

The Superior University

|  |  |  |
| --- | --- | --- |
| Name: Abdul Moeez | Roll No: 017 | Course: PAI Lab |
| Semester: 4th | Section: BSAI 4A | Department: Artificial Intelligence |
| Submitted To: Sir Rasikh | Total Marks: | Date: |

**Lab 2**

**Task: Spaceship Titanic: Documentation**

## **Overview**

This project aims to predict whether a passenger on the Spaceship Titanic was transported to an alternate dimension using machine learning. The dataset contains various features about the passengers, such as age, cabin, destination, and spending habits. The goal is to build a classification model that predicts the Transported column (target variable) with high accuracy.

The code uses **RandomForestClassifier** and **XGBoost** models, along with hyperparameter tuning and ensemble methods, to achieve the best possible accuracy. The final predictions are saved in a submission file for evaluation.

## **Code Workflow**

### **1. Data Loading**

* The dataset is loaded from CSV files:
  + train.csv: Contains the training data with features and the target variable (Transported).
  + test.csv: Contains the test data for making predictions.
  + sample\_submission.csv: A template for the submission file.

train\_df = pd.read\_csv("train.csv")  
test\_df = pd.read\_csv("test.csv")

### **2. Data Preprocessing**

The preprocessing function (preprocess\_data) performs the following steps:

#### **a. Drop Irrelevant Columns**

* Columns like Name, Cabin, and PassengerId are dropped as they are not useful for modeling.

df.drop(columns=['Name', 'Cabin', 'PassengerId'], inplace=True, errors="ignore")

#### **b. Handle Missing Values**

* Numeric columns are imputed with the median value.
* Categorical columns are imputed with the most frequent value (mode).

numeric\_imputer = SimpleImputer(strategy='median')  
df[numeric\_cols] = numeric\_imputer.fit\_transform(df[numeric\_cols])  
  
categorical\_imputer = SimpleImputer(strategy='most\_frequent')  
df[categorical\_cols] = categorical\_imputer.fit\_transform(df[categorical\_cols])

#### **c. Encode Categorical Variables**

* Categorical columns are encoded using LabelEncoder to convert them into numeric values.

label\_encoders = {}  
for col in categorical\_cols:  
 label\_encoders[col] = LabelEncoder()  
 df[col] = label\_encoders[col].fit\_transform(df[col].astype(str))

#### **d. Scale Numeric Features**

* Numeric features are scaled using StandardScaler to normalize their values.

scaler = StandardScaler()  
df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])

### **3. Model Training**

The model training process involves the following steps:

#### **a. Split Data into Training and Validation Sets**

* The training data is split into X\_train, X\_val, y\_train, and y\_val using an 80-20 split.

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#### **b. Hyperparameter Tuning with RandomizedSearchCV**

* RandomizedSearchCV is used to find the best hyperparameters for the RandomForestClassifier.
* The search space includes parameters like n\_estimators, max\_depth, min\_samples\_split, and max\_features.

random\_search = RandomizedSearchCV(  
 estimator=RandomForestClassifier(random\_state=42),  
 param\_distributions=param\_dist,  
 n\_iter=20,  
 cv=5,  
 scoring='accuracy',  
 n\_jobs=-1,  
 random\_state=42  
)  
random\_search.fit(X\_train, y\_train)

#### **c. Train XGBoost Model**

* An XGBoost model is trained with default hyperparameters.

xgb\_model = XGBClassifier(n\_estimators=200, learning\_rate=0.1, max\_depth=5, random\_state=42)  
xgb\_model.fit(X\_train, y\_train)

#### **d. Ensemble Model**

* A VotingClassifier is used to combine the best RandomForestClassifier and XGBoost models.
* The ensemble model uses soft voting to make predictions.

ensemble\_model = VotingClassifier(estimators=[  
 ('rf', random\_search.best\_estimator\_),  
 ('xgb', xgb\_model)  
], voting='soft')  
ensemble\_model.fit(X\_train, y\_train)

### **4. Model Evaluation**

* The ensemble model is evaluated on the validation set using accuracy\_score.

y\_predict = ensemble\_model.predict(X\_val)  
accuracy = accuracy\_score(y\_val, y\_predict)  
print(f"Validation Accuracy: {accuracy}")

### **5. Generate Predictions**

* The final model is retrained on the full training dataset.
* Predictions are made on the test dataset and saved to a submission file.

final\_model = ensemble\_model  
final\_model.fit(X, y)  
  
test\_pred = final\_model.predict(test\_df)  
  
submission = pd.read\_csv('sample\_submission.csv')  
submission['Transported'] = test\_pred.astype(bool)  
submission.to\_csv('submission.csv', index=False)  
print("Submission file created: submission.csv")

## **Key Features**

1. **Data Preprocessing:**
   1. Handles missing values effectively.
   2. Encodes categorical variables.
   3. Scales numeric features.
2. **Model Training:**
   1. Uses RandomizedSearchCV for efficient hyperparameter tuning.
   2. Combines RandomForestClassifier and XGBoost into an ensemble model.
3. **Evaluation:**
   1. Evaluates the model using accuracy on a validation set.
4. **Submission:**
   1. Generates a submission file with predictions for the test dataset.

## **How to Run the Code**

1. Ensure the following libraries are installed:
   1. pandas, numpy, scikit-learn, xgboost
2. Place the dataset files (train.csv, test.csv, sample\_submission.csv) in the correct directory.
3. Run the script. The submission file (submission.csv) will be generated in the working directory.

## **Expected Results**

* The model achieves “0.7918343875790684” **accuracy** on the validation set.
* The submission file contains predictions for the Transported column in the test dataset.

## **Output:**

