제주도 도로 교통량 데이터 머신러닝 예측

2022.11.02

박소영

✓ 데이터 개요

Train.csv

4,701,217개의 데이터

train.shape
(4701217, 24)

- id : 샘플 별 고유 id
- 날짜, 시간, 교통 및 도로구간 등 정보
- target : 도로의 차량 평균 속도(km)

Test.csv

291,241개의 데이터

test.shape
(291241, 23)

- id : 샘플 별 고유 id
- 날짜, 시간, 교통 및 도로구간 등 정보

data_info.csv

데이터의 각 Column 별 설명

data_info.shape
(24, 2)

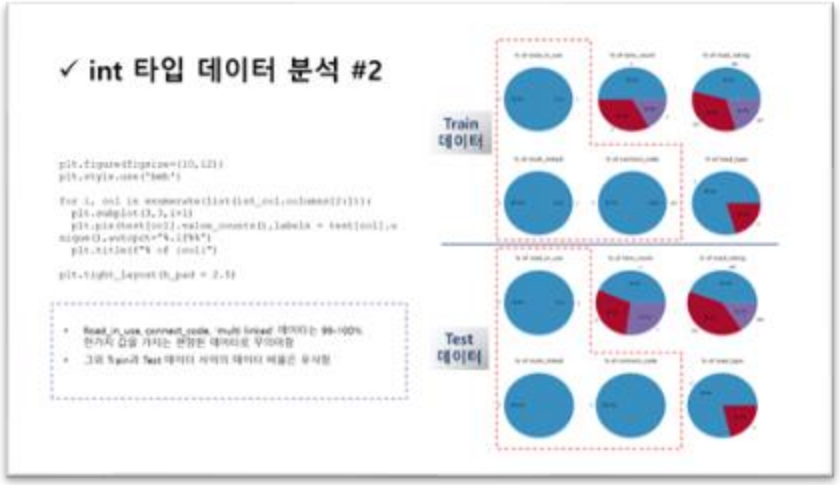
```
print("Train dataset shape is",train.shape," and Test dataset shape is ",test.shape)
```

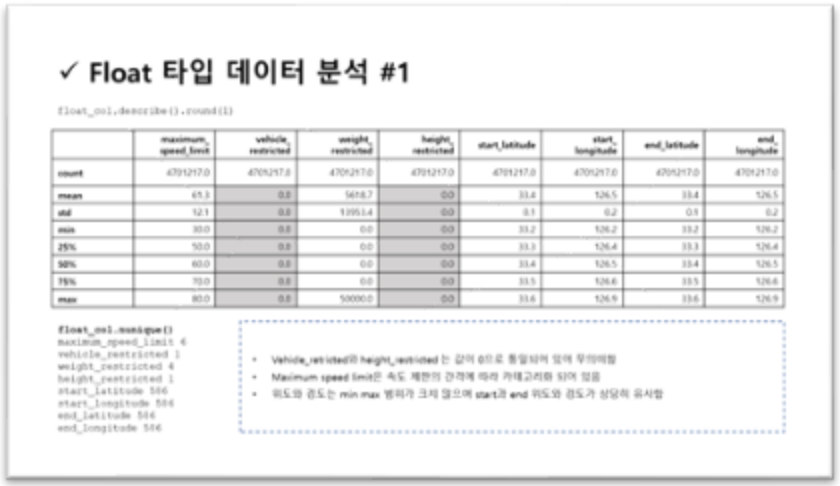
>> Train dataset shape is (4701217, 24) and Test dataset shape is (291241, 23)

```
print("Train data has " + str(train.isnull().sum().sum()) + " null values and Test data has " + str(test.isnull().sum().sum()) + " null values")
```

>> Train data has 0 null values and Test data has 0 null values (결측치 없음)

