통계학과 2019150419 기다연
통계적 머신러닝 최종 보고서 (자유주제: 1~6, 지정과제: 7)

1. 프로젝트 개요 및 목적

본 프로젝트는 다양한 통계적 머신러닝 모형을 이용해 의류 산업의 생산력을 예측하고자 한다. 이를 위해 먼저 자료의 특성을 파악하고, 결측치 대체와 같은 데이터 전처리를 거쳐 분석하기 용이한 형태로 변경해주었다. 각 모형에 데이터를 적합해 학습시켰으며, 이후 실제 산업현장과 실무에 배치(deployment)하기 위한 모형의 자동화를 진행했다.
본 프로젝트의 목적은 머신러닝 모형 중 특히 정형 데이터의 자료분석에서 뛰어난 성능을 보여주는 gradient boosting 계열 모형인 CatBoost, XGBoost, 그리고 LightGBM의 분석과정을 보여주는 것이다. 세 모형의 결과를 비교분석해 최종적으로 가장 성능이 높은 모델에 대해서는 모형의 자동화를 이뤄냈다.

2. 데이터셋 소개

본 프로젝트에서 사용되는 데이터셋은 의류 산업의 생산력 데이터셋이다. 출처는 UCI Machine Learning Repository 이며 수동으로 수집된 다음, 업계 전문가에 의해서 검증된 데이터셋이다.
현대 시대의 주요 산업 중 하나인 의류 산업은 수작업을 많이 요구하는 노동집약적인 산업이다. 의류 제품에 대한 수요를 충족시키는 것은 대부분 의류 제조 회사 직원들의 생산 및 배송 성과에 달려있다. 이 때문에 의류 산업의 의사결정권자들 사이에서는 공장에서 일하는 팀의 생산력 성과를 추적, 분석 및 예측하는 것이 매우 중요하다. 이 데이터셋은 13개의 특성변수를 이용해 의류 산업 내 생산력(actual_productivity)를 예측할 수 있도록 구성된 자료이다.

각 변수는 다음과 같은 의미를 갖는다:
1. quarter: 분기
2. department: 부서
3. day: 요일
4. team: 소속된 팀 번호
5. targeted_productivity: 매일 팀별로 설정한 목표 생산력 (0-1 사이 값)
6. smv: 각 작업에 할당된 시간 (분)
7. wip: 작업이 완료되지 않은 항목의 수
8. over_time: 팀별 초과 근무 시간 (분)
9. incentive: 재정적 인센티브 (BDT)
10. idle_time: 생산이 중단된 시간
11. idle_men: 생산이 중단된 근로자 수
12. no_of_style_change: 특정 제품의 스타일 변경 횟수
13. no_of_workers: 각 팀의 근로자 수
14. actual_productivity: 실제 생산력 (목적변수)

3. 데이터 EDA

EDA(exploratory data analysis) 과정은 주어진 데이터셋의 특성을 파악하기 위한 절차이다. 아래 프로그램으로 데이터셋의 첫 5개의 데이터를 살펴보았다.

```
import pandas as pd

df_prod = pd.read_csv('garments_worker_productivity.csv', engine='python')
df_prod.head()
```

	quarter	department	day	team	targeted_productivity	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers	actu
0	Quarter1	sweing	Thursday	8	0.80	26.16	1108.0	7080.0	98	0.0	0	0	59.0	
1	Quarter1	finishing	Thursday	1	0.75	3.94	NaN	960.0	0	0.0	0	0	8.0	
2	Quarter1	sweing	Thursday	11	0.80	11.41	968.0	3660.0	50	0.0	0	0	30.5	
3	Quarter1	sweing	Thursday	12	0.80	11.41	968.0	3660.0	50	0.0	0	0	30.5	
4	Quarter1	sweing	Thursday	6	0.80	25.90	1170.0	1920.0	50	0.0	0	0	56.0	

```
df_prod.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   quarter                 1197 non-null   object 
 1   department              1197 non-null   object 
 2   day                     1197 non-null   object 
 3   team                    1197 non-null   int64  
 4   targeted_productivity   1194 non-null   float64
 5   smv                     1196 non-null   float64
 6   wip                     691 non-null    float64
 7   over_time               1196 non-null   float64
 8   incentive               1197 non-null   int64  
 9   idle_time               1197 non-null   float64
 10  idle_men                1197 non-null   int64  
 11  no_of_style_change      1197 non-null   int64  
 12  no_of_workers           1193 non-null   float64
 13  actual_productivity     1197 non-null   float64
dtypes: float64(7), int64(4), object(3)
memory usage: 131.0+ KB
```

위 출력으로부터 df_prod 데이터는 1,197개의 관측치와 13개의 특성변수, 1개의 목적변수(actual_productivity)으로 구성되어 있음을 알 수 있다. 또한, 특성변수 quarter, department과 day는 범주형(object)이고, team, incentive, idle_men과 no_of_style_change는 정수형(int64), 그리고 목적변수를 포함한 나머지 변수들은 실수형(float64)이다.

