decision-tree

July 8, 2024

[37]:
```
# Importing required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import plot_tree
```

[38]:
```
# Load the data
def load_data(file_path):
    try:
        data = pd.read_csv(file_path)
        print(f"Data loaded successfully. Shape: {data.shape}")
        return data
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
        return None
    except pd.errors.EmptyDataError:
        print(f"Error: The file at {file_path} is empty")
        return None
    except pd.errors.ParserError:
        print(f"Error: Unable to parse the file at {file_path}")
        return None

data = load_data('booking.csv')

if data is None:
    raise SystemExit("Data loading failed. Exiting the notebook.")
```

[39]:
```
# Display basic information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36285 entries, 0 to 36284
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Booking_ID              36285 non-null  object 
 1   number of adults        36285 non-null  int64  
 2   number of children      36285 non-null  int64  
 3   number of weekend nights  36285 non-null  int64  
 4   number of week nights   36285 non-null  int64  
 5   type of meal            36285 non-null  object 
 6   car parking space       36285 non-null  int64  
 7   room type               36285 non-null  object 
 8   lead time               36285 non-null  int64  
 9   market segment type     36285 non-null  object 
 10  repeated                36285 non-null  int64  
 11  P-C                     36285 non-null  int64  
 12  P-not-C                 36285 non-null  int64  
 13  average price           36285 non-null  float64
 14  special requests        36285 non-null  int64  
 15  date of reservation     36285 non-null  object 
 16  booking status          36285 non-null  object 
dtypes: float64(1), int64(10), object(6)
memory usage: 4.7+ MB
```

[40]:
```
# Check for missing values and display as a percentage
missing_percentages = (data.isnull().sum() / len(data)) * 100
print("Missing values (%):")
print(missing_percentages[missing_percentages > 0])
```

[40]:
```
Booking_ID                0
number of adults          0
number of children        0
number of weekend nights  0
number of week nights     0
type of meal              0
car parking space         0
room type                 0
lead time                 0
market segment type       0
repeated                  0
P-C                       0
P-not-C                   0
average price             0
special requests          0
date of reservation       0
booking status            0
```

```
dtype: int64
```

[41]:
```
# Display summary statistics for numerical columns
numerical_columns = data.select_dtypes(include=[np.number]).columns
data[numerical_columns].describe()
```

[41]:

	number of adults	number of children	number of weekend nights	number of week nights	car parking space	lead time	repeated	P-C	P-not-C	average price	special requests
count	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000
mean	1.844839	0.105360	0.810693	2.204602	0.030977	85.239851	0.025630	0.023343	0.153369	103.421636	0.619733
std	0.518813	0.402704	0.870590	1.410946	0.173258	85.938796	0.158032	0.368281	1.753931	35.086469	0.786262
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	1.000000	0.000000	17.000000	0.000000	0.000000	0.000000	80.300000	0.000000
50%	2.000000	0.000000	1.000000	2.000000	0.000000	57.000000	0.000000	0.000000	0.000000	99.450000	0.000000
75%	2.000000	0.000000	2.000000	3.000000	0.000000	126.000000	0.000000	0.000000	0.000000	120.000000	1.000000
max	4.000000	10.000000	7.000000	17.000000	1.000000	443.000000	1.000000	13.000000	58.000000	540.000000	5.000000

[42]:
```
# Check unique values and their counts in categorical columns
categorical_columns = data.select_dtypes(include=['object']).columns
for col in categorical_columns:
    print(f"\n{col}:")
    print(data[col].value_counts(normalize=True) * 100)
```

```
Booking_ID:
Booking_ID
INN00001    1
INN24194    1
```

```
INN24188    1
INN24189    1
INN24190    1
           ..
INN12090    1
INN12089    1
INN12088    1
INN12087    1
INN36286    1
Name: count, Length: 36285, dtype: int64

type of meal:
type of meal
Meal Plan 1     27842
Not Selected     5132
Meal Plan 2      3306
Meal Plan 3         5
Name: count, dtype: int64

room type:
room type
Room_Type 1    28138
Room_Type 4     6059
Room_Type 6      966
Room_Type 2      692
Room_Type 5      265
Room_Type 7      158
Room_Type 3        7
Name: count, dtype: int64

market segment type:
market segment type
Online           23221
Offline          10531
Corporate         2017
Complementary      391
Aviation           125
Name: count, dtype: int64

date of reservation:
date of reservation
10/13/2018    254
10/16/2017    236
6/15/2018     231
6/24/2018     213
9/18/2017     201
              …
10/2/2015       1
```

```
7/24/2017       1
9/13/2016       1
5/20/2017       1
7/21/2017       1
Name: count, Length: 553, dtype: int64

booking status:
booking status
Not_Canceled    24396
Canceled        11889
Name: count, dtype: int64
```

[43]:
```python
# Preprocess the data
def preprocess_data(data):
    # Separate features and target
    X = data.drop('booking status', axis=1)
    y = data['booking status']

    # Identify numerical and categorical columns
    numerical_columns = X.select_dtypes(include=[np.number]).columns
    categorical_columns = X.select_dtypes(include=['object']).columns

    # Create preprocessing pipelines
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('encoder', LabelEncoder())
    ])

    # Apply transformations
    X[numerical_columns] = numeric_transformer.
 ↪fit_transform(X[numerical_columns])
    for col in categorical_columns:
        X[col] = categorical_transformer.fit_transform(X[col])

    return X, y

X, y = preprocess_data(data)
```

[44]:
```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
print(f"Training set shape: {X_train.shape}")
```

```
print(f"Testing set shape: {X_test.shape}")
```

[45]:
```
# Define parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create and train the Decision Tree model with GridSearchCV
dt_model = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid,␣
 ↪cv=5, n_jobs=-1, verbose=1)
dt_model.fit(X_train, y_train)

print(f"Best parameters: {dt_model.best_params_}")
print(f"Best cross-validation score: {dt_model.best_score_:.4f}")
```

[46]:
```
# Get the best model
best_model = dt_model.best_estimator_

# Make predictions
y_pred = best_model.predict(X_test)

# Calculate and display metrics
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

[47]:
```
# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
Best parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split':
5}
```

[48]:
```
# Plot decision tree (limit depth for visibility)
plt.figure(figsize=(20, 10))
plot_tree(best_model, max_depth=3, feature_names=X.columns,␣
 ↪class_names=['Not_Canceled', 'Canceled'],
          filled=True, rounded=True, fontsize=10)
plt.title("Decision Tree (Limited to Depth 3 for Visibility)")
```

```
plt.show()
```

[49]:
```
# Feature Importance
feature_importance = pd.DataFrame({'feature': X.columns, 'importance':␣
 ↪best_model.feature_importances_})
feature_importance = feature_importance.sort_values('importance',␣
 ↪ascending=False).head(10)

plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Top 10 Feature Importance')
plt.show()
```

[50]:
```
# Additional analysis: Partial Dependence Plots
from sklearn.inspection import PartialDependenceDisplay

features = feature_importance['feature'].head(3).tolist()  # Top 3 important␣
 ↪features
fig, ax = plt.subplots(figsize=(15, 5))
PartialDependenceDisplay.from_estimator(best_model, X, features, ax=ax)
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.87
```