regression-76

June 7, 2024

0.1 # BIG MART PRICE ANALYSIS AND PREDICTION

USMAN JILLANI
Department of Computer Science, Superior University, Lahore, Pakistan

0.2 ##INTRODUCTION

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store. Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales We used Supervised machine learning and target value will be SALE The Aim to create model that can predict the sales per product for each store.Using machine learing models to help us to increase Big mart sales.

0.3 ##Dataset information

This dataset has 8523 values and 12 features in which 7 qualitative features and 5 quantitative features these columns names are below table. ####Qualitative Features Item_Identifier

1. Item_Fat_Content

2. Item_Type

3. Outlet_Identifier
4. Outlet_Size
5. Outlet_Location ####Quantitative Features
6. Item_Weight
7. Item_Visibility
8. Item_MRP
9. Outlet_Establishment_Year .

###LIBRARIES

[4]:
```
import pandas as pd #It is used for working with data sets
import numpy as np #It is used to perform a wide variety of mathematics
import matplotlib.pyplot as plt #It is used collection of functions
import seaborn as sns #It is used data visualization and graphical
import plotly.express as px # IT used for plots
%matplotlib inline
```

###READ DATASET

This dataset download from kaggle and datset link is https://www.kaggle.com/datasets/brijbhushannanda1979/big
sales-data

[5]:
```
data=pd.read_csv('/content/Train.csv')
```

[6]:
```
data.shape
```

[6]:
```
(8523, 12)
```

[7]:
```
data.describe()
```

[7]:
```
       Item_Weight  Item_Visibility     Item_MRP  Outlet_Establishment_Year  \
count  7060.000000      8523.000000  8523.000000                8523.000000
mean     12.857645         0.066132   140.992782                1997.831867
std       4.643456         0.051598    62.275067                   8.371760
min       4.555000         0.000000    31.290000                1985.000000
25%       8.773750         0.026989    93.826500                1987.000000
50%      12.600000         0.053931   143.012800                1999.000000
75%      16.850000         0.094585   185.643700                2004.000000
max      21.350000         0.328391   266.888400                2009.000000

       Item_Outlet_Sales
count        8523.000000
mean         2181.288914
std          1706.499616
min            33.290000
25%           834.247400
50%          1794.331000
75%          3101.296400
max         13086.964800
```

[8]:
```
data.duplicated().sum()
```

[8]:
```
0
```

[9]:
```
data.isnull().sum()
```

[9]:
```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
```

```
Outlet_Location_Type          0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Most Data scientist say if 30% Feauter have mising values if it is not a unique feature then we droped it . So in this dataset only Item_weight have 17.165317% null values and Outlet_Size have 28.276428% null values so we fill these empty entery. Item_weight is Quantative data so suitable Average is Mean so we apply it. Outlet_Size is Qualtative data so suitable Average is Mode so we apply it.

[10]:
```
data['Item_Weight'].fillna(data['Item_Weight'].mean(), inplace=True)
data['Outlet_Size'].fillna(data['Outlet_Size'].mode()[0], inplace=True)
```

[11]:
```
data.isnull().sum()
```

[11]:
```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

Remove Outliers

[12]:
```
def calculate_outliers(col):
    sorted(col)
    Q1, Q3 = col.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range
```

[13]:
```
data['Item_Visibility']=np.log1p(data['Item_Visibility'])
lower_limit, upper_limit = calculate_outliers(data['Item_Visibility']) #lower␣
 ↪and upper range
data['Item_Visibility'] = np.where(data['Item_Visibility'] < lower_limit,␣
 ↪lower_limit, data['Item_Visibility'])
data['Item_Visibility'] = np.where(data['Item_Visibility'] > upper_limit,␣
 ↪upper_limit, data['Item_Visibility'])
```

```
[14]: data['Item_Outlet_Sales']=np.log1p(data['Item_Outlet_Sales'])
lower_limit, upper_limit = calculate_outliers(data['Item_Outlet_Sales']) #lower␣
 ↪and upper range
data['Item_Outlet_Sales'] = np.where(data['Item_Outlet_Sales'] < lower_limit,␣
 ↪lower_limit, data['Item_Outlet_Sales'])
data['Item_Outlet_Sales'] = np.where(data['Item_Outlet_Sales'] > upper_limit,␣
 ↪upper_limit, data['Item_Outlet_Sales'])
```

Drop 2 features

```
[15]: data=data.drop(['Item_Identifier','Outlet_Identifier'],axis=1)
```

```
[16]: data.shape
```

```
[16]: (8523, 10)
```

```
[17]: data.head()
```

```
[17]:    Item_Weight Item_Fat_Content  Item_Visibility              Item_Type  \
0          9.30          Low Fat         0.015920                  Dairy
1          5.92          Regular         0.019095            Soft Drinks
2         17.50          Low Fat         0.016621                   Meat
3         19.20          Regular         0.000000  Fruits and Vegetables
4          8.93          Low Fat         0.000000              Household

   Item_MRP  Outlet_Establishment_Year Outlet_Size Outlet_Location_Type  \
0  249.8092                       1999      Medium               Tier 1
1   48.2692                       2009      Medium               Tier 3
2  141.6180                       1999      Medium               Tier 1
3  182.0950                       1998      Medium               Tier 3
4   53.8614                       1987        High               Tier 3

         Outlet_Type  Item_Outlet_Sales
0  Supermarket Type1           8.225808
1  Supermarket Type2           6.096776
2  Supermarket Type1           7.648868
3      Grocery Store           6.597664
4  Supermarket Type1           6.903451
```

