Decision Tree를 이용한 음식 배달 시간 예측

COSE362(02) Machine Learning

Final Report

2021320010 김수아

2021320077 윤현지

2021320097 이현서

2021320122 김정우

1. 문제 정의

간편한 배달 앱의 등장으로 배달 요식업의 수요가 점차 늘어가고 있던 와중 , 코로나 19로 인해 온라인 서비스에 대한 수요가 폭발적으로 증가하였다. 배달업은 업종의 특성상 예기치 못한 상황으로 서비스의 품질이 저하될 수 있으며 이는 날씨, 교통 상황, 이동 거리 등 다양한 요인의 영향을 받는다.
본 보고서는 날씨, 교통 상황 등 특정 구역에서 측정된 데이터와 여러가지 방법론을 적용하여 최적화된 배달 시간 예측 모델을 구현한다. 이를 통해 정확한 도착 예상 시간을 산출하여 배달 외식 서비스의 품질을 향상하고 고객 만족도 개선에 관한 인사이트를 제공하는 것을 목표로 한다.

2. 방법론

1) 데이터 전처리

a. data slicing

학습을 진행하기에 앞서 원본의 data[1]를 학습하기 좋은 형태로 가공하는 것이 선행되어야 한다. midterm report 때와 마찬가지로 각 value에 있는 값을 보고 불필요하거나 중복되는 부분들을 slicing하여 value들을 가공했다.

```
train['Weatherconditions'] = train['Weatherconditions'].map(lambda x: str(x)[11:])
train['Time_taken(min)'] = train['Time_taken(min)'].map(lambda x: str(x)[6:])
train['Order_Date'] = train['Order_Date'].map(lambda x: str(x)[3:5])
train['Time_Orderd'] = train['Time_Orderd'].map(lambda x: str(x)[:5])
train['Time_Order_picked'] = train['Time_Order_picked'].map(lambda x: str(x)[:5])
```

b. missing value(Regression based single imputation)

먼저 isnull() 메서드를 사용하여 데이터에 있는 missing value 들을 확인하였다.

---

[1] https://www.kaggle.com/datasets/gauravmalik26/food-delivery-dataset?select=test.csv
위 주소의 'train.csv', 'test.csv', 'Sample_Submission.csv'를 이용함.

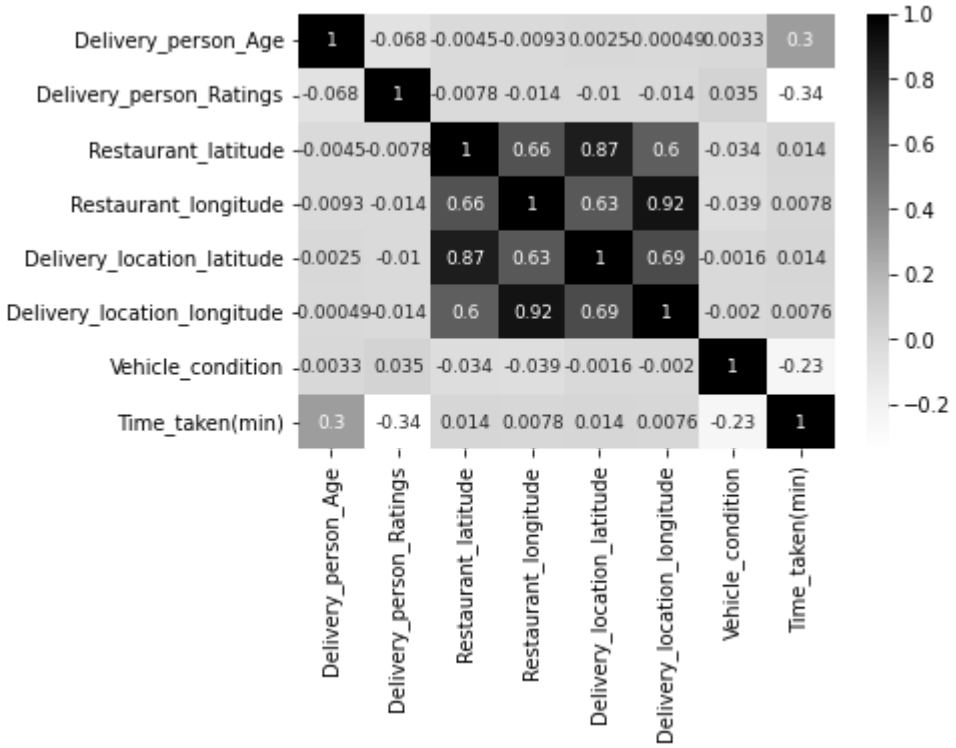
```
ID                             0
Delivery_person_ID             0
Delivery_person_Age          491
Delivery_person_Ratings      507
Restaurant_latitude            0
Restaurant_longitude           0
Delivery_location_latitude     0
Delivery_location_longitude    0
Order_Date                     0
Time_Orderd                  444
Time_Order_picked              0
Weatherconditions            158
Road_traffic_density         154
Vehicle_condition              0
Type_of_order                  0
Type_of_vehicle                0
multiple_deliveries          238
Festival                      65
City                         324
dtype: int64
```

