Mega store

Team ID: SC_14

Team members:

	Name	University ID
1	محمد عبدالرؤف أحمد عبد الشافي	20191700557
2	فاطمة محمود عبد الحكم موسى بدر	20201700571
3	مصطفى محمد بيومي محمد	20201700837
4	مصطفى هانى محمد محروس	20201700843
5	عمر محمد عبدالرؤوف إبراهيم	20201700545
6	رحمة ايمن عطية عوض	20201700257

Report

First Splitting data:

- we have divided the data into two parts **: X(input features)** and **Y(target)** then we divided each of them into **train(80%)** and **test(20%),**and we have used **shuffle parameter** to shuffle the data randomly before splitting to avoid any order biases. Finally**, the random_state parameter** is set to 42, which ensures that the split will be the same every time the code is run.
- After split the entire data to train and test we call pre_processing(X_train) which explained below.

Second preprocessing data:

- We have implemented a preprocessing.py which contains three functions.
  - pre_processing(X)
    - ❖ This function takes the data frame of features and makes some changes to it and then returns it again.
    - ❖ the changes are represented in:
      - split columns of date {'Order Date' , 'Ship Date'} each to three columns day , month , year.
      - Split the column => 'Category tree' which data type is Dictionary to number of columns same as the number of unique Keys is all samples.
    - ❖ The output data frame contains 8 new columns, and we drop the old 3 columns so its net size increase by 5 columns.
  - numerical_Categorical(X:pd.dataframe())
    - ❖ This function takes a data frame of features and returns the numerical and categorical columns as two data frames.
  - date_split(X:pd.dataframe , cols)
    - ❖ This function takes two parameters:
      - X: data frame which contains a date column.
      - cols: List of string holds the names of columns which contain date.
    - ❖ split columns of date each to three columns day, month, and year.

- **Encoding:**
    - **After some experiments we decide to use The Ordinal Encoder from the categorical encoders library with two arguments:**
        - handle_unknown='value'   and   handle_missing='value'
        - We used handle_unknown argument to handle categories in the test set that has not seen in the training set.
        - We used handle_missing argument to handle missing values in the data

- Statistical Analysis :

computes descriptive statistics for the training dataset X_train and the target variable y_train, and saves the results as a pickle file. Here is a brief description of each part of the code:
X_train.mean(axis=0, skipna=False): This line computes the mean of each column in X_train, ignoring missing values.
X_train.median(axis=0, skipna=False): This line computes the median of each column in X_train, ignoring missing values.
X_train.mode(axis=0, numeric_only=False).loc[0, :]: This line computes the mode of each column in X_train, ignoring missing values. If multiple values have the same frequency, the first one encountered is returned. The numeric_only=False argument allows non-numeric columns to be included in the computation.
y_train.mean()[0]: This line computes the mean of the target variable y_train.
y_train.median()[0]: This line computes the median of the target variable y_train.
y_train.mode().loc[0, 'ReturnCategory']: This line computes the mode of the target variable y_train. The ReturnCategory column is specified explicitly.
pd.Series(): This line creates a new pandas series with the computed statistics.
statistics.columns = ['mean', 'median', 'mode']: This linerenames the columns of the statistics dataframe to 'mean', 'median', and 'mode'.
statistics = statistics.append(tar): This line appends the target variable statistics to the statistics dataframe.
pickle.dump(statistics, open("models/statistics2.pkl", "wb")): This line saves the statistics dataframe as a pickle file named "statistics2.pkl" in the "models" folder.
Overall, this code block is a good example of how to compute and save descriptive statistics for a dataset and target variable in Python using pandas and pickle. The resulting statistics can be used to better understand the distribution

of the data, and can help inform the choice of model and hyperparameters. It is important to note that the choice of statistics to compute will depend on the specific characteristics of the problem being solved.

- Feature selection: (Milestone 2)

After separating data into categorical and numerical we made feature selection on each one

Introduction: We do feature selection on a training dataset that contains both numerical and categorical features. Feature selection is an important step in machine learning, as it helps to improve the predictive performance of a model by selecting a subset of the most relevant features.