아래 프로그램에서 각 특성변수별 결측치의 합과 총 결측치의 수를 계산했다. 전체자료 중 targeted_productivity에서 3개, no_of_workers에서 4개, 그리고 smv와 over_time 각각에서 1개, wip에서는 506개의 결측치가 발생했음을 알 수 있다.

```
print(df_prod.isna().sum())
print(df_prod.isna().sum().sum())
```

```
quarter                   0
department                0
day                       0
team                      0
targeted_productivity     3
smv                       1
wip                     506
over_time                 1
incentive                 0
idle_time                 0
idle_men                  0
no_of_style_change        0
no_of_workers             4
actual_productivity       0
dtype: int64
515
```

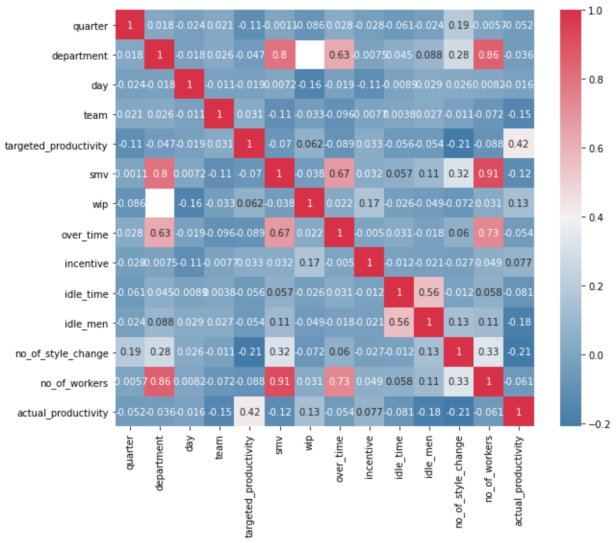
wip 변수는 전체 1,197개의 관측치 중 약 40%에 해당하는 506개의 결측치가 있으므로 무의미하다고 판단해 제거해주었다. 결측치를 가진 나머지 변수들은 모두 실수값을 가지므로 -999로 결측치를 대체해주었다.

```
df_prod['targeted_productivity'] = df_prod['targeted_productivity'].fillna(-999)
df_prod['smv'] = df_prod['smv'].fillna(-999)
df_prod['over_time'] = df_prod['over_time'].fillna(-999)
df_prod['no_of_workers'] = df_prod['no_of_workers'].fillna(-999)
del df_prod['wip']
```

범주형 변수인 quarter, department과 day 변수들을 인코딩해준 다음, 목적변수인 actual_productivity와의 상관관계를 오름차순으로 나열한 결과이다. 가장 높은 상관관계를 보인 변수는 targeted_productivity로 0.422의 정적 관계를 갖는다. 모든 변수들간의 상관관계를 시각화한 결과, smv와 no_of_workers간의 0.91의 높은 상관관계가 나타나는 것을 볼 수 있다. 이는 각 작업을 수행하는 데에 할당된 시간이 근로자의 수와 높은 상관성을 갖는다는 의미이기도 하다. 추가적으로 department와 no_of_workers는 0.86, 그리고 department과 smv 사이에는 0.80의 높은 상관관계도 보인다.

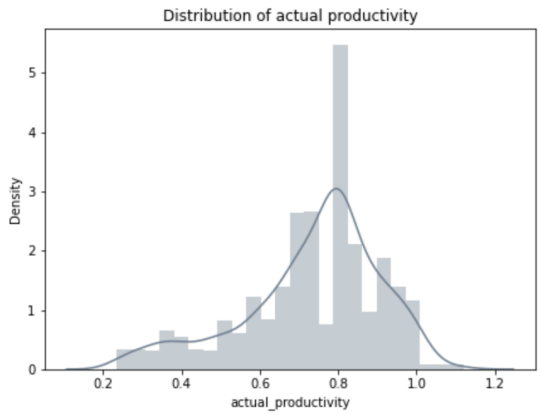
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

f, ax = plt.subplots(figsize=(10, 8))
corr = df_prod.corr()
sns.heatmap(corr,
            mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(240, 10, as_cmap=True),
            square=True, annot=True,  ax=ax)
```



목적변수인 actual_productivity 변수에 대한 분포를 시각화해본 결과, 대부분 0.8 부근에 포집되어 있음을 볼 수 있다.

