 테스트 데이터에는 y_train(정답 레이블)이 없으므로 이를 추가하지 x

결과적으로, 확장된 입력 데이터만 x_test에 저장

(뒤에 이어서...)

```
def known_data_model(input_layer, start_neurons):
    #각각의 input값에 대한 embedding
    input_dims = [7, 24, 3, 3, 61, 2, 2, 6, 4, 2, 487, 586, 586, 2, 487, 586, 586, 2, 4]

    for i in range(20):
        if i==0:
            input_embedding = layers.Embedding(input_dim=input_dims[i], output_dim=start_neurons)(input_layer[:, i])
        elif i >= 19:
            input_embedding = layers.concatenate([input_embedding, layers.Dense(start_neurons)(input_layer[:, i:i+1])])
        else :
            input_embedding = layers.concatenate([input_embedding, layers.Embedding(input_dim=input_dims[i], output_dim=start_neurons)(input_


    print(input_embedding.get_shape().as_list())

    all_layer = input_embedding

    for layer_num in range(5):
        all_layer_d = layers.Dropout(0.2)(all_layer)
        all_layer_d_gate = layers.Dense(all_layer_d.get_shape().as_list()[-1])(all_layer_d)
        all_layer_ = all_layer * tf.math.sigmoid(all_layer_d_gate) #weighted sigmoid gate unit
        all_layer_c = layers.concatenate([all_layer, all_layer_])
        all_layer += layers.Dense(20*start_neurons, activation='relu')(all_layer_c)
```



```
    output1 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output2 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output3 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output4 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output5 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output6 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output7 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output8 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output9 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)
    output10 = tf.squeeze(layers.Dense(1)(all_layer), axis=-1)

    output = (output1 + output2 + output3 + output4 + output5 + output6 + output7 + output8 + output9 + output10) / 10 #average output
    return output
```

```
strategy = tf.distribute.MirroredStrategy() # multi GPU parallelization strategy
```

```
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0', '/job:localhost/replica:0/task:0/device:GPU:1')
```

입력 데이터에 대한 Embedding 처리와 신경망 레이어를 구성하여 최종적으로 여러 출력을 평균화하는 딥러닝 모델을 정의한 코드

```
#fold include day (need test w/o day), include output_dim, 256, layer 6, w/ all BatchNorm (need test w/o BatchNorm),


# Ensemble codes in public discussion were used.
def trainGenerator():
    for data in train_data_:
        target = data[1]
        feature = data[0]
        yield (feature, target)
def valGenerator():
    for data in val_data:
        target = data[1]
        feature = data[0]
        yield (feature, target)
kfold_list = [2, 3, 4, 5, 6, 10, 20]
for kfold in kfold_list:
    kf = KFold(n_splits=kfold, random_state=42, shuffle=True)
    for fold, (train, val) in enumerate(kf.split(x_train)):


        val_data = np.array(x_train)[val]
        train_data_ = np.array(x_train)[train]



        tr_ds = tf.data.Dataset.from_generator(trainGenerator, (tf.float32, tf.float32), (tf.TensorShape([20]), tf.TensorShape([])))
        tr_ds = tr_ds.cache()
        tr_ds = tr_ds.shuffle(100000).padded_batch(4096)
        tr_ds = tr_ds.prefetch(tf.data.experimental.AUTOTUNE)

        val_ds = tf.data.Dataset.from_generator(valGenerator, (tf.float32, tf.float32), (tf.TensorShape([20]), tf.TensorShape([])))
        val_ds = val_ds.cache()
        val_ds = val_ds.batch(4096).prefetch(tf.data.experimental.AUTOTUNE)

        with strategy.scope():
            input_layer = Input((20))



            outputs = known_data_model(input_layer, 32)
            model = Model(input_layer, outputs)


            adam = tf.keras.optimizers.Adam()
            model.compile(optimizer=adam,
                loss=tf.keras.losses.MeanAbsoluteError())
            callbacks = tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_loss', factor=.1, patience=2, verbose=0, mode='min', min_delta=1e-4, cooldown=0, min_lr=0
            )
            sv = tf.keras.callbacks.ModelCheckpoint(
            f'./models/ohfohf-{fold}-read_all_org_fold_{kfold}.h5', monitor='val_loss', verbose=0, save_best_only=True
```



```
            del model
            gc.collect()
print('Training complete.')
```

K-Fold 교차 검증과 TensorFlow 모델 학습

```
pred = []
for kfold in kfold_list:
    for n_fold in range(kfold):
        input_layer = Input((20))
        outputs = known_data_model(input_layer, 32)
        model = Model(input_layer, outputs)
        model.load_weights(f'./models/ehfehf-{n_fold}-road_all_org_fold_{kfold}.h5')
        val_pred = model.predict(np.array(x_test))
        pred.append(val_pred)
```

```
pred_sum = sum(pred)
```

K-Fold 교차 검증을 활용한 앙상블 예측을 수행함. 각 폴드에서 모델을 불러와 테스트 데이터에 대한 예측값을 계산하고, 최종적으로 모든 폴드의 예측값을 합함(pred_sum)

```
pred_sum /= len(pred)
pred_sum
```

```
array([25.806948, 42.550606, 66.98541 , ..., 22.449648, 21.991238,
       48.92733 ], dtype=float32)
```

```
val_pred = np.round(pred_sum) #정수화
```

```
val_pred
```

```
array([26., 43., 67., ..., 22., 22., 49.], dtype=float32)
```

```
sample_submission = pd.read_csv('./sample_submission.csv')
sample_submission['target'] = val_pred
sample_submission.to_csv("./submit.csv", index = False)
```