위에서 볼 수 있듯이 배달원 평점과 같은 continuous 한 feature 나 날씨와 같은 discrete 한 feature 모두에서 missing value가 있다는 것을 확인할 수 있었다.

midterm report에서는 continuous 한 missing value 들에 대해 단순 평균으로 이를 대체하는 과정을 적용하였다. 평균으로 missing value를 대체하는 것은 값을 제거하는 방법보다는 정보량 손실을 줄인다는 점에서 장점이 있지만, 평균값을 대체값으로 넣기 때문에 분산이 감소하는 효과를 가져올 수 있고 공분산과 상관계수에 혼란을 줄 수 있다.

이렇게 원본 데이터의 분산을 왜곡시킨다는 한계를 극복하기 위해 최종적으로 regression 기반의 single imputation을 채택하였다. 이는 데이터의 feature들 끼리 상관관계가 있다고 가정하고, missing value를 다른 feature을 통한 회귀식의 예측값으로 대체하는 것이다.

```
a=train.loc[:,['Delivery_person_Ratings','Vehicle_condition','Delivery
_person_Age']]
seaborn.heatmap(train.corr(),annot=True,annot_kws={"size":
9},cmap='Greys')
```



첫번째로 상관관계를 나타낸 히트맵을 통해 배달원 평점은 배달원 나이, 이동수단 컨디션과 관련이 있다고 가정했고, 이 변수들을 통해 선형회귀 모델을 만들었다.

```
x=test.dropna(axis=0)[['Delivery_person_Age','Vehicle_condition']]
y=test.dropna(axis=0)[['Delivery_person_Ratings']]
lin_reg_model_Ratings=lin_reg.fit(x,y)
```

만들어진 모델을 통해 각 row의 배달원 평점을 예측하였고 NaN 값을 모두 예측값으로 채워 넣었다. 다른 feature 들도 상관관계를 가질 것 같은 feature들을 통해 선형회귀 모델을 만들고 똑같은 과정을 반복하였다.

```
Ratings_pred_train=lin_reg_model_Ratings.predict(train.loc[:,['Deliver
y_person_Age','Vehicle_condition']])
train['Delivery_person_Ratings'].fillna(pd.Series(Ratings_pred_train.f
latten()), inplace=True)
Ratings_pred_test=lin_reg_model_Ratings.predict(test.loc[:,['Delivery_
person_Age','Vehicle_condition']])
test['Delivery_person_Ratings'].fillna(pd.Series(Ratings_pred_test.fla
tten()), inplace=True)
```

