SpaceShip Titanic Prediction Report

M.Junaid 109(4B)

Table of Contents

Introduction

This report explains a machine learning project that predicts which passengers were transported to an alternate dimension during the SpaceShip Titanic collision with a spacetime anomaly. The code was written to preprocess data and create predictions that can be submitted to a Kaggle competition.

Data Loading and Exploration

First, we load the SpaceShip Titanic dataset and look at its basic information. This helps us understand what data we're working with, including passenger details, cabin information, and other features that might help predict transportation status.

```python
import pandas as pd
df=pd.read_csv("train.csv")
df.head()
```

[1] ✓ 1.3s Python

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	Maham Ofracculy
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	Juanna Vines
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	Altark Susent

Handling Missing Values

We check which columns have missing values. We need to fill these missing values because machine learning models can't work with incomplete data.

```python
null_columns = df.isnull().sum()
print(f'Total number of columns with null values: {null_columns[null_columns > 0].count()}')
print(null_columns[null_columns > 0])
```

```
Total number of columns with null values: 12
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
```

Filling Missing Values in Text Columns

For text columns like HomePlanet, CryoSleep, Destination, and VIP status, we fill missing values with the most common value in each column. This is a simple approach that works well for categorical data.

```python
# Object columns filled
for column in null_columns[null_columns > 0].index:
    if df[column].dtype == 'object':
        mode_value = df[column].mode()[0]
        df[column] = df[column].fillna(mode_value).astype(str)
```

Python

```
C:\Users\M.Junaid\AppData\Local\Temp\ipykernel_11460\1298781846.py:5: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill
  df[column] = df[column].fillna(mode_value).astype(str)
```

Filling Missing Values in Number Columns

For number columns like Age, RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck, we use a smarter approach. We first check for outliers (unusual values) using the IQR method. Then we fill missing values with the average of normal values. This helps prevent outliers from affecting our estimates.

```
for column in null_columns[null_columns > 0].index:
    if df[column].dtype in ['int64', 'float64']:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        if df[(df[column] < lower_bound) | (df[column] > upper_bound)].empty:
            mean_value = df[column].mean()
            df[column].fillna(mean_value,inplace=True)
        else:
            values = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)][column]
            df[column].fillna(values.mean(),inplace=True)
            df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = values.mean()
```

Converting Text to Numbers

Machine learning models need numbers, not text. We convert all text columns to numbers using Label Encoding, which assigns numerical values to categorical features.

```
from sklearn.preprocessing import LabelEncoder

object_columns = df.select_dtypes(include='object').columns.tolist()
label_encoder = LabelEncoder()
for column in object_columns:
    df[column] = label_encoder.fit_transform(df[column].astype(str))
```

Preparing Data for Training

- Separate the features (X)
- Scale the data so all features have similar ranges, which helps the model learn better

```
from sklearn.preprocessing import StandardScaler
X=df.drop(['PassengerId', 'Transported','CryoSleep','VIP'], axis=1)
scaler = StandardScaler()
df[X.columns]=scaler.fit_transform(df[X.columns])
```

Training the Model

We use a Random Forest Classifier because it works well for classification problems. The model learns patterns from our training data to predict whether a passenger was transported or not.

```
from sklearn.linear_model import LogisticRegression
import joblib
train_data=df.drop(['PassengerId', 'Transported'], axis=1)
Y=df['Transported']


model = LogisticRegression()

model.fit(train_data, Y)

joblib.dump(model, 'Mymodel.pkl')
```

Testing Process

Loading and Preprocessing Test Data

We load the test data and apply the same preprocessing steps as we did with the training data.

```
import pandas as pd
df=pd.read_csv("test.csv")
df.head()
```

Python

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name
0	0013_01	Earth	True	G/3/S	TRAPPIST-1e	27.0	False	0.0	0.0	0.0	0.0	0.0	Nelly Carsoning
1	0018_01	Earth	False	F/4/S	TRAPPIST-1e	19.0	False	0.0	9.0	0.0	2823.0	0.0	Lerome Peckers
2	0019_01	Europa	True	C/0/S	55 Cancri e	31.0	False	0.0	0.0	0.0	0.0	0.0	Sabih Unhearfus

Handling Missing Values in Test Data

The test data also has missing values that need to be filled.

```
null_columns = df.isnull().sum()
print(f'Total number of columns with null values: {null_columns[null_columns > 0].count()}')
print(null_columns[null_columns > 0])
```

```
Total number of columns with null values: 12
HomePlanet       87
CryoSleep        93
Cabin           100
Destination      92
Age              91
VIP              93
RoomService      82
FoodCourt       106
ShoppingMall     98
Spa             101
VRDeck           80
```

Filling Missing Values and Encoding

We apply the same methods to fill missing values and encode categorical variables as we did with the training data.

```
for column in null_columns[null_columns > 0].index:
    if df[column].dtype == 'object':
        mode_value = df[column].mode()[0]
        df[column] = df[column].fillna(mode_value)
        df[column] = df[column].astype(str)
```

```
for column in null_columns[null_columns > 0].index:
    if df[column].dtype in ['int64', 'float64']:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        if df[(df[column] < lower_bound) | (df[column] > upper_bound)].empty:

            mean_value = df[column].mean()
            df[column].fillna(mean_value,inplace=True)
        else:

            values = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)][column]
            df[column].fillna(values.mean(),inplace=True)
            df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = values.mean()
```

Converting Text to Numbers and Scaling

Machine learning models need numbers, not text. We convert all text columns to numbers using Label Encoding, which assigns numerical values to categorical features and scale the data.

```python
from sklearn.preprocessing import LabelEncoder

object_columns = df.select_dtypes(include='object').columns.tolist()
label_encoder = LabelEncoder()
for column in object_columns:
    df[column] = label_encoder.fit_transform(df[column].astype(str))
```

```python
from sklearn.preprocessing import StandardScaler
X=df.drop(['PassengerId','CryoSleep','VIP'], axis=1)
scaler = StandardScaler()
df[X.columns]=scaler.fit_transform(df[X.columns])

test_data=df.drop('PassengerId', axis=1)
```

Making Predictions on Test Data

- Loads our saved model
- Uses it to predict transportation status for the test data
- Creates a submission file with passenger IDs and predicted status
- Saves this file for uploading to Kaggle

```python
import joblib

model = joblib.load('Mymodel.pkl')
```

```python
predictions = model.predict(test_data)
submission_df = pd.DataFrame({'PassengerId': df['PassengerId'], 'Transported': predictions})

submission_df.to_csv('submission_file.csv', index=False)
```

Model Accuracy

The model's performance was evaluated using metrics like Accuracy, Precision, Recall, and F1-Score. These metrics help us understand how well our model is predicting passenger transportation status.