1. **SelectKBest(score_func=f_regression)** method from the scikit-learn library to select the top k features based on their F-scores, The F-score measures the linear dependence between the target variable and each feature in the numerical data input to find out the most important of columns, then The code first selects the top 4 most significant numerical features based on their F-score using the SelectKBest function from scikit-learn. The F-score measures the degree of linear dependency between a numerical feature and the target variable. The selected numerical features are then kept in the training dataset, and the rest are dropped.

2. **SelectKBest(score_func=chi2)** method from the scikit-learn library to select the top k features based on their F-scores, The F-score measures the linear dependence between the target variable and each feature in the categorical data input to find out the most important of columns, then the code selects the top 7 most significant categorical features using the chi-squared test (chi2). The chi-squared test measures the degree of association between a categorical feature and the target variable. The selected categorical features are kept in the training dataset, and the rest are dropped.

Third Classification models:

At first we select best parameters for some models using a function named best Parameters that takes several arguments and is used for hyper parameter tuning of a classification model. Here is a brief description of each part of the function:

Best Parameters that takes as input the following arguments:

X: The input features (independent variables).

Y: The target variable (dependent variable).

classifier: The type of classifier to use (e.g., DecisionTreeClassifier).

parameter1: The name of the first parameter to tune.

list1: A list of values to try for parameter1.

parameter2: The name of the second parameter to tune (optional).

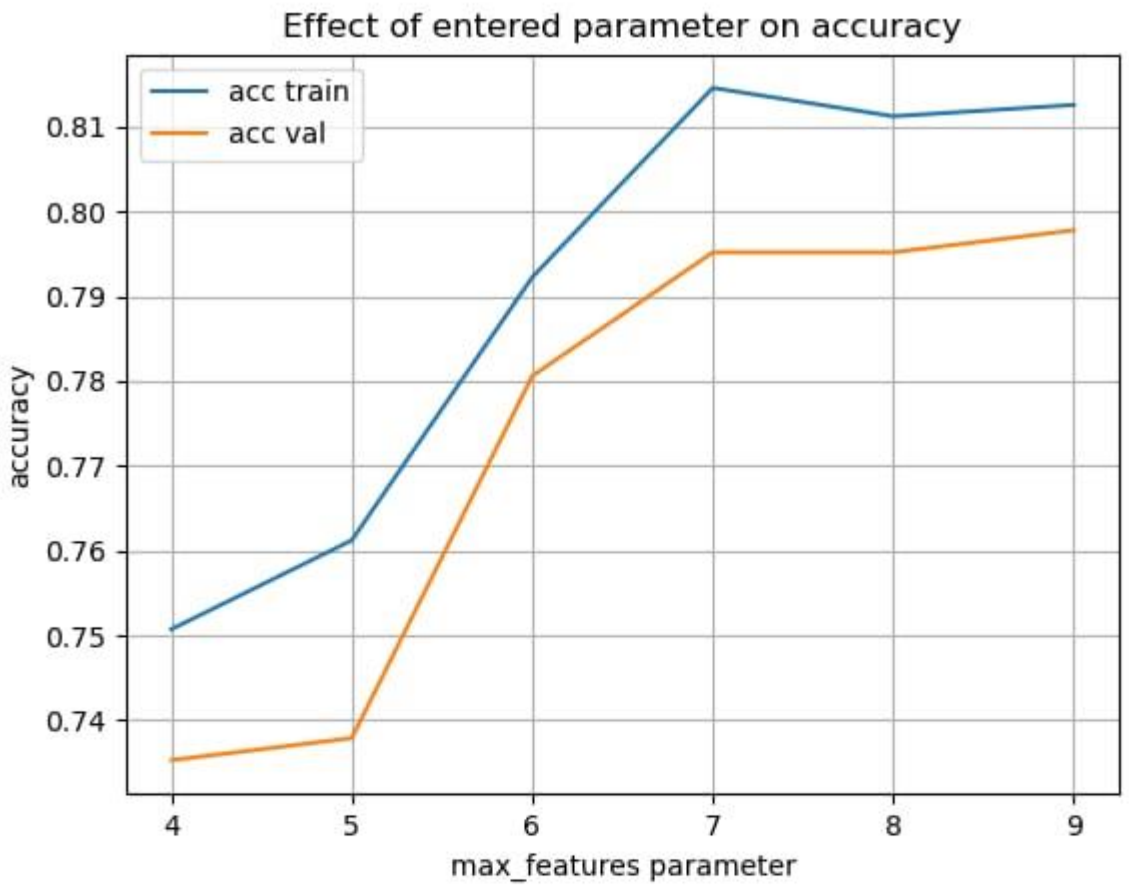
value: The value to use for parameter2 (optional).