선형회귀 모델을 사용하기 까다로운 discrete 한 missing value에 대해서는 midterm 에서 다음 row의 값으로 대체하였다. 이는 데이터의 전체적인 분포를 전혀 고려하지 않은 방법이었기에 이번에는 최빈값을 대체값으로 선택하여 개선하였다.

```
to_be_filled=['Weatherconditions','Road_traffic_density','City','multi
ple_deliveries','Festival']
for i in to_be_filled:
```

```
train[i]=train[i].fillna(train[i].mode()[0])
test[i]=test[i].fillna(test[i].mode()[0])
```

c. 범주형 데이터 수치화**(label encoding)**

midterm report 와 같은 방식으로 존재하는 범주형 feature 들을 pandas 내부에서 제공하는 categorical 함수를 사용해 수치화하였다.각 column들의 value에 categorical을 적용하면 자동으로 숫자가 지정되는데, 소스코드의 .codes가 이 숫자에 해당한다.

```
for column,value in data.items():
    if not pd.api.types.is_numeric_dtype(value):
        data[column]=pd.Categorical(value).codes+1
```

d. **outlier** 탐지

midterm report의 데이터 전처리 과정에서는 데이터의 outlier들을 고려하지 않았다. outlier 탐지란 극단적으로 크거나 작은 값, 혹은 데이터의 이상치에서 멀리 떨어진 데이터들을 감지하고, 이를 제거하여 결과값의 왜곡을 방지하는 것이다.

outlier 탐지를 위해서 두가지의 방법을 사용하였다. 첫번째는 z-score를 이용하는 방법이다. z-score란 데이터값에서 평균을 뺀 후 표준편차로 나눠준 값으로, 이를 이용하면 데이터가 평균에서 얼마나 벗어나 있는지를 측정할 수 있다. threshold의 값을 정하고 범위 밖의 데이터들을 outlier로 탐지한다. 일반적으로 threshold의 값은 2 정도로 잡는다. (97.7%)

```
from scipy import stats

data = train['distance']
zscore_threshold = 2

# stats의 zscor 함수를 이용해 데이터를 zscore 값으로 변환
outliers = data[np.abs(stats.zscore(data)) > zscore_threshold]
```

두번째로 사용한 방법은 제 3사분위수에서 제 1사분위수를 뺀 interquartile range(IQR)를 계산하는 것이다. 하위 이상치를 1사분위수에서 1.5*IQR을 뺀 값보다 작은 값으로, 상위 이상치를 3사분위수에서 1.5*IQR을 더한 값보다 큰 값으로 지정하고, 이에 속하는 값들을 outlier로 탐지한다.

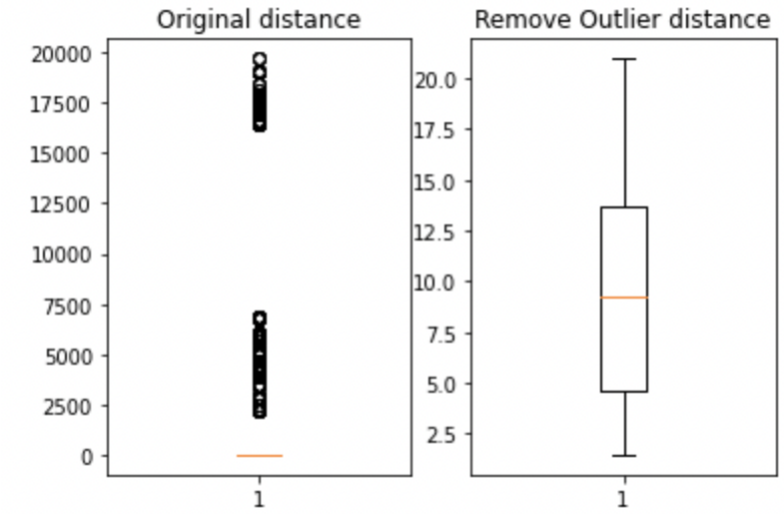
```
data = train['distance']
IQR = np.percentile(data,75) - np.percentile(data,25)

# numpy의  percentile 함수를 이용해 사분위수 구하기
upper_fense = np.percentile(data,75) + 1.5 * IQR
lower_fense = np.percentile(data,25) - 1.5 * IQR

outliers = data[(data > upper_fense) | (data < lower_fense)]
```

