Customer Spend Prediction - Using Decision Trees Regression

by A4Ayub Data Science Labs (http://www.a4ayub.me/ (http://www.a4ayub.me/))

Class Problem Statement

Build a model to predict customer spend based on the hour for shopping

Please take note that the illustrations in this notebook is NOT for results/accuracy but for explaining the various concepts

Data Description

This data is proprietory and cannot be shared to anyone who is NOT attending A4Ayub Data Science Labs.!

Each row in the dataset corresponds to one unique product in a basket (e.g. if there are three occurences of the same product in that basket, it will have one row for the product in that basket, with quantity equal to three)

The file has the below structure:

Column Name	Description	Type	Sample Values
shop_week	Identifies the week of the basket	Char	Format is YYYYWW where the first 4 characters identify the fiscal year and the other two characters identify the specific week within the year (e.g. 200735). Being the fiscal year, the first week doesn't start in January. (See time.csv file for start/end dates of each week)
shop_date	Date when shopping has been made. Date is specified in the yyyymmdd format	Char	20060413, 20060412
shop_weekday	Identifies the day of the week	Num	1=Sunday, 2=Monday, …, 7=Saturday
shop_hour	Hour slot of the shopping	Num	0=00:00-00:59, 1=01:00-01:59, …23=23:00-23:59
Quantity	Number of items of the same product bought in this basket	Num	Integer number
spend	Spend associated to the items bought	Num	Number with two decimal digits
prod_code	Product Code	Char	PRD0900001, PRD0900003
prod_code_10	Product Hierarchy Level 10 Code	Char	CL00072, CL00144
prod_code_20	Product Hierarchy Level 20 Code	Char	DEP00021, DEP00051
prod_code_30	Product Hierarchy Level 30 Code	Char	G00007, G00015
prod_code_40	Product Hierarchy Level 40 Code	Char	D00002, D00003

Column Name	Description	Type	Sample Values
cust_code	Customer Code	Char	CUST0000001624, CUST0000001912
cust_price_sensitivity	Customer's Price Sensitivity	Char	LA=Less Affluent, MM=Mid Market, UM=Up Market, XX=unclassified
cust_lifestage	Customer's Lifestage	Char	YA=Young Adults, OA=Older Adults, YF=Young Families, OF=Older Families, PE=Pensioners, OT=Other, XX=unclassified
basket_id	Basket ID. All items in a basket share the same basket_id value.	Num	994100100000020, 994100100000344
basket_size	Basket size	Char	L=Large, M=Medium, S=Small
basket_price_sensitivity	Basket price sensitivity	Char	LA=Less Affluent, MM=Mid Market, UM=Up Market, XX=unclassified
basket_type	Basket type	Char	Small Shop, Top Up, Full Shop, XX
basket_dominant_mission	Shopping dominant mission	Char	Fresh, Grocery, Mixed, Non Food, XX
store_code	Store Code	Char	STORE00001, STORE00002
store_format	Format of the Store	Char	LS, MS, SS, XLS
store_region	Region the store belongs to	Char	E02, W01, E01, N03

Workbench

Importing the required libraries

In [10]:

```
# Import the numpy and pandas package
import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# Import the warnings
import warnings

# Import statsmodels
import statsmodels.formula.api as smf

# Import RMSE
from statsmodels.tools.eval_measures import rmse

# Import Decison Tree Regressor
from sklearn.tree import DecisionTreeRegressor

# Import train test split
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_val_predict, trair

# Feature Scaling
from sklearn.preprocessing import StandardScaler

# Import the metrics
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score

# Import Pre-Processing
from sklearn import preprocessing


# configuration settings
%matplotlib inline
sns.set(color_codes=True)
warnings.filterwarnings('ignore') ## Surpress the warnings
sns.set_style('whitegrid')
sns.set_context('talk')
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (30, 10),
          'axes.labelsize': 'x-large',
          'axes.titlesize':'x-large',
          'xtick.labelsize':'x-large',
          'ytick.labelsize':'x-large'}

plt.rcParams.update(params)
```

Load the data into a dataframe

In [11]:

```
# load the data into a dataframe called supermarket_till_transactions_df
supermarket_till_transactions_df = pd.read_csv("../data/beginner/supermarket_till_transacti
```

In [12]:

```
# view the top five records
supermarket_till_transactions_df.head(5)
```

Out[12]:

	SHOP_WEEK	SHOP_DATE	SHOP_WEEKDAY	SHOP_HOUR	QUANTITY	SPEND	PROD_COD
0	200607	20060413	5	20	1	103	PRD090009
1	200607	20060412	4	19	1	28	PRD090035
2	200607	20060413	5	20	3	84	PRD090055
3	200607	20060412	4	19	1	221	PRD090164
4	200607	20060413	5	20	1	334	PRD090206

5 rows × 22 columns

In order to illustrate Support Vector Regression we just need two variables which are:

1. SHOP_HOUR
2. SPEND

In [13]:

```
supermarket_till_transactions_df = supermarket_till_transactions_df[["SHOP_HOUR","SPEND"]]
supermarket_till_transactions_df.head(5)
```

Out[13]:

	SHOP_HOUR	SPEND
0	20	103
1	19	28
2	20	84
3	19	221
4	20	334

In [14]:

```
X = supermarket_till_transactions_df.iloc[:,:-1].values
y = supermarket_till_transactions_df.iloc[:,-1].values
```