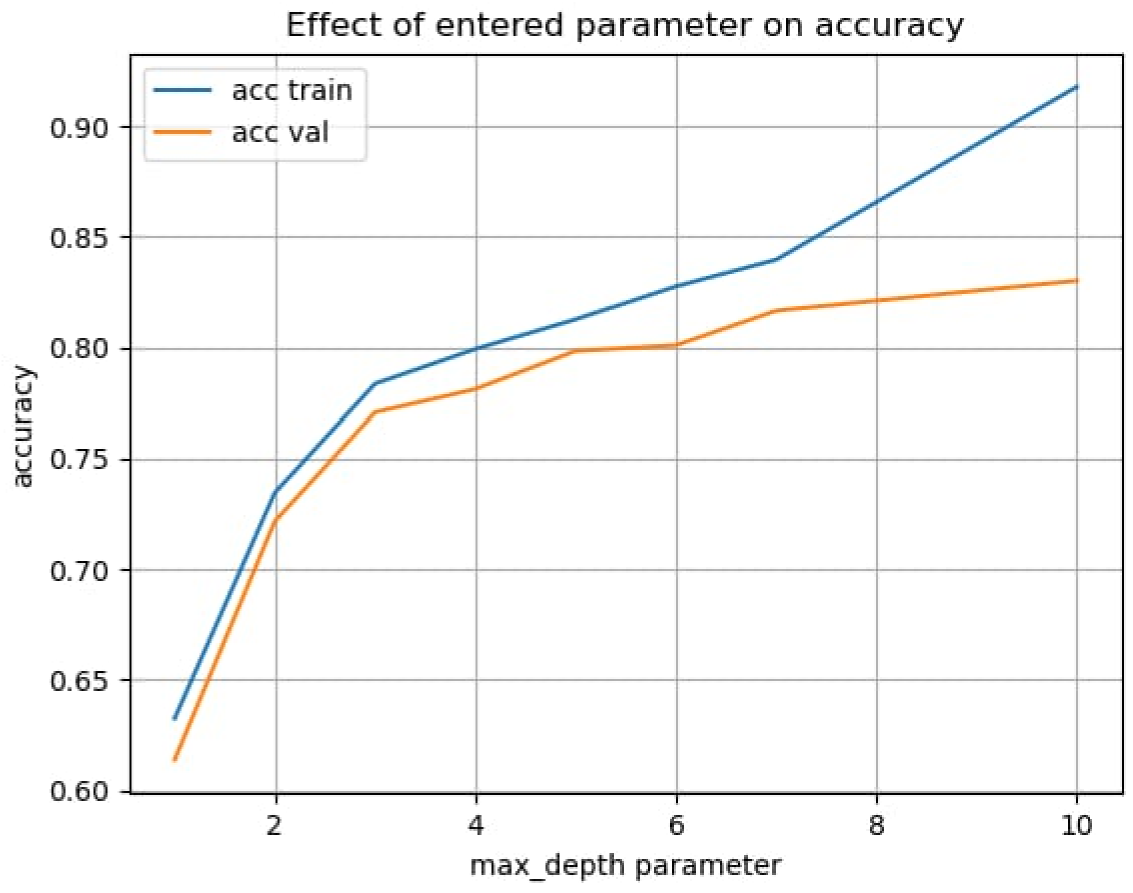
The function first splits the data into training and validation sets using the train_test_split function from scikit-learn. It then iterates over the values in list1, and for each value, trains a classifier with the specified parameters using the classifier argument. If parameter2 is specified, it is also passed to the classifier.

After training the model, the function computes the accuracy of the model on both the training and validation sets using the accuracy_score function from scikit-learn. The accuracy values for both the training and validation sets are then stored in separate lists.

Finally, the function plots the accuracy values for both the training and validation sets as a function of the values in list1. This allows the user to visualize the effect of changing parameter1 on the accuracy of the model.