```
sns.distplot(df_prod["actual_productivity"], color='slategray')
ax.set_title('Distribution of actual productivity')
```



4. 분석 모형 설정

Tabular 데이터의 자료분석에서 가장 뛰어난 성능을 보여주고 있는 분석기법은 XGBoost, LightGBM, 그리고 CatBoost이다. 3가지 모형은 모두 앙상블러닝의 일종으로 bagging과 boosting 중 boosting에 속한다. 대표적인 boosting은 Gradient Boosting이 있으며, 3개의 모형은 모두 Gradient Boosting의 개량형이라고 할 수 있다. Gradient Boosting을 기반으로 수렴속도와 성능을 대폭 개선할 수 있는 모형이기 때문에 본 프로젝트의 분석 모형으로 설정하였다. 전통적인 머신러닝 모형인 선형 회귀(Linear Regression)모형과 비선형 SVM(서포트벡터머신)을 이용해서도 분석을 진행했다. 또한, 본 프로젝트 데이터셋의 목적변수는 연속형 변수이기 때문에 분류보다는 회귀 모형으로 적합하기로 판단했다.

5. Pipeline 구성/모형 자동화

아래에서는 5개의 모형 (Linear Regression, SVM, CatBoost, XGBoost와 LightGBM)의 일련의 자료분석과정과 최종적인 모형 자동화를 위해 모델 pipeline을 구성하는 각 요소에 대해 더 심도있게 설명한다.

5.1 Linear Regression

LabelEncoder을 이용해 3개의 범주형 변수를 실수화해준 다음, 학습데이터와 시험데이터로 분할해주었다. 선형회귀모형에 적합한 결과, test RMSE값이 0.170으로 계산된 것을 알 수 있다.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as MSE

y_reg = df_prod.iloc[:, -1]
X_reg = df_prod.iloc[:, :-1]

le = LabelEncoder()
le.fit(X_reg.quarter.drop_duplicates())
X_reg.quarter = le.transform(X_reg.quarter)
le.fit(X_reg.department.drop_duplicates())
X_reg.department = le.transform(X_reg.department)
le.fit(X_reg.day.drop_duplicates())
X_reg.day = le.transform(X_reg.day)
X_train, X_test, y_train, y_test = train_test_split(X_reg, y_reg, random_state=4)

model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
pred_train = model_lr.predict(X_train)
pred_test = model_lr.predict(X_test)
print("train_RMSE:", np.sqrt(MSE(y_train, pred_train)))
print("test_RMSE:", np.sqrt(MSE(y_test, pred_test)))
```

```
train_RMSE: 0.16269957218769693
test_RMSE: 0.1700788633841052
```

5.2 SVM (Nonlinear)

5.1에서 실수화한 3개의 범주형 변수와 학습데이터와 시험데이터로 분할한 데이터에 대해 비선형 SVM (rbf 커널) 모형을 적합한 결과, RMSE이 0.172로 선형회귀모형(0.170)과 유사한 값을 보였다.

```
from sklearn.svm import SVR

svr = SVR(kernel='rbf', gamma='scale')
svr.fit(X_train,y_train)
y_pred_svr=svr.predict(X_test)
print("RMSE (rbf SVM):", np.sqrt(MSE(y_test, y_pred_svr)))
```

```
RMSE (rbf SVM): 0.1718551200153251
```