✓ 불필요한 데이터 피처 drop



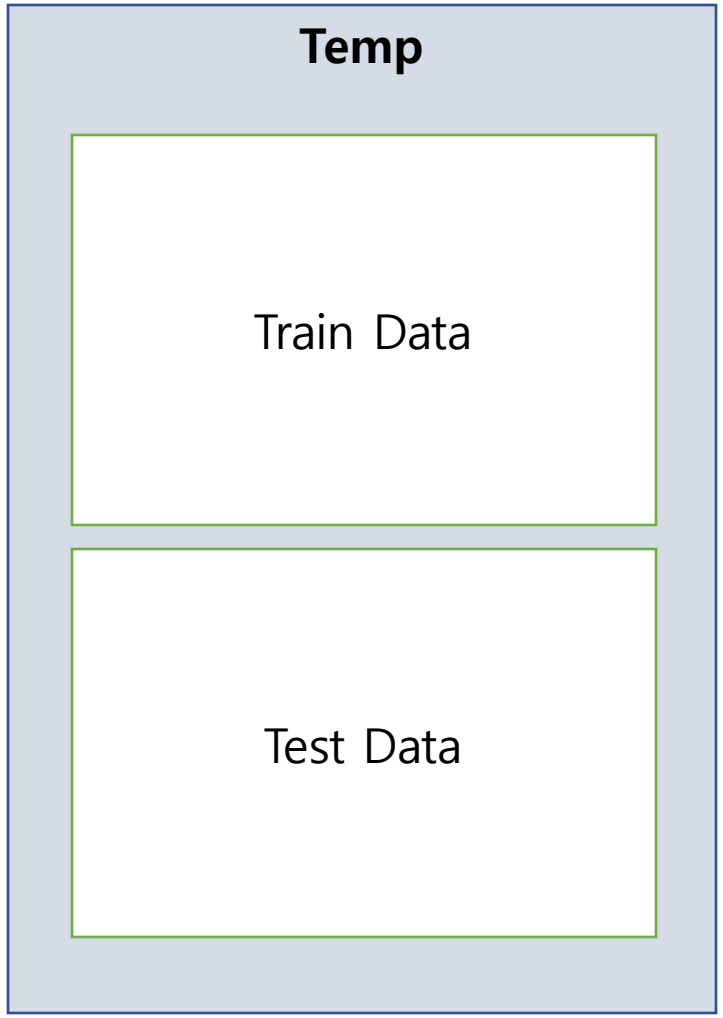


시각화를 통해 파악한 편향된, 무의미한 컬럼 제거

```
train.drop(['id','road_in_use','multi_linked','connect_code','height_restricted','vehicle_restricted','end_latitude','end_longitude'],axis=1,inplace=True)
```

```
test.drop(['id','road_in_use','multi_linked','connect_code','height_restricted','vehicle_restricted','end_latitude','end_longitude'],axis=1,inplace=True)
```

✓ 피처 스케일링 / 변환을 위한 train + test 병합




```
train['source'] = "train"
test['source'] = "test"

#이후 데이터셋 분리를 위한 추가 컬럼활용
```

```
temp = pd.concat([test,train])

temp.shape
(4992458, 18)
```

✓ Datetime 피처 포맷변경/칼럼추가

base_month

base_quarter

base_day

~~Base_date~~

```
train['base_date'] = pd.to_datetime(train['base_date'],format="%Y%m%d")
test['base_date'] = pd.to_datetime(test['base_date'],format="%Y%m%d")

temp['base_month'] = temp['base_date'].dt.month
temp['base_quarter'] = temp['base_date'].dt.quarter
temp['base_day'] = temp['base_date'].dt.day

temp.drop('base_date',axis=1,inplace=True)
```

✓ 전처리를 위한 데이터타입별 컬럼 확인

```
object_info = pd.DataFrame({'object':temp.select_dtypes('object').columns})
int_info = pd.DataFrame({'int':temp.select_dtypes('int64').columns})
float_info = pd.DataFrame({'float':temp.select_dtypes('float64').columns})
```

	object
0	day_of_week
1	road_name
2	start_node_name
3	start_turn_restricted
4	end_node_name
5	end_turn_restricted
6	source

	int
0	base_hour
1	lane_count
2	road_rating
3	road_type
4	base_month
5	base_quarter
6	base_day

	float
0	maximum_speed_limit
1	weight_restricted
2	start_latitude
3	start_longitude
4	target