Encode the Qualtative features

```
[18]: from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
data['Item_Type']= label_encoder.fit_transform(data['Item_Type'])
data['Outlet_Type']= label_encoder.fit_transform(data['Outlet_Type'])
data['Outlet_Location_Type']= label_encoder.
 ↪fit_transform(data['Outlet_Location_Type'])
```

```
data['Item_Fat_Content']= label_encoder.fit_transform(data['Item_Fat_Content'])
data['Outlet_Size']= label_encoder.fit_transform(data['Outlet_Size'])
```

[19]:
```
data.head()
```

[19]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0         9.30                 1         0.015920          4  249.8092
1         5.92                 2         0.019095         14   48.2692
2        17.50                 1         0.016621         10  141.6180
3        19.20                 2         0.000000          6  182.0950
4         8.93                 1         0.000000          9   53.8614

   Outlet_Establishment_Year  Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                       1999            1                     0            1
1                       2009            1                     2            2
2                       1999            1                     0            1
3                       1998            1                     2            0
4                       1987            0                     2            1

   Item_Outlet_Sales
0           8.225808
1           6.096776
2           7.648868
3           6.597664
4           6.903451
```

[20]:
```
data.shape
```

[20]:
```
(8523, 10)
```

[21]:
```
data.head()
```

[21]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0         9.30                 1         0.015920          4  249.8092
1         5.92                 2         0.019095         14   48.2692
2        17.50                 1         0.016621         10  141.6180
3        19.20                 2         0.000000          6  182.0950
4         8.93                 1         0.000000          9   53.8614

   Outlet_Establishment_Year  Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                       1999            1                     0            1
1                       2009            1                     2            2
2                       1999            1                     0            1
3                       1998            1                     2            0
4                       1987            0                     2            1

   Item_Outlet_Sales
```

```
0           8.225808
1           6.096776
2           7.648868
3           6.597664
4           6.903451
```

[22]:
```
data.shape
```

[22]:
```
(8523, 10)
```

Standardize the Quantative features

[23]:
```
from sklearn.preprocessing import StandardScaler
```

[24]:
```
standard_scaler = StandardScaler()
data['Item_Weight'] = standard_scaler.fit_transform(np.
 ↪array(data['Item_Weight']).reshape(-1,1))
data['Item_Visibility'] = standard_scaler.fit_transform(np.
 ↪array(data['Item_Visibility']).reshape(-1,1))
data['Item_Type']=standard_scaler.fit_transform(np.array(data['Item_Type']).
 ↪reshape(-1,1))
data['Item_MRP']=standard_scaler.fit_transform(np.array(data['Item_MRP']).
 ↪reshape(-1,1))
data['Outlet_Establishment_Year']=standard_scaler.fit_transform(np.
 ↪array(data['Outlet_Establishment_Year']).reshape(-1,1))
data['Item_Outlet_Sales']=standard_scaler.fit_transform(np.
 ↪array(data['Item_Outlet_Sales']).reshape(-1,1))
data.head()
```

[24]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0    -0.841872                 1        -1.022884  -0.766479  1.747454
1    -1.641706                 2        -0.952936   1.608963 -1.489023
2     1.098554                 1        -1.007433   0.658786  0.010040
3     1.500838                 2        -1.373631  -0.291391  0.660050
4    -0.929428                 1        -1.373631   0.421242 -1.399220

   Outlet_Establishment_Year  Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                   0.139541            1                     0            1
1                   1.334103            1                     2            2
2                   0.139541            1                     0            1
3                   0.020085            1                     2            0
4                  -1.293934            0                     2            1

   Item_Outlet_Sales
0           0.929362
1          -1.223928
2           0.345849
```

```
3           -0.717333
4           -0.408062
```

0.3.1 3.2 Regression

Splitting data into features(x) and target(y).

[25]:
```
x=data.drop(['Item_Outlet_Sales'],axis=1)
y=data['Item_Outlet_Sales']
```

[26]:
```
x.shape
```

[26]:
```
(8523, 9)
```

[27]:
```
y.shape
```

[27]:
```
(8523,)
```

[28]:
```
# prompt: split dataset import
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.
 ↪30,random_state=42)
```

[29]:
```
pip install lazypredict
```

```
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from lazypredict) (8.1.7)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from lazypredict) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from lazypredict) (2.0.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from lazypredict) (4.66.4)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from lazypredict) (1.4.2)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-
packages (from lazypredict) (4.1.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-
packages (from lazypredict) (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from lightgbm->lazypredict) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from lightgbm->lazypredict) (1.11.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
```