Overall, this code is a good example of how to tune hyperparameters for a classifier using Python and scikit-learn. It allows the user to experiment with different values of the hyperparameters and see how they affect the model's accuracy. This can be a useful tool for finding the optimal hyperparameters for a given dataset and classifier.





Then we define several classification models and fits them to the training data. Here is a brief description of each part of the code:

model = LogisticRegression().fit(X_train, y_train): This line defines a logistic regression classifier and fits it to the training data X_train and y_train. Logistic regression is a linear model that is commonly used for binary classification problems.

model2 = BaggingClassifier(n_estimators=100, max_samples=6000).fit(X_train, y_train)fit(X_train, y_train): This line defines a bagging classifier and fits it to the training data. Bagging is an ensemble method that combines multiple models to improve their performance.

model3 = RandomForestClassifier(n_estimators=100, max_samples =6000).fit(X_train, y_train): This line defines a random forest classifier and fits it to the training data. Random forests are an extension of bagging that use decision trees as the base models.

model4 = svm.SVC(C=150).fit(X_train, y_train): This line defines a support vector machine (SVM) classifier with a regularization parameter C set to 150 and fits it to the training data. SVMs are a powerful class of linear and nonlinear models that can be used for both classification and regression problems.

model5 = DecisionTreeClassifier=>fit(X_train, y_train): This line defines a decision tree classifier with a maximum depth of 5 and maximum number of features to consider foreach split set to 6, and fits it to the training data. Decision trees are a popular class of models that can handle both numerical and categorical data, and are known for their interpretability.

Overall, this code block is a good example of how to define and fit several classification models in Python using scikit-learn. The resulting models can be used to make predictions on new data, and their performance can be compared to choose the best model for the given problem. It is worth noting that the choice of model and its hyperparameters will depend on the specific characteristics of the problem being solved.

- Accuracy result for each model:

Name of the model	train score	Test score	Train time In seconds	Test time In seconds
Logistic regression	0.605065666041	0.6047529	0.2642221450805664	0.003953218460083008
Bagging classifier	0.99124452782	0.8555347	1.9322521686553955	0.15163803100585938
Random forest classifier	1.0	0.85240775	0.7306480407714844	0.12655282020568848
SVM classifier	0.54002501563	0.54721701	1.8387360572814941	1.0384657382965088
Decision tree	0.81222639149	0.80050031	0.02195882797241211	0.003983974456787109

Total training time for models =4.8289268016815186 seconds

Total testing time for models = 1.3020336627960205 seconds

Test Script

Test Steps:

Step 1: Load the file

Step 2: Load selected features which has been saved in the training processes

Step 3: Split data into features columns(X) and target column(Y)

Step 4: deal with missing values : by filling NA with mean , median or mode handles missing values in the input features and target variable of a regression dataset. Here is a brief description of each part of the code:

skew_values = X_test_num.skew(): This line computes the skewness values of the numerical features.

for column in X_test_num.columns[X_test_num.isnull().any()]:: This loop iterates over the numerical features that have missing values.

skewness = skew_values[column]: This line computes the skewness of the current numerical feature.

if abs(skewness) < 0.5: # Assuming a skewness threshold of 0.5: This line checks if the absolute value of the skewness is less than a threshold of 0.5.

X[column].fillna(statistics.at[column, 'mean'], inplace=True): This line fills the missing values in the current numerical feature with its mean value.

X[column].fillna(statistics.at[column, 'median'], inplace=True): This line fills the missing values in the current numerical feature with its median value.

for col in X_test_cat.columns:: This loop iterates over the categorical features that have missing values.

X[col].fillna(statistics.at[col, 'mode']): This line fills the missing values in the current categorical feature with its mode value.

if abs(Y.skew()) <0.5: # Assuming a skewness threshold of 0.5: This line checks if the absolute value of the skewness of the target variable is less than a threshold of 0.5.

Y.fillna(statistics.at['Profit', 'mean'], inplace=True): This line fills the missing values in the target variable with its mean value.

Y.fillna(statistics.at['Profit', 'median'], inplace=True): This line fills the missing values in the target variable with its median value.

Step 5: Load encoders which has been saved in training process and Encode the categorical data

Step 6: Load the models and calculate score for each model

<u>**Conclusion**</u>

The results show that the bagging classifier and random forest classifier have the best performance on the test set, while the SVM model has the lowest performance.