카테고리형 데이터는
숫자형 데이터로,

숫자형 데이터는
스케일링 필요

✓ Road_name, start_node_name, end_node_name 그룹화

```
def categorize_road(df,col):
  df.loc[df[col].str.contains("일반국도"),col] = "general_road"
  df.loc[df[col].str.contains("지방도"),col] = "rural"
  df.loc[df[col].str.contains("로$",regex=True),col] = "general"
  df.loc[df[col].str.contains("교$",regex=True),col] = "bridge_road"
  df.loc[df[col].str.contains("-"),col] = "error"
  df.loc[df[col].str.contains("^((?!general_road|rural|general|bridge_road|error).)*$",regex=True),col] = "others"

def categorize(df,col):
  df.loc[df[col].str.contains("교차로"),col] = "intersect"
  df.loc[df[col].str.contains("교$|.측",regex=True),col] = "bridge"
  df.loc[df[col].str.contains("거리$|사거",regex=True),col] = "road"
  df.loc[df[col].str.contains("주택|아파트$|아파트[0-9]|오피스텔|빌라|맨션|빌리지$|여관"),col] = "living"
  df.loc[df[col].str.contains("입구$|입$|입구[0-9]$",regex=True),col] = "entrance"
  df.loc[df[col].str.contains("마을$",regex=True),col] = "village"
  df.loc[df[col].str.contains("청$|경찰|복지|파출소",regex=True),col] = "government"
  df.loc[df[col].str.contains("공장|창고|목장|농장",regex=True),col] = "industry"
  df.loc[df[col].str.contains("^((?!intersect|bridge|living|entrance|government|industry|village|road).)*$",regex=Tru
e),col] = "others"
```

오버피팅 방지 목적

Start_node_name, end_nod_name 487 개 unique values → Start_node_name, end_nod_name 9 개 unique values

Road_name 61 개 unique values → Start_node_name, end_nod_name 6 개 unique values

✓ 문자형 데이터 => 원핫인코더
✓ 수치형 데이터 => StandardScaler

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer

num_cols = temp.select_dtypes(include=['int64','float64']).columns.tolist()
obj_cols = temp.select_dtypes(include=['object']).columns.tolist()


ct = make_column_transformer((StandardScaler(),num_cols),(OneHotEncoder(),obj_cols))
ct.fit(temp)
col_names = num_cols + ct.transformers_[1][1].get_feature_names_out(obj_cols).tolist()

preprocessed_df = pd.DataFrame(ct.transform(temp),columns=col_names)
preprocessed_df
```

✓ 전처리 완료 후, 데이터셋 다시 train, test 분리

```
train = preprocessed_df.loc[preprocessed_df['source_train']==1,:]
test = preprocessed_df.loc[preprocessed_df['source_test']==1,:]

print(f'train data shape is {train.shape}, test data shape is {test.shape}’)
>> train data shape is (4701217, 47), test data shape is (291241, 46)
```

#source 데이터도 원핫인코딩 처리가 됐으므로 각 source_train, source_test가 1인 값으로 분리

✓ 회귀 모델 성능 점검을 위한 rmse 함수 활용

```
from sklearn.model_selection import cross_val_score,KFold

n_folds = 5

def rmse(model):
  kf = KFold(n_folds,shuffle=True,random_state=42).get_n_splits(train.values)
  rmse = np.sqrt(-
cross_val_score(model,x_train,y_train,scoring='neg_mean_squared_error',cv=kf))
  return rmse
```

✓ 회귀 모델 임의 1차 적용

```
from sklearn.linear_model import ElasticNet, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import lightgbm as lgb
import xgboost as xgb


