Business Analytics

Cabbage Price Prediction

Team 4

18102087 Lee Yeongju

20102122 Jeong Hyoan

17102039 Kang Juho

Table of Contents

1. Brief Explanation about Topic

- **Topic: Predicting the price of cabbage, to help consumers make decisions**

- **Data**

Explanatory variable

- Daily Temperature
- Daily Rainfall
- Monthly Imports and Exports
- Monthly Consumer Price Index
- Yearly Production per area

Target variable

Daily Cabbage price

2. Additional Preprocessing – Deleting error data

- **Monthly Imports and Exports data (m_trade)**

→ We found *error data*

✓ Import amount > 0, but the imports = 0, because of unit '$1000'

```
In [12]: m_trade
```

Out[12]:

	년월	수출액($1000)	수출량(kg)	수입액($1000)	수입량(kg)
0	2022.09	33	12522	627	1081765
1	2022.08	34	14287	436	781725
2	2022.07	296	341857	33	74030
3	2022.06	968	1362508	0	0
4	2022.05	1889	2276613	0	0
...	...	...	...	...	...
124	2012.05	861	1317991	38	111135
125	2012.04	54	29408	524	865030
126	2012.03	99	71207	48	135850
127	2012.02	102	179261	0	0
128	2012.01	342	644146	0	78

```python
#수입액 결측치 제거
m_trade.rename(columns={'수입액($1000)':'수입액'}, inplace = True)
m_trade.rename(columns={'수입량(kg)':'수입량'}, inplace = True)

m_trade['수입액']=m_trade['수입액'].astype(int)
m_trade['수입량']=m_trade['수입량'].astype(int)

m_trade= m_trade.loc[(m_trade.수입액 != 0) | (m_trade.수입량 == 0)]
```

2. Additional Preprocessing – Missing values

- **Checking missing values**

```
In [37]: d2.isnull().sum()

Out[37]: Year            0
         Month           0
         Day             0
         평균배추가격           0
         지점              0
         평균기온(℃)          0
         최저기온(℃)          0
         최고기온(℃)          0
         강수량(mm)          0
         배추:면적 (ha)        0
         생산량 (톤)         212
         소비자물가지수            9
         수출액($1000)      189
         수출량(kg)        189
         수입액($1000)      189
         수입량(kg)        189
         dtype: int64
```

- **D2 data:** 2012.01~2022.11
- **Production data:** 2012 ~ 2011
- **Consumer Price Index data:** 2012.01 ~ 2022.10
- **Imports and exports data:** 2012.01~2022.09

→ Also, it has other missing values in the data

2. Additional Preprocessing – Deleting Missing value

- **Monthly Consumer Price Index** (m_price)
  - The dataset has missing value in the data
  - We delete the missing data

→ Deleting null data after 2022.09 : Do not need to handle 'Consumer Price Index data'

```
# 수출수입 데이터 결측치 제거
d2.dropna(subset=['수출액($1000)'], inplace=True)
d2.isnull().sum()
```

	Year	Month	Day	평균배추 가격	지점	평균기온(℃)	최저기온(℃)	최고기온(℃)	강수량 (mm)	배추:면적 (ha)	생산량 (톤)	소비자물가 지수	수출액 ($1000)	수출량 (kg)	수입액 ($1000)	수입량 (kg)
20	2012	2	1	3560	전국	-6.5	-10.6	-0.4	0.3	29524	2001642.0	91.588	102.0	179261.0	0.0	0.0
21	2012	2	2	3740	전국	-10.2	-13.9	-5.7	0.1	29524	2001642.0	91.588	102.0	179261.0	0.0	0.0
22	2012	2	3	3840	전국	-7.0	-13.7	-0.9	0.0	29524	2001642.0	91.588	102.0	179261.0	0.0	0.0
23	2012	2	6	4100	전국	1.2	-4.3	6.2	1.6	29524	2001642.0	91.588	102.0	179261.0	0.0	0.0
24	2012	2	7	4100	전국	-3.7	-8.5	3.0	0.0	29524	2001642.0	91.588	102.0	179261.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2642	2022	9	26	27760	전국	17.8	13.9	23.4	0.0	23853	NaN	108.930	33.0	12522.0	627.0	1081765.0
2643	2022	9	27	26720	전국	18.7	13.2	26.3	0.0	23853	NaN	108.930	33.0	12522.0	627.0	1081765.0
2644	2022	9	28	25860	전국	18.4	13.5	24.8	0.0	23853	NaN	108.930	33.0	12522.0	627.0	1081765.0
2645	2022	9	29	24640	전국	18.6	13.5	26.4	0.0	23853	NaN	108.930	33.0	12522.0	627.0	1081765.0
2646	2022	9	30	23740	전국	18.9	12.7	27.6	0.0	23853	NaN	108.930	33.0	12522.0	627.0	1081765.0