두 방법으로 'distance' feature의 outlier를 탐지하였다. 총 45593개의 데이터중 z-score 방식에서는 414개, IQR 방식에서는 431개의 outlier가 탐지되었다. 실행결과 IQR 방식에서 모델의

성능이 더 높다는 것을 확인하였고, IQR 방식을 채택하였다. 아래 사진은 각각 원래 데이터, outlier를 제거한 데이터의 분포를 나타낸 것이다.



outlier로 탐지된 값들은 모두 nan값으로 변환된다. 따라서 이후 모델 피팅시 nan값으로 인한 에러를 방지하기 위해 nan값을 가진 행들을 모두 제거하였다.

```
import pandas as pd
df = train
result = df.dropna(axis=0)
train = result
```

2) Regression

a. regression model 선택

multiple linear regression, decision tree regression model 중 하나를 선택한다.

다수의 범주형 데이터를 모두 고려하기 위하여 label encoding을 진행하였기 때문에, 선형성을 이용한 모델이 적합하지 않을 것이며, 범주형 데이터를 활용할 수 있는 decision tree의 효율성 우위를 예측했다. 이를 검증하기 위해, 실제로 두 가지 모델을 모두 사용한 뒤 결정계수(r squared score)를 기준으로 삼아 model의 정확성을 평가했다. multiple linear regression에 해당하는 모델은 lasso model, decision tree regression에 대해서는 XGBoosting model을 사용한다.

XGBoosting기법은 boosting의 매 단계에서 loss function의 negative gradient를 계산한다. 이것은 곧 loss function이 줄어드는 방향이기에, 이 방향에 새로운 model을 fit하는 과정을 거친다. 이것을 병렬적으로 진행해 속도가 빠르고, 자체적으로 과적합을 방지하는 기능이 있어 신뢰도가 높다.

boosting은 여러 decision tree의 learning error에 가중치를 두고, 합쳐가며 모델을 강화시키는 ensemble기법 중 하나이다.

a-1. 모델 비교 평가

사용하는 데이터(train.csv, test.csv)에 대해 앞서 기술한 전처리 과정을 적용한다. 이후 training set에 대하여 두 가지의 모델(multiple linear regression 기반 - **lasso model**; decision tree 기반 - **XGB model**)을 fitting한다. 학습을 마친 두 가지의 모델의 training set, validation set, test set에 대해 결과값(배달 시간) 예측을 진행하고, 각 모델의 결정계수(r squared score)를 측정한다.

결정계수를 통해 직관적인 성능 차이를 확인한 뒤, 보다 정밀한 모델 평가를 위해 test set에 대해 추가로 RMSE, MAPE를 측정한다.

평가 지표에 대한 이론적 배경은 다음과 같다.

- 결정계수 : MAPE, RMSE 등과 같은 지표들과 달리 상대적인 성능이 어느정도인지 직관적으로 판단할 수 있기에, 결정계수를 주 성능 비교 지표로 채택하였다. 0부터 1사이의 값을 가지며, 1에 가까울수록 독립변수가 종속변수에 대해 설명력이 높다.
- RMSE : 0과 1 사이의 값을 가진다. MSE처럼, 0에 가까울수록 error가 적다. outlier를 잘 다루는 지표이다.
- MAPE : 0과 1 사이의 값을 가진다. MAPE가 5%인 경우 실제 값과 예측 값의 차이가 5% 난다고 표현할 수 있다.

측정 결과는 다음과 같다.

	r2 (train)	r2 (valid)	r2 (test)	rmsle (test)	mape (test)
LASSO MODEL	0.41709	0.42237	0.51443	0.2235	0.19842
XGB MODEL	0.77274	0.77465	0.89937	0.10771	0.06802

모든 지표에서 LASSO model보다 XGB MODEL이 더 적합함이 확인되었다. 또한 validation set, test set에 대하여 약 0.77, 0.89이라는 높은 결정계수를 띠므로 XGB가 적합한 regression model이라고 확정짓고 이를 채택하였다.