```
packages (from pandas->lazypredict) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->lazypredict) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->lazypredict) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas->lazypredict) (1.16.0)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

[30]:
```python
import lazypredict
from sklearn.model_selection import train_test_split
from lazypredict.Supervised import LazyRegressor
```

[31]:
```python
reg = LazyRegressor(verbose=0, ignore_warnings=False, custom_metric=None)
models,pred = reg.fit(x_train,x_test,y_train,y_test)
models
```

```
 21%|          | 9/42 [00:03<00:17,  1.88it/s]

GammaRegressor model failed to execute
Some value(s) of y are out of the valid range of the loss 'HalfGammaLoss'.

 74%|       | 31/42 [00:48<00:07,  1.44it/s]

PoissonRegressor model failed to execute
Some value(s) of y are out of the valid range of the loss 'HalfPoissonLoss'.
QuantileRegressor model failed to execute
Solver interior-point is not anymore available in SciPy >= 1.11.0.

 98%|      | 41/42 [00:57<00:00,  1.11it/s]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000803 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 782
[LightGBM] [Info] Number of data points in the train set: 5966, number of used
features: 9
[LightGBM] [Info] Start training from score 0.014419

100%|      | 42/42 [00:58<00:00,  1.39s/it]
```

[31]:

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
GradientBoostingRegressor	0.74	0.74	0.52	1.82
MLPRegressor	0.73	0.73	0.52	4.37
LGBMRegressor	0.73	0.73	0.52	0.31
HistGradientBoostingRegressor	0.73	0.73	0.52	3.51
NuSVR	0.73	0.73	0.53	2.29

SVR	0.72	0.72	0.53	4.41
RandomForestRegressor	0.71	0.71	0.55	3.06
ExtraTreesRegressor	0.70	0.70	0.56	1.74
AdaBoostRegressor	0.69	0.69	0.56	0.26
XGBRegressor	0.69	0.69	0.57	1.30
BaggingRegressor	0.68	0.68	0.57	0.39
KNeighborsRegressor	0.67	0.67	0.58	0.28
KernelRidge	0.59	0.59	0.65	6.77
LassoCV	0.59	0.59	0.65	0.37
OrthogonalMatchingPursuitCV	0.59	0.59	0.65	0.04
ElasticNetCV	0.59	0.59	0.65	0.39
LinearRegression	0.59	0.59	0.65	0.12
TransformedTargetRegressor	0.59	0.59	0.65	0.07
Lars	0.59	0.59	0.65	0.05
LarsCV	0.59	0.59	0.65	0.08
LassoLarsCV	0.59	0.59	0.65	0.12
LassoLarsIC	0.59	0.59	0.65	0.09
Ridge	0.59	0.59	0.65	0.02
RidgeCV	0.59	0.59	0.65	0.03
BayesianRidge	0.59	0.59	0.65	0.07
SGDRegressor	0.59	0.59	0.65	0.03
HuberRegressor	0.58	0.59	0.65	0.17
LinearSVR	0.58	0.58	0.65	0.17
DecisionTreeRegressor	0.46	0.46	0.74	0.15
ExtraTreeRegressor	0.45	0.46	0.75	0.13
TweedieRegressor	0.39	0.40	0.79	0.18
RANSACRegressor	0.33	0.33	0.83	0.24
OrthogonalMatchingPursuit	0.26	0.26	0.87	0.02
PassiveAggressiveRegressor	0.16	0.16	0.93	0.03
ElasticNet	-0.00	0.00	1.01	0.03
LassoLars	-0.01	-0.00	1.01	0.09
DummyRegressor	-0.01	-0.00	1.01	0.03
Lasso	-0.01	-0.00	1.01	0.07
GaussianProcessRegressor	-2118.15	-2110.68	46.45	24.78

[32]:
```
# prompt: Hyperparameter tuning of features GradientBoostingRegressor

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
gb = GradientBoostingRegressor()
parameters = {'subsample':[0.65,0.7,0.75],
              'n_estimators':[350,400,450],
              'learning_rate':[0.05,0.075,0.1]}
clf = GridSearchCV(gb,parameters,cv=3,scoring='neg_mean_squared_error')
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'learning_rate': 0.05, 'n_estimators': 350, 'subsample': 0.65}
```

[33]:
```
gb = GradientBoostingRegressor(subsample=0.7, n_estimators=400, learning_rate=0.
↪05)
```

[34]:
```
# prompt: GradientBoostingRegressor model run and print

gb.fit(x_train,y_train)
print("GradientBoostingRegressor model score:",gb.score(x_train,y_train))
```

```
GradientBoostingRegressor model score: 0.7690460650975708
```

0.4 [4] Conclusion :

###Gradient Boosting Regressor show use the accuracy 76.95%, and there RMSE is 0.52 ###After hyperparameters tuninng we get 76.98% Accuracy