2. Additional Preprocessing – Changing Missing value

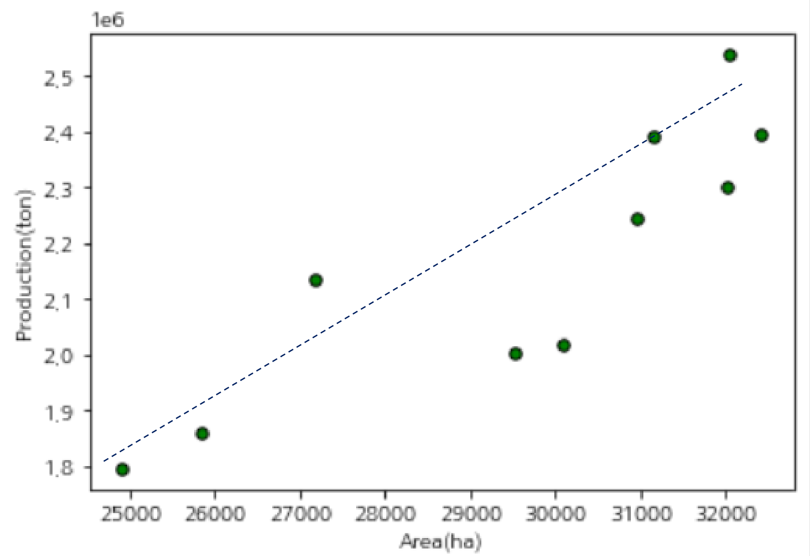
- **Yearly production per area data (y_produce)**

  → **We change missing value in y_produce to minimum value of production**

```
In [19]: y_produce
```

Out[19]:

	시점	배추:면적 (ha)	생산량 (톤)
0	2022	23853	NaN
1	2021	30085	2017507.0
2	2020	30949	2242640.0
3	2019	25837	1859705.0
4	2018	31143	2391946.0
5	2017	32416	2395686.0
6	2016	24902	1793391.0
7	2015	27174	2134976.0
8	2014	32027	2538804.0
9	2013	32020	2299251.0
10	2012	29524	2001642.0




```
#생산량 결측치 처리: min값으로 대체
production_min = d2['생산량 (톤)'].min()
d2.fillna(production_min, inplace=True)
d2.isnull().sum()
```

3. Model Selection

- **Linear Regression**

  - **Multi-linear regression models tend to be overfitted**

    → Generalization error is increased about new data

  - **By solving this problem, we used <u>Ridge</u> and <u>Lasso</u>**

  - **We will find the <u>best hyperparameters</u> for each model**

  - **And we will <u>compare the test scores</u> of each models applying the hyperparameters**

4. Feature Engineering

- **Changing the "Month" column to categorical '계절' (season) column**

  → Applying OneHotEncoder to the '계절'

```
d2['Month']=d2['Month'].astype(str)
d2['Month']=d2['Month'].str.replace('10', '가을')
d2['Month']=d2['Month'].str.replace('12', '겨울')
d2['Month']=d2['Month'].str.replace('1', '겨울')
d2['Month']=d2['Month'].str.replace('2', '겨울')
d2['Month']=d2['Month'].str.replace('3', '봄')
d2['Month']=d2['Month'].str.replace('4', '봄')
d2['Month']=d2['Month'].str.replace('5', '봄')
d2['Month']=d2['Month'].str.replace('6', '여름')
d2['Month']=d2['Month'].str.replace('7', '여름')
d2['Month']=d2['Month'].str.replace('8', '여름')
d2['Month']=d2['Month'].str.replace('9', '가을')
d2['Month']=d2['Month'].str.replace('11', '가을')
```

