CS564 DATA SCIENCE METHODOLOGY
PROJECT REPORT

Project Overview

The spaceship Titanic is a famous Kaggle competition where the main task is to predict if a particular passenger will be transported to an alternate dimension or not. Having learnt about various supervised techniques and ensemble methods in this course, this project was an excellent way to implement the obtained knowledge to solve some real-world problems.

About the Dataset

The project consists of the train set and the test set. In the train set, the data has 8693 rows and 14 columns.

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
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	False
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	True
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	False
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	Solam Susent	False
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	Willy Santantines	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	Gravior Noxnuther	False
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	Kurta Mondalley	False
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	Fayey Connon	True
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	Celeon Hontichre	False
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	Propsh Hontichre	True

8693 rows × 14 columns

Whereas the test set consists of 4277 rows of data. The feature variables include PassengerID, HomePlanet, Cabin, Age, Destination, Name and several others. The variable to predict is "Transported" which consists of only True/False. The value True indicates that the passenger is transported, and False indicates that the passenger is not transported.

Data Preprocessing

1. Knowing your Dataset

The first step in preprocessing the data getting to know our dataset well. First, the NaN values were checked and the results are as follows:

```
PassengerId       0
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
Transported       0
```

As we can see, almost all the columns had NaN values which will be taken care of in the later stages.
Next, we check the value type of the columns.

```
PassengerId      object
HomePlanet       object
CryoSleep        object
Cabin            object
Destination      object
Age             float64
VIP              object
RoomService     float64
FoodCourt       float64
ShoppingMall    float64
Spa             float64
VRDeck          float64
Name             object
Transported        bool
dtype: object
```

As observed our dataset has a combination of object types and float types. For being able to use the data, we must convert the object types to floats or integer types.

Finally, we check if the dataset is balanced or not.

```
df['Transported'].value_counts()

True     4378
False    4315
Name: Transported, dtype: int64
```

As observed the dataset has a nice distribution of True and False, indicating that the dataset is well-balanced.

2. Data cleaning and manipulation

Since we have an idea about our dataset now it's time to perform some data manipulation. The first thing that we should do is convert the categorical, boolean attributes into numerical ones. I converted "HomePlanet", "CryoSleep", and "VIP" to numerical attributes.

```
#converting categorical/boolean values to numerical values
df['HomePlanet'].replace({"Earth":1.0, "Europa":2.0, "Mars": 3.0}, inplace =True)
df['CryoSleep'].replace({True:1.0, False:0.0}, inplace = True)
df['VIP'].replace({True:1.0,False:0.0}, inplace = True)
df
```

Later to clean the dataset, I moved my focus to the NaN values. My idea was to replace the NaN values of columns that had two or three unique values with the mode value. And for columns that were numerical from the beginning, I replaced their NaN values with their mean values.
Now with the mean values being taken care of, I tried using the binning method learnt in class on the "Age" attribute. I made five bins and then assigned a float value to each of them.

```
#binning the ages into age groups and then assigning the groups to a float

df.loc[df['Age'].between(0, 10, 'both'), 'Age_group'] = 'child'
df.loc[df['Age'].between(10, 20, 'right'), 'Age_group'] = 'teen'
df.loc[df['Age'].between(20, 35, 'right'), 'Age_group'] = 'young_adult'
df.loc[df['Age'].between(35, 50, 'right'), 'Age_group'] = 'adult'
df.loc[df['Age'].between(50, 80, 'right'), 'Age_group'] = 'old'

df['Age_group'].replace({"child":1.0, "teen":2.0, "young_adult": 3.0, "adult": 4.0, "old": 5.0}, inplace =True)
```

Finally, to clean the dataset one last time, I removed all the unnecessary columns.

Choosing the Model

The next step was to choose a model for our dataset. I tried out several options including KNN, XGBoost, and SVC and I ended up using XGBoost because it gave me the best accuracy. Then I split the “train” dataset into the “training” section and the “testing” section, with 80% of the dataset being used in training.

Hypertuning our Model

Hypertuning was essential to find the optimal hyperparameters for the model. As we know XGBoost had several parameters like learning_rate, n_estimators, max-depth etc which play a crucial role in model performance. Hence we used GridSearch with Cross Validation to hyper-tune our model. After the hyper tuning, the best parameters that we obtained were 0.1 for the learning rate, 200 for n_estimators and a max depth of 4 which fave an accuracy of 79.45% on the training section and an accuracy of 80.33% on the testing section.

```
#grid search with cross validation for hypertuning
xgb_model = xgb.XGBClassifier()
xgb_param_grid = {'learning_rate': [0.1, 0.01], 'n_estimators': [50,100,150,200,250],'max_depth':[2,3,4,5]}
grid= GridSearchCV(xgb_model, xgb_param_grid, cv=5, scoring='accuracy',return_train_score=False)
grid_search=grid.fit(X_train, y_train)
```

```
print(grid_search.best_params_)

accuracy = grid_search.best_score_ *100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
```

```
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
Accuracy for our training dataset with tuning is : 79.45%
```

```
#initializing the model with the optimal parameters and fitting the test set
model = xgb.XGBClassifier(learning_rate=0.1,n_estimators=200,max_depth = 3)

xgb_model.fit(X_train, y_train)

y_test_hat=xgb_model.predict(X_test)

test_accuracy=accuracy_score(y_test,y_test_hat)*100
```

```
print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 80.33%
```

Concluding Remarks

Despite an accuracy of 80.33% is not extremely high, we must take into account that the highest noted accuracy on Kaggle is below 90%, which indicates the complication of this dataset. The performance of the model might be improved by having a larger dataset and also better preprocessing.