# lasso = Lasso(alpha=0.0005,random_state=12)

# ELNet = ElasticNet(alpha=0.0005,l1_ratio=0.4,random_state=11)

# Rg = Ridge(alpha=0.4,random_state=10)

# GBR = GradientBoostingRegressor(n_estimators=4000,learning_rate=0.05,max_depth=4,max_fea
tures='sqrt',min_samples_split=10)

# xgb_m = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
#                                learning_rate=0.05, max_depth=3,
#                                min_child_weight=1.7817, n_estimators=2200,
#                                reg_alpha=0.4640, reg_lambda=0.8571,
#                                subsample=0.5213, silent=1,
#                                random_state =7, nthread = -1)

# lgb_m = lgb.LGBMRegressor(objective='regression',num_leaves=5,
#                                 learning_rate=0.05, n_estimators=720,
#                                 max_bin = 55, bagging_fraction = 0.8,
#                                 bagging_freq = 5, feature_fraction = 0.2319,
#                                 feature_fraction_seed=9, bagging_seed=9,
#                                 min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
```

캐글의 회귀 관련 대회 상위권 코드 참조

✓ 회귀 모델결과 바탕 사용 모델 선별

```
# rmse(lasso) array([0.70785874, 0.70765433, 0.70810295, 0.70766365, 0.70791972])
# rmse(ELNet) array([0.70770386, 0.70745235, 0.70792294, 0.70749008, 0.70772522])