→ Code changing months to seasons

- **To put all numeric features on same scale, we used 'StandardScaler'**

- **In order to enrich a feature representation, we added 'Polynomial Features'**

5. Hyperparameter Tunning

- **Lasso**
  - **Find Alpha & value of degree in Polynomial**

```
kfold = KFold(n_splits=5, shuffle=True, random_state=0)

scaler = StandardScaler()

d_settings=[]
alpha_settings=[]
avg_r2=[]
avg_mse=[]

best_r2=0

for alpha in np.logspace(-1, 1, 50): # candidates for alpha

    print(alpha)

    r2_val = []
    mse_val = []
    maxdegree=5

    for d in range(1,maxdegree):

        print(d)

        alpha_settings.append(alpha)
        d_settings.append(d)

        poly = PolynomialFeatures(degree=d,include_bias=False)
```

 # Using KFold for cross-validation with data scaling and polynomial

 # We will select the hyperparameter having the highest R2

 # We set the 50 candidates of alpha in range from 0.1 to 10

 # We set the candidates of degree in range from 1 to 4

5. Hyperparameter Tunning

- **Lasso**
  - **Find Alpha & value of degree in Polynomial**

```
for train_idx, val_idx in kfold.split(x_trainval, y_trainval):

    x_train = x_trainval.iloc[train_idx]
    y_train = y_trainval.iloc[train_idx]
    x_val = x_trainval.iloc[val_idx]
    y_val = y_trainval.iloc[val_idx]

    x_train_cat = x_train[['계절']]
    x_train_num = x_train[['평균기온(℃)', '최저기온(℃)', '최고기온(℃)', '강수량(mm)', '생산면적 (ha)', '생산량 (톤)', '소

    x_val_cat = x_val[['계절']]
    x_val_num = x_val[['평균기온(℃)', '최저기온(℃)', '최고기온(℃)', '강수량(mm)', '생산면적 (ha)', '생산량 (톤)', '소비자

    # For x_train_cat and x_val_cat, apply onehotencoding
    ohe = OneHotEncoder(sparse=False)
    ohe.fit(x_train_cat)

    x_train_cat_ohe = ohe.transform(x_train_cat)
    x_val_cat_ohe = ohe.transform(x_val_cat)


    # For x_train_num and x_val_num, apply standardscaler
    scaler = StandardScaler()
    scaler.fit(x_train_num)

    x_train_num_scaled = scaler.transform(x_train_num)
    x_val_num_scaled = scaler.transform(x_val_num)

    poly.fit(x_train_num_scaled)
    x_train_poly = poly.transform(x_train_num_scaled)
    x_val_poly = poly.transform(x_val_num_scaled)
```

 # Dividing category variables and numerical variables for training and validation set

 # Transforming categorical features into numeric using OneHotEncoding

5. Hyperparameter Tunning

- **Lasso**

  - **Find Alpha & value of degree in Polynomial**

Concatenating into x_train_trans and x_val_trans

```
        # concatenate them agin into x_train and X_val.
        x_train_trans = np.concatenate([x_train_cat_ohe, x_train_poly], axis=1)
        x_val_trans = np.concatenate([x_val_cat_ohe, x_val_poly], axis=1)


        # training is performed with the Lasso set to the current alpha.
        lasso = Lasso(alpha = alpha, random_state=0, max_iter=10000)
        lasso.fit(x_train_trans, y_train)

        # get y_valid_hat with the trained model & store r2 score in scores_val
        y_val_hat= lasso.predict(x_val_trans)

        # store r2 score,mse
        r2_val.append(r2_score(y_val, y_val_hat))

    mean_r2 = np.mean(r2_val) # get the cross-validation score

    # When the mean_score is higher than current best score,best_score is updated and the hyperparameter at that time is saved
    if mean_r2 > best_r2:
        best_r2 = mean_r2
        best_parameters = {'alpha': alpha,'degree': d}

print("Best score on validation set: {:.7f}".format(best_r2))
print("Best hyperparameters: {}".format(best_parameters))
```

```
Best score on validation set: 0.7552897
Best hyperparameters: {'alpha': 10.0, 'degree': 4}
```