5.3 CatBoost

자료의 정제과정에서 결측치 처리는 모형에 관계없이 필수적이지만 자료의 수량화는 분석도구에 따라 필수적이거나 불필요한 경우가 있다. 이처럼 CatBoost 모형은 특성변수가 범주형인 경우에 one-hot encoding을 통한 범주형 자료의 수량화가 불필요하기에 수량화하는 과정은 이후 등장하는 XGBoost와 LightGBM 모형에 대해서만 적용해주었다.
먼저 CatBoostRegressor의 초모수 튜닝을 위해 맞춤형 함수를 정의하고, 모든 범주형 변수의 인덱스값을 cat_features에 지정해주었다.

```
kfold_c = KFold(n_splits=5, shuffle=True, random_state=2)

def grid_search_cat(params, reg=CatBoostRegressor(cat_features=[0,1,2])):
    grid_reg = GridSearchCV(reg, params, scoring='neg_mean_squared_error', cv=kfold_c)
    grid_reg.fit(X_cat, y_cat)
    best_params = grid_reg.best_params_
    print("Best params:", best_params)
    best_score = np.sqrt(-grid_reg.best_score_)
    print("Best score:", best_score)
```

위에서 정의한 grid_search_cat 함수를 이용해 초모수 튜닝을 적용했다.

```
grid_search_cat(params={'max_depth':[4],
                        'min_child_samples':[1, 2, 3],
                        'n_estimators':[500, 600]})
```

이 과정을 반복한 결과, 최종적인 초모수 튜닝 결과는 아래와 같다.

```
grid_search_cat(params={'max_depth':[3],
                'subsample':[0.8, 0.9, 1],
                'colsample_bylevel':[0.7, 0.8, 1],
                'n_estimators':[500, 600]})
```

```
Best params: {'colsample_bylevel': 0.8, 'max_depth': 3, 'n_estimators': 600, 'subsample': 1}
Best score: 0.12442547052382683
```

최적 초모수로 튜닝된 CatBoost 모형을 자동화하기 위해, 앞에서 논의한 결측치 처리과정을 맞춤형 클래스로 정의해주었다. 이를 위해 내장되어 있는 TransformerMixin이라는 클래스를 부모클래스로 해 맞춤형 클래스를 아래와 같은 형식으로 구성할 수 있다.

```
class NullValueImputer(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        for column in X.columns.tolist():
            X[column] = X[column].fillna(-999.0)
        return X
```

NullValueImputer과 CatBoostRegressor을 pipeline으로 연결해 자동으로 실행될 수 있도록 한 프로그램이다. cat_pipeline으로 자동화된 모형을 데이터에 적용해 성능을 측정하기 위해 학습데이터와 시험데이터로 분류해 적용한 결과, 성능 측도 RMSE는 0.122로 계산된 것을 알 수 있다.

```
from sklearn.pipeline import Pipeline

cat_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                         ('cat', CatBoostRegressor(cat_features=[0,1,2], subsample=1, max_depth=4, n_estimators=500))])
X_train_cat, X_test_cat, y_train_cat, y_test_cat = train_test_split(X_cat, y_cat, random_state=2)

cat_pipeline.fit(X_train_cat,y_train_cat)
y_pred_cat = cat_pipeline.predict(X_test_cat)
rmse_cat = MSE(y_test_cat, y_pred_cat)**0.5
print('RMSE (CatBoost):', rmse_cat)
```

```
RMSE (CatBoost): 0.12213578211510001
```

5.4 XGBoost

XGBoost나 LightGBM의 경우, 자동화하기 위해서는 범주형자료를 수량화해야 한다. 범주형 자료 (quarter, department, day)를 one-hot encoding하기 위해서 pipeline에서 OneHotEncoder 클래스를 정의해주었다. OneHotEncoder은 결과를 sparse 행렬인 1이 있는 위치만을 저장해주는 특징이 있다.

```
categorical_columns = df_prod.columns[df_prod.dtypes==object].tolist()

from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
hot = ohe.fit_transform(df_prod[categorical_columns])
hot_df = pd.DataFrame(hot.toarray())
```

기존 자료에서 범주형 변수가 아닌 특성변수는 표준형식이기에 hot_df의 형식인 csr(compressed sparse row) 형식과 이어붙이기 위해 형식을 통일화했다.

```
from scipy.sparse import csr_matrix
from scipy.sparse import hstack

cold_df = df_prod.select_dtypes(exclude=["object"])
cold = csr_matrix(cold_df)
final_sparse_matrix = hstack((hot, cold))
```

```
final_df = pd.DataFrame(final_sparse_matrix.toarray())
final_df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	8.0	0.80	26.16	7080.0	98.0	0.0	0.0	0.0	59.0	0.940725
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	1.0	0.75	3.94	960.0	0.0	0.0	0.0	0.0	8.0	0.886500
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	11.0	0.80	11.41	3660.0	50.0	0.0	0.0	0.0	30.5	0.800570
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	12.0	0.80	11.41	3660.0	50.0	0.0	0.0	0.0	30.5	0.800570
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	6.0	0.80	25.90	1920.0	50.0	0.0	0.0	0.0	56.0	0.800382

위 과정을 TransformerMixin 부모클래스를 이용해 SparseMatrix이란 맞춤형 클래스를 구성했다.

