new-customer-churn

October 20, 2023

1 Task 1: Data Preprocessing

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.model_selection import GridSearchCV
```

[2]:
```
# Load the dataset
data = pd.read_csv("customer_churn_large_dataset.csv")
```

[3]:
```
# Explore the data
print(data.head())
print(data.info())
```

```
   CustomerID        Name  Age  Gender     Location  \
0           1  Customer_1   63    Male  Los Angeles
1           2  Customer_2   62  Female     New York
2           3  Customer_3   24  Female  Los Angeles
3           4  Customer_4   36  Female        Miami
4           5  Customer_5   46  Female        Miami

   Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn
0                          17         73.36             236      0
1                           1         48.76             172      0
2                           5         85.47             460      0
3                           3         97.94             297      1
4                          19         58.14             266      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   CustomerID                  100000 non-null  int64
```

```
 1   Name                         100000 non-null  object
 2   Age                          100000 non-null  int64
 3   Gender                       100000 non-null  object
 4   Location                     100000 non-null  object
 5   Subscription_Length_Months   100000 non-null  int64
 6   Monthly_Bill                 100000 non-null  float64
 7   Total_Usage_GB               100000 non-null  int64
 8   Churn                        100000 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 6.9+ MB
None
```

[4]:
```
data.duplicated().sum()
```

[4]:
```
0
```

[5]:
```
data.value_counts().sum()
```

[5]:
```
100000
```

[6]:
```
print(data.isnull().sum())
```

```
CustomerID                   0
Name                         0
Age                          0
Gender                       0
Location                     0
Subscription_Length_Months   0
Monthly_Bill                 0
Total_Usage_GB               0
Churn                        0
dtype: int64
```

[7]:
```
data.sample(5)
```

[7]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
68326	68327	Customer_68327	23	Female	Los Angeles	4	37.74	161	0
44180	44181	Customer_44181	66	Female	Chicago	19	84.85	493	1
10352	10353	Customer_10353	58	Female	Houston	5	89.36	77	0
18201	18202	Customer_18202	22	Female	Houston	23	42.89	215	0
45998	45999	Customer_45999	69	Female	Chicago	6	99.71	250	0

[8]:
```
# Handle missing data (e.g., replace missing values with the mean)
# data.fillna(data.mean(), inplace=True)
df = data.drop(columns=['CustomerID', 'Name'])
```

[9]:
```
df
```

[9]:

	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	63	Male	Los Angeles	17	73.36	236	0
1	62	Female	New York	1	48.76	172	0
2	24	Female	Los Angeles	5	85.47	460	0
3	36	Female	Miami	3	97.94	297	1
4	46	Female	Miami	19	58.14	266	0
…	…	…	…	…	…	…	…
99995	33	Male	Houston	23	55.13	226	1
99996	62	Female	New York	19	61.65	351	0
99997	64	Male	Chicago	17	96.11	251	1
99998	51	Female	New York	20	49.25	434	1
99999	27	Female	Los Angeles	19	76.57	173	1

[100000 rows x 7 columns]

[10]:
```
# Create an empty list to store the names of categorical columns
categorical_columns = []

# Loop through the columns in the DataFrame and check their data types
for column in data.columns:
    if data[column].dtype == 'object':  # Assuming object dtype represents␣
 ↪categorical data (strings)
        categorical_columns.append(column)

# Print the list of categorical columns
print("Categorical Columns:", categorical_columns)
```

```
Categorical Columns: ['Name', 'Gender', 'Location']
```

[11]:
```
# Encode categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
```

[12]:
```
label_encoder = LabelEncoder()
df['Location'] = label_encoder.fit_transform(df['Location'])
```

[13]:
```
# Split data into features (X) and target (y)
# X = df.drop('Churn', axis=1)
X = df.iloc[:,0:6]
y = df.iloc[:,-1]
```

[14]:
```
X
```

[14]:
```
       Age  Gender  Location  Subscription_Length_Months  Monthly_Bill  \
0       63       1         2                          17         73.36
1       62       0         4                           1         48.76
2       24       0         2                           5         85.47
3       36       0         3                           3         97.94
4       46       0         3                          19         58.14
…      …       …         …                           …            …
99995   33       1         1                          23         55.13
99996   62       0         4                          19         61.65
99997   64       1         0                          17         96.11
99998   51       0         4                          20         49.25
99999   27       0         2                          19         76.57

       Total_Usage_GB
0                 236
1                 172
2                 460
3                 297
4                 266
…                 …
99995             226
99996             351
99997             251
99998             434
99999             173

[100000 rows x 6 columns]
```

[15]:
```
# 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)
```

2 Task 2: Feature Engineering (create additional features if needed)

[16]:
```
# Apply feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3 Task 3: Model Building

[ ]:
```
# Choose an appropriate machine learning algorithm (e.g., Random Forest)
model = RandomForestClassifier(random_state=30)
# Train the model
model.fit(X_train, y_train)
```

[ ]:
```
# Make predictions
y_pred = model.predict(X_test)
```

[ ]:
```
y_pred
```

[ ]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

[ ]:
```
accuracy
```

[ ]:
```
precision = precision_score(y_test, y_pred)
```

[ ]:
```
precision
```

[ ]:
```
recall = recall_score(y_test, y_pred)
```

[ ]:
```
recall
```

[ ]:
```
f1 = f1_score(y_test, y_pred)
```

[ ]:
```
f1
```

4 Task 4: Model Optimization

[ ]:
```
# Fine-tune the model parameters (e.g., using GridSearchCV)
# param_grid = {
#     'n_estimators': [100, 200, 300],
#     'max_depth': [10, 20, 30]
# }
```

```
# grid_search = GridSearchCV(model, param_grid, cv=5, scoring='f1')
# grid_search.fit(X_train, y_train)
# best_model = grid_search.best_estimator_
```

5 Task 5: Model Deployment

[ ]:
```
# For deployment, you can create a function that takes new customer data as␣
↪input and returns churn predictions

def predict_churn(new_customer_data):
    # Preprocess new_customer_data (handle missing data, encode categorical␣
↪variables, apply feature scaling)
    # new_customer_data = new_customer_data.fillna(data.mean())
    new_customer_data['categorical_column'] = label_encoder.
↪transform(new_customer_data['categorical_column'])
    new_customer_data = scaler.transform(new_customer_data)

    # Predict churn
    churn_prediction = best_model.predict(new_customer_data)
    return churn_prediction
```

[ ]:
```

```