5. Hyperparameter Tunning

- ## Ridge

  - ### Find Alpha & value of degree in Polynomial

```
            x_train_num_scaled = scaler.transform(x_train_num)
            x_val_num_scaled = scaler.transform(x_val_num)

            poly.fit(x_train_num_scaled)
            x_train_poly = poly.transform(x_train_num_scaled)
            x_val_poly = poly.transform(x_val_num_scaled)


            # concatenate them agin into x_train and X_val.
            x_train_trans = np.concatenate([x_train_cat_ohe, x_train_poly], axis=1)
            x_val_trans = np.concatenate([x_val_cat_ohe, x_val_poly], axis=1)


            # training is performed with the Lasso set to the current alpha.
            ridge = Ridge (alpha = alpha, random_state=0, max_iter=10000)
            ridge.fit(x_train_trans, y_train)

            # get y_valid_hat with the trained model & store r2 score in scores_val
            y_val_hat= ridge.predict(x_val_trans)

            # store r2 score,mse
            r2_val.append(r2_score(y_val, y_val_hat))

        mean_r2 = np.mean(r2_val) # get the cross-validation score

        # When the mean_score is higher than current best score,best_score is updated and the hyperparameter at that time is saved
        if mean_r2 > best_r2:
            best_r2 = mean_r2
            best_parameters = {'alpha': alpha,'degree': d}

print("Best score on validation set: {:.7f}".format(best_r2))
print("Best hyperparameters: {}".format(best_parameters))
```

```
Best score on validation set: 0.7328941
Best hyperparameters: {'alpha': 7.543120063354615, 'degree': 3}
```

6. Model evaluation & Comparison

- **Lasso**

```
Best hyperparameters: {'alpha': 10.0, 'degree': 4}
```

```
x_trainval_cat = x_trainval[['계절']]
x_trainval_num = x_trainval[['평균기온(℃)', '최저기온(℃)', '최고기온(℃)', '강수량(mm)', '생산면적 (ha)', '생산량 (톤)', '소비자물

x_test_cat = x_test[['계절']]
x_test_num = x_test[['평균기온(℃)', '최저기온(℃)', '최고기온(℃)', '강수량(mm)', '생산면적 (ha)', '생산량 (톤)', '소비자물가지수'
```

```
ohe = OneHotEncoder(sparse=False)
ohe.fit(x_trainval_cat)

x_trainval_cat_ohe = ohe.transform(x_trainval_cat)
x_test_cat_ohe = ohe.transform(x_test_cat)
```

```
scaler = StandardScaler()

scaler.fit(x_trainval_num)

x_trainval_num_scaled = scaler.transform(x_trainval_num)
x_test_num_scaled = scaler.transform(x_test_num)

poly = PolynomialFeatures(degree=4,include_bias=False)

poly.fit(x_trainval_num_scaled)
x_trainval_poly = poly.transform(x_trainval_num_scaled)
x_test_poly = poly.transform(x_test_num_scaled)
```

```
In [24]: lasso = Lasso(alpha = 10, random_state=0, max_iter=10000)
lasso.fit(x_trainval_trans, y_trainval)

y_test_hat = lasso.predict(x_test_trans)

test_score = r2_score(y_test, y_test_hat)

print("Test-set score:", test_score)
```

```
Test-set score: 0.8709727352447184
```

6. Model evaluation & Comparison

- **Ridge**

```
Best hyperparameters: {'alpha': 7.543120063354615, 'degree': 3}
```

```
scaler = StandardScaler()

scaler.fit(x_trainval_num)

x_trainval_num_scaled = scaler.transform(x_trainval_num)
x_test_num_scaled = scaler.transform(x_test_num)

poly = PolynomialFeatures(degree=3,include_bias=False)

poly.fit(x_trainval_num_scaled)
x_trainval_poly = poly.transform(x_trainval_num_scaled)
x_test_poly = poly.transform(x_test_num_scaled)
```

```
In [29]: ridge = Ridge(alpha = 7.543120063354615, random_state=0, max_iter=10000)
ridge.fit(x_trainval_trans, y_trainval)

y_test_hat = ridge.predict(x_test_trans)

test_score = r2_score(y_test, y_test_hat)

print("Test-set score:", test_score)
```

```
Test-set score: 0.8784319654985169
```

✓ **Ridge has higher test score than Lasso**

✓ **We are able to get a Ridge model with quite an explanatory power**

THANK YOU