```
class SparseMatrix(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        categorical_columns= X.columns[X.dtypes==object].tolist()
        ohe = OneHotEncoder()
        hot = ohe.fit_transform(X[categorical_columns])
        cold_df = X.select_dtypes(exclude=["object"])
        cold = csr_matrix(cold_df)
        final_sparse_matrix = hstack((hot, cold))
        return final_sparse_matrix
```

early_stopping_rounds를 이용해 최적의 n_estimators를 구하기 위해 학습데이터와 검증데이터로 나눠준뒤 적용해본 결과, XGBRegressor의 n_estimators는 27으로, 이때의 RMSE는 0.137로 나타난다.

```
y_xgb = df_prod.iloc[:, -1]
X_xgb = df_prod.iloc[:, :-1]
X_train, X_test, y_train, y_test = train_test_split(X_xgb, y_xgb, random_state=4)

data_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                          ('sparse', SparseMatrix())])
X_train_transformed = data_pipeline.fit_transform(X_train)
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_train_transformed, y_train, random_state=2)

def n_estimators(model):
    eval_set = [(X_test_2, y_test_2)]
    eval_metric="rmse"
    model.fit(X_train_2, y_train_2, eval_metric=eval_metric, eval_set=eval_set, early_stopping_rounds=10)
    y_pred = model.predict(X_test_2)
    rmse = MSE(y_test_2, y_pred)**0.5
    return rmse

n_estimators(XGBRegressor(n_estimators=5000, missing=-999.0))
```

```
[27]    validation_0-rmse:0.13816

0.1366314233097678
```

초모수 튜닝을 위해 CatBoost에서와 같이 grid_search를 진행해주었다. 이를 반복적으로 적용한 결과, 최종적으로 튜닝한 XGBRegressor의 초모수는 다음과 같다.

```
kfold = KFold(n_splits=5, shuffle=True, random_state=2)
def grid_search(params, reg=XGBRegressor(missing=-999.0)):
    grid_reg = GridSearchCV(reg, params, scoring='neg_mean_squared_error', cv=kfold)
    grid_reg.fit(X_train_transformed, y_train)
    best_params = grid_reg.best_params_
    print("Best params:", best_params)
    best_score = np.sqrt(-grid_reg.best_score_)
    print("Best score:", best_score)
```

```
grid_search(params={'max_depth':[1],
                    'min_child_weight':[1, 2, 3],
                    'subsample':[0.8, 0.9, 1],
                    'colsample_bytree':[0.9],
                    'colsample_bylevel':[0.6, 0.7, 0.8, 0.9, 1],
                    'colsample_bynode':[0.6, 0.7, 0.8, 0.9, 1],
                    'n_estimators':[31]})
```

```
Best params: {'colsample_bylevel': 0.9, 'colsample_bynode': 0.6, 'colsample_bytree': 0.9, 'max_depth': 1, 'min_child_
weight': 1, 'n_estimators': 31, 'subsample': 0.8}
Best score: 0.139478342366748
```

XGBoost의 튜닝된 초모수는 평가 척도 RMSE 값이 0.135로 CatBoost (0.122)보다 낮은 성능인 것을 볼 수 있다.

```
X_test_transformed = data_pipeline.fit_transform(X_test)
model = XGBRegressor(n_estimators=31,
                     max_depth=1,
                     min_child_weight=3,
                     subsample=0.8,
                     colsample_bytree=0.9,
                     colsample_bylevel=0.9,
                     colsample_bynode=0.6,
                     missing=-999.0)
model.fit(X_train_transformed, y_train)
y_pred = model.predict(X_test_transformed)
rmse = MSE(y_pred, y_test)**0.5
print('RMSE (XGBoost):', rmse)
```

```
RMSE (XGBoost): 0.13501687268269447
```

5.5 LightGBM

LightGBM 모형 분석에서는 5.3의 XGBoost에서 n_estimators을 구하기 전까지의 과정이 동일하기에 생략한다. XGBoost와 동일하게 최적의 n_estimators를 구한 결과, 37으로 이때의 RMSE는 0.141로 나타났다.