# rmse(Rg) array([0.7076432 , 0.70730146, 0.70780776, 0.70740209, 0.70759758])

# rmse(GBR)  array([0.5445078 , 0.54463975, 0.54512056, 0.54535376, 0.54478809])
# rmse(model_lgb)  array([0.51892352, 0.52100445, 0.52094264, 0.52213064, 0.52290713])
```

rmse(XGB) 는 시간이 오래 걸려서 오류가 남

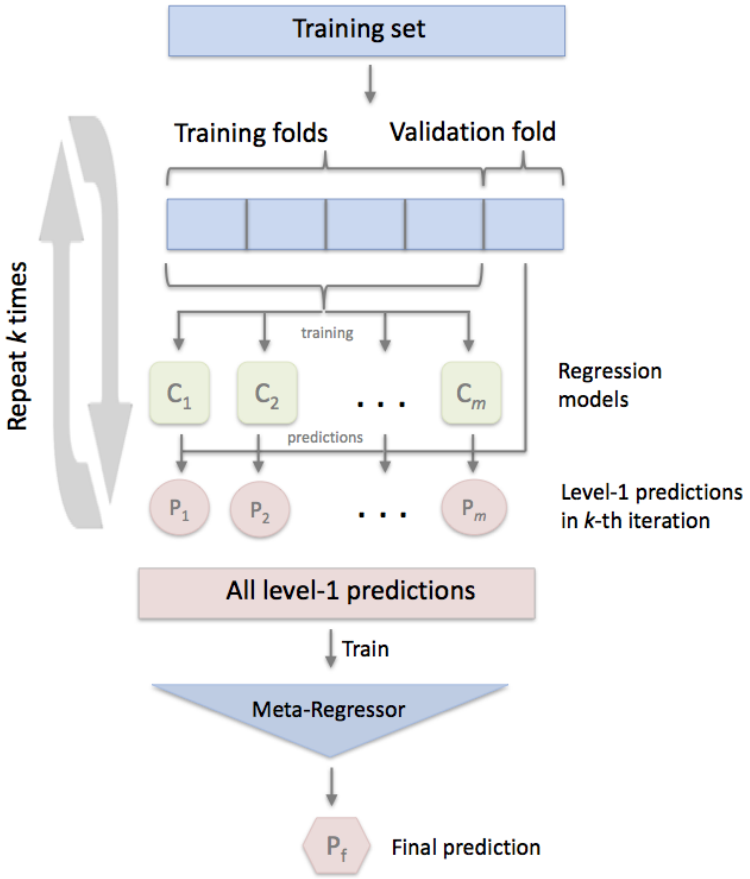
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

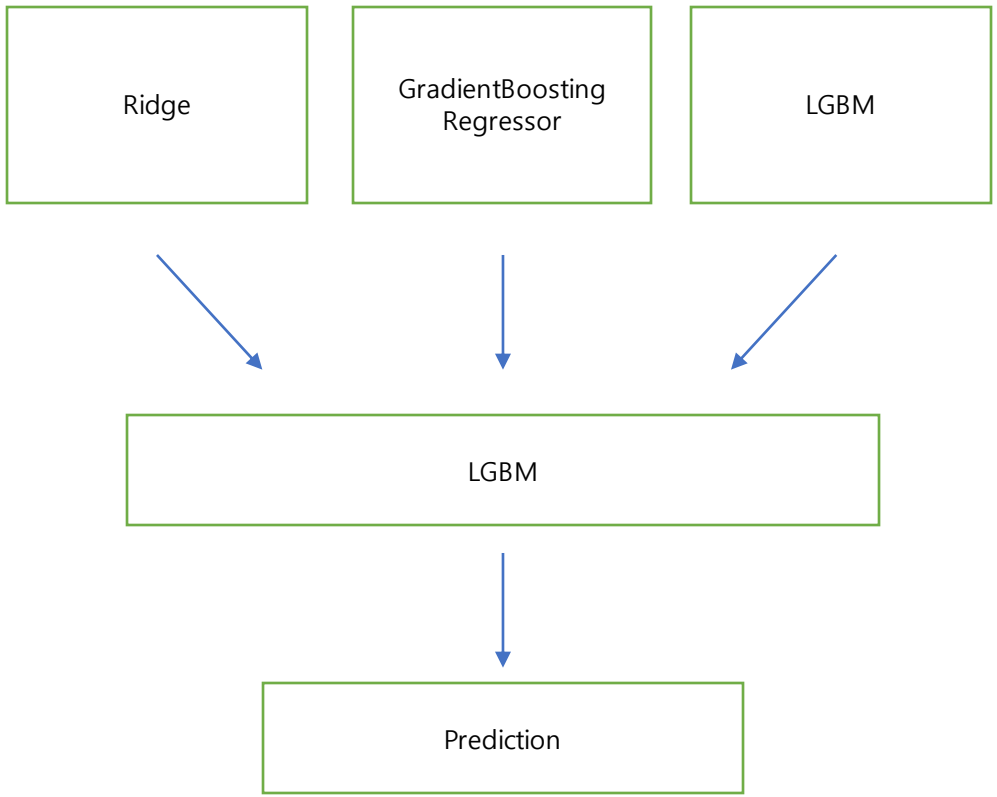
✓ GridSearchCV, RandomizedSearchCV 최적 파라미터 선별

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

#Ridge params
Rg = Ridge()
Ridge_param = {'alpha': [0.1,0.2,0.3,0.4,0.5], 'random_state' :[10]}

#GBR params
GBR = GradientBoostingRegressor()
GBR_param = {'n_estimators':[5000,6000],
             'max_depth':[3,4],
             'min_samples_split':[20,30],
             'learning_rate': [0.02,0.05]
}

#lgb params
lgb_m = lgb.LGBMRegressor()
LGB_param = {'num_leaves': [5,10,15,20],
             'max_depth': [3,4,5],
             'learning_rate': [0.02,0.05],
             'n_estimators':[2000,4000,6000],
             'min_data_in_leaf': [5,10],
             'min_gain_to_split':[0,0.2,0.4],
             'min_sum_hessian_in_leaf': [10,20]
}

param_list = [Ridge_param,GBR_param,LGB_param]
clf_list = [Rg,GBR,lgb_m]
```

✓ Stacking 모델을 통한 최종 결과 예측 시도 예정

```
from mlxtend.regressor import StackingCVRegressor
```





감사합니다