In [16]:

```
# Divide the dataset into training and testing sets
X, X_test, y, y_test = train_test_split(supermarket_till_transactions_df.iloc[:,0:-1],
                                        supermarket_till_transactions_df.iloc[:,-1],
                                        test_size=0.33,
                                        random_state=42)
#X.reset_index(inplace=True)
#y = y.reset_index()

#X_test.reset_index(inplace=True)
#y_test = y_test.reset_index()
```

In [17]:

```
dtm = DecisionTreeRegressor(max_depth=4,
                            min_samples_split=5,
                            max_leaf_nodes=10)

dtm.fit(X,y)
print("R-Squared on train dataset={}".format(dtm.score(X_test,y_test)))

dtm.fit(X_test,y_test)
print("R-Squared on test dataset={}".format(dtm.score(X_test,y_test)))
```

```
R-Squared on train dataset=-0.09551020565850865
R-Squaredon test dataset=0.15535223922482966
```

Hyper-parameter tuning with GridSearchCV

In [18]:

```
param_grid = {"criterion": ["mse", "mae"],
              "min_samples_split": [10, 20, 40],
              "max_depth": [2, 6, 8],
              "min_samples_leaf": [20, 40, 100],
              "max_leaf_nodes": [5, 20, 100],
              }

grid_cv_dtm = GridSearchCV(dtm, param_grid, cv=5)

grid_cv_dtm.fit(X,y)
```

Out[18]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeRegressor(criterion='mse', max_depth=4,
                                             max_features=None,
                                             max_leaf_nodes=10,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=5,
                                             min_weight_fraction_leaf=0.0,
                                             presort=False, random_state=Non
e,
                                             splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'criterion': ['mse', 'mae'], 'max_depth': [2, 6,
8],
                         'max_leaf_nodes': [5, 20, 100],
                         'min_samples_leaf': [20, 40, 100],
                         'min_samples_split': [10, 20, 40]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [19]:

```
print("R-Squared::{}".format(grid_cv_dtm.best_score_))
print("Best Hyperparameters::\n{}".format(grid_cv_dtm.best_params_))
```

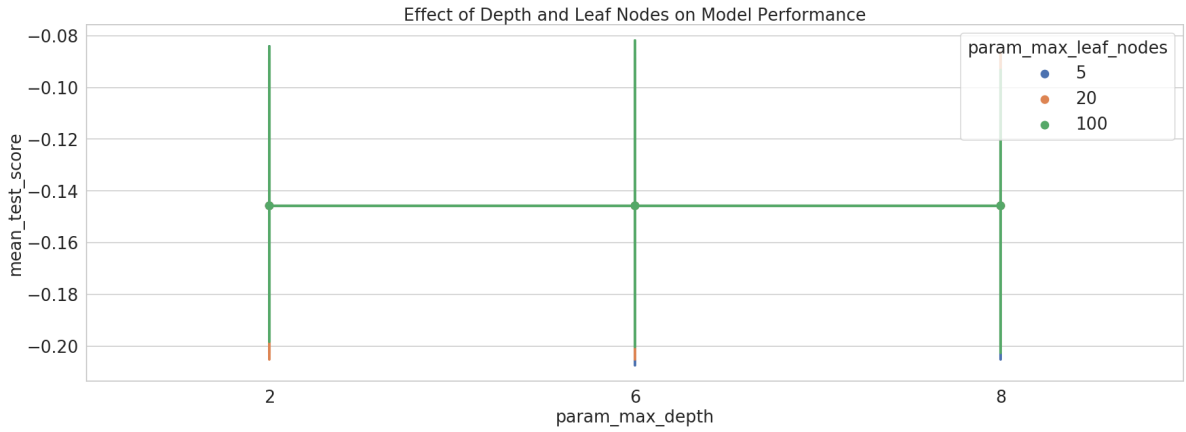
```
R-Squared::0.07377200633240072
Best Hyperparameters::
{'criterion': 'mae', 'max_depth': 2, 'max_leaf_nodes': 5, 'min_samples_lea
f': 20, 'min_samples_split': 10}
```

In [21]:

```
fig,ax = plt.subplots()
sns.pointplot(data=df[['mean_test_score',
                       'param_max_leaf_nodes',
                       'param_max_depth']],
              y='mean_test_score',x='param_max_depth',
              hue='param_max_leaf_nodes',ax=ax)
ax.set(title="Effect of Depth and Leaf Nodes on Model Performance")
```

Out[21]:

```
[Text(0.5, 1.0, 'Effect of Depth and Leaf Nodes on Model Performance')]
```



In [23]:

```
# Checking the training model scores
r2_scores = cross_val_score(grid_cv_dtm.best_estimator_, X, y, cv=10)
mse_scores = cross_val_score(grid_cv_dtm.best_estimator_, X, y, cv=10,scoring='neg_mean_squ

print("avg R-squared::{:.3f}".format(np.mean(r2_scores)))
print("MSE::{:.3f}".format(np.mean(mse_scores)))
```

```
avg R-squared::-0.840
MSE::-72856.535
```

In [25]:

```
best_dtm_model = grid_cv_dtm.best_estimator_

y_pred = best_dtm_model.predict(X_test)


r2_score = best_dtm_model.score(X_test,y_test)
print("R-squared:{:.3f}".format(r2_score))
print("MSE: %.2f" % metrics.mean_squared_error(y_test, y_pred))
```

```
R-squared:0.026
MSE: 36949.90
```

In [ ]:

```

```