```
from lightgbm import LGBMRegressor
n_estimators(LGBMRegressor(n_estimators=5000))
```

```
0.14098572511204166
```

초모수 튜닝을 위해 grid_search을 반복적으로 적용한 결과, 최종적으로 튜닝한 LightGBMRegressor의 초모수는 다음과 같다. 기존 0.141에서 0.128로 크게 발전한 것을 보아 모형 성능개선에 초모수 튜닝이 중요한 역할이었다고 추정된다.

```
def grid_search_lgb(params, reg=LGBMRegressor()):
    grid_reg = GridSearchCV(reg, params, scoring='neg_mean_squared_error', cv=kfold)
    grid_reg.fit(X_train_transformed, y_train)
    best_params = grid_reg.best_params_
    print("Best params:", best_params)
    best_score = np.sqrt(-grid_reg.best_score_)
    print("Best score:", best_score)

grid_search_lgb(params={'boosting_type':['gbdt','goss'],
                    'max_depth':[1, 2, 5, -1],
                    'min_child_weight':[1, 3, 5],
                    'subsample':[0.7, 0.8, 0.9, 1],
                    'colsample_bytree':[0.8, 0.9, 1],
                    'n_estimators':[37, 100]})
```

```
Best params: {'boosting_type': 'gbdt', 'colsample_bytree': 0.9, 'max_depth': 5, 'min_child_weight': 1, 'n_estimator
s': 37, 'subsample': 0.7}
Best score: 0.1284985313099552
```

최종적으로 조율된 초모수를 가진 LGBMRegressor의 성능 측정 결과, RMSE 값은 0.117으로 XGBoost(0.135)과 CatBoost(0.122)보다 우수한 것으로 나타났다.

```
model_lgb = LGBMRegressor(boosting_type='gbdt',
                  min_child_weight=1,
                  max_depth=5,
                  n_estimators=37,
                  subsample=0.7,
                  colsample_bytree=0.9)
model_lgb.fit(X_train_transformed, y_train)
y_pred = model_lgb.predict(X_test_transformed)
rmse = MSE(y_pred, y_test)**0.5
print('RMSE (LightGBM):', rmse)
```

```
RMSE (LightGBM): 0.117320925736586
```

5.6 모형 자동화

가장 우수한 성능을 보인 LightGBM을 이용해 앞에서 논의한 모든 절차들을 모아, 아래와 같이 최종적으로 자동화 모형을 구성해보았다.

```
lgb_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                  ('sparse', SparseMatrix()),
                  ('lgb', LGBMRegressor(boosting_type='gbdt',
                                  min_child_weight=1,
                                  max_depth=5,
                                  n_estimators=37,
                                  subsample=0.7,
                                  colsample_bytree=0.9))])
```

위 프로그램으로 객체화된 자동화 모형 lgb_pipeline을 최초의 자료를 입력해 적합하고, 새로운 데이터에 대한 결과를 예측해보았다.

```
y = df_prod.iloc[:, -1]
X = df_prod.iloc[:, :-1]
lgb_pipeline.fit(X, y)
```

```
Pipeline(steps=[('null_imputer',
                 <__main__.NullValueImputer object at 0x7fc5880a2cd0>),
                ('sparse', <__main__.SparseMatrix object at 0x7fc5880a21f0>),
                ('lgb',
                 LGBMRegressor(colsample_bytree=0.9, max_depth=5,
                               min_child_weight=1, n_estimators=37,
                               subsample=0.7))])
```