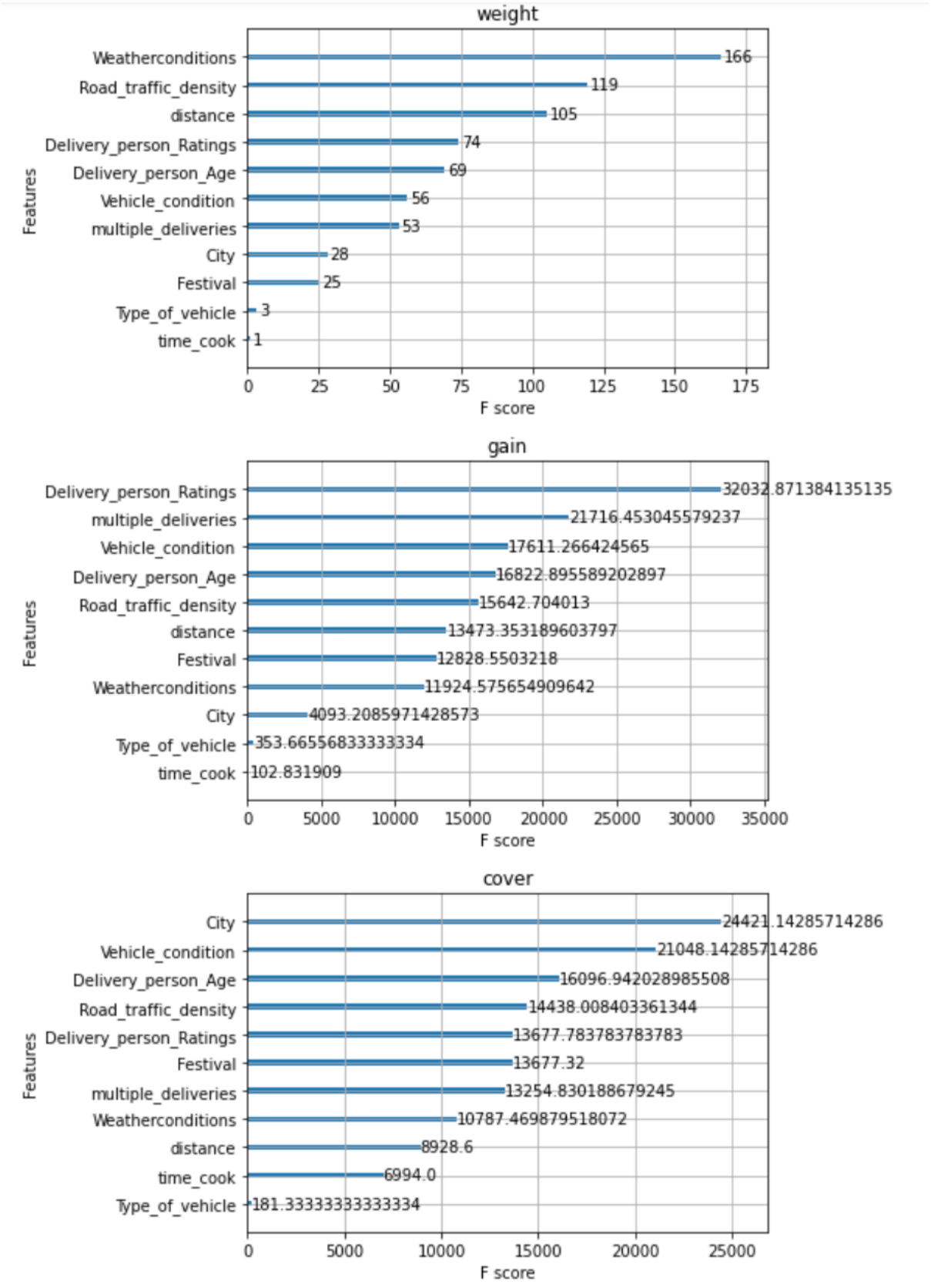
b. 성능 개선

1. feature selection

XGBoost에서 제공하는 feature의 중요도 타입은 weight, cover, gain 세가지가 있다. 타입별로 각 feature들의 중요도 양상이 다르게 나타나기 때문에 세타입을 종합적으로 고려하여 feature들을 선택한다면 예측값의 정확도에 유의미한 향상이 일어날 것이다.