```
np.round(lgb_pipeline.predict(X_test), 3)
```

```
array([0.795, 0.659, 0.756, 0.773, 0.889, 0.695, 0.683, 0.554, 0.76 ,
       0.812, 0.642, 0.786, 0.688, 0.756, 0.678, 0.773, 0.874, 0.43 ,
       0.443, 0.736, 0.881, 0.644, 0.795, 0.575, 0.89 , 0.791, 0.697,
       0.776, 0.567, 0.8  , 0.587, 0.99 , 0.813, 0.68 , 0.6  , 0.683,
       0.725, 0.531, 0.683, 0.716, 0.825, 0.728, 0.762, 0.744, 0.674,
       0.786, 0.592, 0.626, 0.8  , 0.846, 0.697, 0.782, 0.493, 0.792,
       0.436, 0.761, 0.801, 0.799, 0.781, 0.76 , 0.795, 0.822, 0.79 ,
       0.743, 0.862, 0.836, 0.878, 0.778, 0.924, 0.793, 0.76 , 0.617,
       0.705, 0.447, 0.986, 0.693, 0.804, 0.851, 0.717, 0.501, 0.797,
       0.819, 0.994, 0.904, 0.831, 0.676, 0.89 , 0.532, 0.766, 0.874,
       0.829, 0.586, 0.837, 0.612, 0.422, 0.491, 0.541, 0.399, 0.877,
       0.689, 0.767, 0.744, 0.823, 0.85 , 0.696, 0.788, 0.8  , 0.86 ,
       0.656, 0.837, 0.893, 0.498, 0.829, 0.898, 0.866, 0.906, 0.867,
       0.736, 0.783, 0.771, 0.637, 0.852, 0.852, 0.886, 0.702, 0.732,
       0.861, 0.884, 0.565, 0.515, 0.575, 0.826, 0.754, 0.69 , 0.805,
       0.769, 0.75 , 0.702, 0.707, 0.859, 0.736, 0.794, 0.719, 0.624,
       0.757, 0.683, 0.709, 0.683, 0.742, 0.831, 0.84 , 0.782, 0.901,
       0.532, 0.632, 0.901, 0.703, 0.897, 0.809, 0.808, 0.694, 0.723,
       0.801, 0.826, 0.524, 0.894, 0.417, 0.799, 0.833, 0.811, 0.555,
       0.684, 0.831, 0.75 , 0.567, 0.635, 0.735, 0.554, 0.799, 0.848,
       0.829, 0.766, 0.808, 0.768, 0.801, 0.74 , 0.699, 0.535, 0.744,
       0.722, 0.982, 0.853, 0.804, 0.802, 0.485, 0.791, 0.779, 0.715,
       0.822, 0.936, 0.787, 0.591, 0.693, 0.805, 0.75 , 0.843, 0.882,
       0.817, 0.936, 0.738, 0.74 , 0.994, 0.805, 0.654, 0.797, 0.68 ,
       0.502, 0.673, 0.783, 0.813, 0.85 , 0.456, 0.795, 0.652, 0.726,
       0.661, 0.663, 0.804, 0.801, 0.453, 0.687, 0.612, 0.839, 0.822,
       0.577, 0.791, 0.798, 0.738, 0.806, 0.423, 0.83 , 0.54 , 0.651,
       0.585, 0.882, 0.759, 0.881, 0.563, 0.68 , 0.851, 0.829, 0.762,
       0.685, 0.885, 0.853, 0.762, 0.742, 0.76 , 0.43 , 0.771, 0.684,
       0.502, 0.808, 0.682, 0.874, 0.459, 0.516, 0.788, 0.794, 0.837,
       0.838, 0.883, 0.583, 0.807, 0.714, 0.85 , 0.748, 0.862, 0.829,
       0.814, 0.643, 0.886, 0.584, 0.554, 0.635, 0.745, 0.793, 0.532,
       0.459, 0.79 , 0.799, 0.647, 0.589, 0.799, 0.505, 0.776, 0.527,
       0.806, 0.608, 0.668])
```

6. 결과 분석

아래 표는 분석을 진행해본 5개 모형에 대한 평가 척도 RMSE (root mean square error, 평균 제곱근 오차)값을 비교해본 결과이다. LightGBM 모형의 RMSE 값이 0.117로 가장 높은 성능을 보였으며, 그 이후로는 CatBoost (0.122), XGBoost (0.135), 그리고 Linear Regression (0.170), 비선형 SVM (0.172) 순이다.

표 1. 모형 성능 RMSE 값 비교

Linear Regression	Nonlinear SVM	XGBoost	LightGBM	CatBoost
0.170	0.172	0.135	**0.117**	0.122

7. 부분의존도 분석

CatBoost 모형에 대해서 부분의존도를 시각화할 수 있는 맞춤형 partial_dependency 함수를 구성해볼 수 있다. partial_dependency 함수의 입력에서 model은 적용할 모형, X는 학습데이터를, features는 학습데이터 X에 있는 모든 특성변수의 수를, 그리고 f_id는 부분의존도를 구하고 싶은 특성변수의 index를 나타낸다. f_id에 지정된 특성변수가 실수형 특성변수인 경우에는 0.1% 분위수부터 99.5% 분위수까지 50등분을 하는 방법이 있다.

본 프로젝트에서는 범주형 특성변수 (quarter, department, days)도 포함하고 있으므로 이에 대해서도 고려해주었다. 아래 프로그램은 CatBoost 모형에서 one-hot encoding을 진행하지 않은 범주형 자료에 대해 부분의존도를 구하는 사용자 정의 함수이다. np.linspace 함수를 이용해 등분하는 것이 아닌, 범주형 자료에 대해서는 np.unique 함수를 사용했다.