첫째로 weight 타입은 해당 feature가 tree에서 노드의 분기에 사용된 횟수를 뜻한다. gain 타입은 노드가 특정 feature로 분기되었을때 얻는 성능 상의 이득으로 해당 feature을 사용하였을때 줄어드는 평균 손실을 의미한다. 마지막으로 cover 타입은 해당 feature로 분리된 데이터의 수를 나타낸다.

feature들의 importance 확인을 위해 타입별로 중요도를 나타내는 그래프를 생성하였다.



'Weatherconditions'는 weight타입에서는 높은 중요도를 보이지만 gain과 cover타입에서는 상대적으로 낮은 중요도를 보인다. 한편 'City'는 cover타입에서는 높은 중요도를, weight과 gain타입에서는 낮은 중요도를 보인다. 타입별로 feature들의 중요도가 상이한 결과를 보이는 만큼, 어떤 feature가 결과값에 긍정적 영향을 미치는 것인지는 판단할 수 없다. 하지만 대체적으로 feature들의 일관성이 유지되기 때문에 세가지 타입 모두에서 낮은 중요도를 보이는 feature인 'Type_of_order', 'Type_of_vehicle', 'Order_Date', 'time_cook'를 제거한다.

2. hyperparameter tuning

model이 자동으로 학습을 완료하도록 했던 parameter와 다르게, hyperparameter는 사용자가 직접 모델에 설정하여 최적의 모델을 구현하도록 하는 변수이다. 그 예시로 learning rate, epoch, weight initialization 등이 있다. 하이퍼파라미터는 데이터 분석 결과로 얻어지는 값이 아니므로 절대적인 최적값은 존재하지 않는다는 특징이 있다.

저번 단계의 모델 구현은 이러한 hyperparameter를 default 값으로 유지한 채(임의로 설정하지 않은 채) 모델 학습을 진행했으나, 이를 최적화하여 모델을 업그레이드하고, 예측 성능을 한 단계 향상시키는 것[2]이 hyperparameter tuning의 목표이다.

hyperparameter tuning을 구현하는 기법은 일반적으로 'Manual Search', 'Random Search', 'Grid Search', 'Bayesian Optimization' 4가지로 구분할 수 있다. 본 연구 단계에서는 xgb regression시 일반적으로 사용하는 grid search 방식을 채택하여 hyperparameter tuning을 진행하겠다. GridSearchCV는 tuning을 원하는 hyperparameter 범위를 사용자가 인자로 넘겨주면, 범위 전체에 대한 모든 조합을 전부 구성하여 최적의 파라미터를 찾는다.

범위 내 모든 조합을 고려한다는 점에서 시간적 이점이 있는 방식은 아니지만, 본 연구에서 사용하는 데이터의 크기를 고려해볼 때, 충분히 적용 가능한 방법이라 판단해 해당 방법을 채택하였다.

2-1. GridSearchCV 구현

parameters 변수에 최적의 값을 도출하고자 하는 hyperparameter 후보 값을 입력한다. learning 과정에서의 error 측정 방식은 tuning 이전과 동일하게 squared-error를 사용하였다.

```
xgb1 = XGBRegressor()
parameters = {
'nthread':[4], 'objective':['reg:squarederror'], 'learning_rate': [.03, 0.05, .07], 'max_depth': [5, 6, 7],
'min_child_weight': [4], 'silent': [1], 'subsample': [0.7], 'colsample_bytree': [0.7], 'n_estimators': [500]
}
```

[2] hyperparameter tuning을 통해 향상될 r2 score의 기대값 계산이 어렵기에, 구체적인 목표 수치는 설정하지 않음

이후 GridSearchCV 함수를 사용해, 위 파라미터 후보 값들 사이에서 최적의 조합을 탐색하도록 학습 세팅을 완료한다. 이후 전처리가 완료된 training data를 통해 model fitting을 진행한다.

```
xgb_grid = GridSearchCV(
xgb1, parameters, cv = 2, n_jobs = 5, verbose=True
)
xgb_grid.fit(x_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

3. 결과와 해석

```
#r-squared score; training set
tuned_pred = xgb_grid.predict(x_train)
score = r2_score(y_train,tuned_pred)
print('tuned : r2score(training) = ', score)

#r-squared score; validation set
tuned_pred = xgb_grid.predict(x_valid)
score = r2_score(y_valid,tuned_pred)
print('tuned : r2score(validation) = ', score)

#r-squared score; test set
tuned_pred = xgb_grid.predict(c_test)
score = r2_score(real_time,tuned_pred)
print('tuned : r2score(test) = ', score)
```

```
tuned : r2score(training) =  0.8572711537498117
tuned : r2score(validation) =  0.8271304103846366
tuned : r2score(test) =  0.9381385324351174
```

```
#root mean squared log error; test set
rmsle = mean_squared_log_error(real_time, tuned_pred) ** 0.5
print('tuned : rmsle(test) = ', rmsle)

#mean absolute percentage error; test set
mape = mean_absolute_percentage_error(real_time, tuned_pred)
print('tuned : mape(test) = ', mape)
```

```
tuned : rmsle(test) =  0.08180783292999873
```

tuned : mape(test) = 0.060307615015702 38

	r2 (train)	r2 (valid)	r2 (test)	rmsle (test)	mape (test)
hyperparameter tuning 이전	0.77974	0.78011	0.90195	0.10606	0.08303
hyperparameter tuning 이후	0.86727	0.82713	0.93813	0.08180	0.06030

각각의 data set에 대하여

결정계수 (r2 score)가 유의미하게 상승하였다.

두 가지 error가 유의미하게 감소하였다.

4. 본 결과를 바탕으로 향후 가능한 추가 연구 개발 방향

1) 앞서 새로 도입한 feature 'distance' 는 위도 경도 좌표에 헤버사인 공식을 적용해 계산한 유클리드 기하상의 최단 거리이다. 허나 실제 배달 경로는 좌표 간의 직선 거리가 아니다. 실생활과 가까운 결과를 산출하기 위해선 택시 구조에서의 비유클리드 경로를 반영하는 거리 함수의 도입이 필요하며 변경 시 보다 개선된 결과를 도출할 수 있을 것이다.

2) 배달 외식 서비스의 품질은 주문, 음식, 배달 앱, 외식업체, 배달 직원 등 다양한 요인의 영향을 받으며 이는 고객의 지속적인 이용의도와 만족도에 직접적인 영향력을 미친다. 본 연구는 그 중 '신속성'의 품질 향상을 위해 일부 요인들을 학습해 배달 시간 예측 모델을 구성하였다. 주요인들의 가중치를 분석해 우선적으로 영향을 미치는 feature에 대한 정보 역시 얻을 수 있었으며, feature의 조정으로 배달 시간 최소화에 대한 솔루션을 제공할 수 있다. 배달주문 방식에 있어서 상세한 정보 제공과 함께 고객들이 어떤 정보를 통해 음식 구매를 결정하는지에 대한 데이터를 모아 연구를 지속 개선한다면 보다 정확한 인사이트를 도출할 수 있을 것이다.

본 연구의 결과는 기존 배달비의 세부 요인 (기본료, 할인, 면제 혜택) 관련 배달 정책과 배달 앱 개선 방향에 대해 배달 시간 예측과 관련한 가이드를 제시할 수 있으며 나아가 배달 정보 서비스품질 향상에도 이바지할 것으로 기대가 된다.