```
import numpy as np

def partial_dependency_categorical(model, X, features, f_id):
    X_temp = X.copy()
    grid = np.unique(X_temp.iloc[:, f_id])
    y_pred = np.zeros(len(grid))

    for i, val in enumerate(grid):
        X_temp.iloc[:, f_id] = val
        y_pred[i] = model.predict(X_temp.iloc[:, :features]).mean()

    return grid, y_pred
```

5.3에서 정의한 cat_pipeline을 이용해서 데이터셋을 분할하고 CatBoost regressor 모델을 학습시켰다.

```
from catboost import CatBoostRegressor
from sklearn.base import TransformerMixin
from sklearn.pipeline import Pipeline

y_cat = df_prod.iloc[:, -1]
X_cat = df_prod.iloc[:, :-1]
cat_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                         ('cat', CatBoostRegressor(cat_features=[0,1,2],
                                                   subsample=1,
                                                   max_depth=4,
                                                   n_estimators=500))])

from sklearn.model_selection import train_test_split
X_train_cat, X_test_cat, y_train_cat, y_test_cat = train_test_split(X_cat, y_cat, random_state=2)

cat_pipeline.fit(X_train_cat, y_train_cat)
```

학습된 모델과 위에서 정의한 함수를 이용해 데이터셋에 있는 3개의 범주형 변수 (quarter, department, day) 중 1번째 범주형 변수인 quarter의 grid값과 예측된 부분의존도 값을 나타냈다.

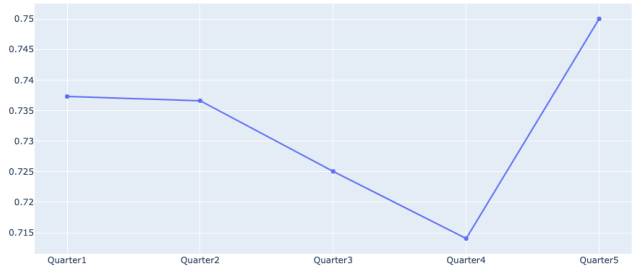
```
features = X_train_cat.shape[1]
f_id = 0
grid, y_pred = partial_dependency_categorical(cat_pipeline, X_train_cat, features, f_id=f_id)
print(grid)
print(y_pred)
```

```
['Quarter1' 'Quarter2' 'Quarter3' 'Quarter4' 'Quarter5']
[0.73730401 0.73661698 0.7250485  0.71405941 0.75000995]
```

Quarter5의 부분의존도가 0.750으로 가장 높으며, 그 뒤로 Quarter1(0.737), Quarter2(0.736), Quarter3(0.725), Quarter4(0.714) 순이다. 부분의존도는 목적변수와 특성변수의 관계를 보여주는 척도로써 다른 특성변수가 고정되어 있다는 가정 하에 관심있는 특성변수가 증가함에 따라 목적변수의 변화를 볼 수 있는 값이다. 즉, 목적변수인 생산력은 Quarter5 department에 근무하는 근로자들에 따라 가장 크게 변화한다는 의미이다. 시각화된 부분의존도는 다음과 같다.

```
import plotly.graph_objects as go

fig = go.Figure(data=go.Scatter(x=grid, y=y_pred))
fig.show()
```



나머지 2개의 범주형 변수인 department과 day에 대해서도 부분의존도를 구하고 시각화해주었다. (왼쪽: department, 오른쪽: day) Department(부서) 변수의 경우, finishing 부서에서 일하는 근로자들이 sewing 부서에서 일하는 근로자들에 비해 생산력에 더 큰 영향을 주게 된다는 의미이다. Day(요일) 변수의 경우, 일요일에 근로하는 근로자들의 자료가 생산력 예측에 더 큰 효과를 주게 되는 것을 볼 수 있다.

```
['finishing' 'sweing']
[0.74028928 0.71171328]
```


```
['Monday' 'Saturday' 'Sunday' 'Thursday' 'Tuesday' 'Wednesday']
[0.72462376 0.72607412 0.73981144 0.73711882 0.72347823 0.73635241]
```