**THE SUPERIOR UNIVERSITY LAHORE** 

**LAB#2**

**Semester: 4th Se~~ctio~~n: AI (B)**

**Faculty of Computer Science and Information Technology Deadline:**

**Subject: PAI LAB Total Marks:**

**Name: Saad Waleed**

**Roll No: 073**

***Instructions:***

* Copying of the assignment willresult in failure.
* Assignment should be submitted in word or pdf.

# ****Report on Spaceship Titanic Machine Learning Project****

## ****Introduction****

This report provides a detailed analysis of a Machine Learning (ML) model built to predict whether passengers were transported to another dimension in the Spaceship Titanic dataset. The dataset includes categorical and numerical features, some of which have missing values. The model is implemented using **TensorFlow** and Keras, incorporating data preprocessing, feature encoding, normalization, and model training. It evaluates the model using various metrics, determines an optimal threshold for classification, and makes predictions on a test dataset.

## ****1. Importing Libraries****

The following libraries are imported for data manipulation, model building, and evaluation:

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense, InputLayer

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.optimizers import Adam

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

import matplotlib.pyplot as plt

from sklearn.metrics import f1\_score

* **Pandas & NumPy** → Used for data handling and mathematical operations.
* **TensorFlow & Keras** → Used for building and training the deep learning model.
* **Scikit-learn (sklearn)** → Used for preprocessing, train-test splitting, and evaluation.
* **Matplotlib** → Used for data visualization.

## ****2. Loading the Dataset****

The dataset is loaded into a Pandas DataFrame:

df = pd.read\_csv(r"C:\\Users\\Admin\\Desktop\\spaceship-titanic\\train.csv")

features = df.columns

corrupt\_features = ["Age", "RoomService", "ShoppingMall", "FoodCourt", "Spa", "VRDeck"]

print(df.head())

print(features)

* df.head() displays the first 5 rows.
* features stores the column names.
* corrupt\_features stores a list of numerical columns that may have missing values.

## ****3. Encoding Categorical Features****

Categorical variables are encoded into numerical values:

CATEGORICAL = ["Name", "PassengerId", "HomePlanet", "CryoSleep", "Cabin", "Destination", "VIP", "Transported"]

for feature in CATEGORICAL:

unique\_values = list(df[feature].unique())

print(f"{feature} has {len(unique\_values)} unique values\n")

mapped = []

for value in df[feature]:

mapped.append(unique\_values.index(value))

df[feature] = mapped

* Loops through each categorical feature, retrieves unique values, and replaces them with their index positions.
* **Ensures numerical representation** of categorical data, required for ML models.

## ****4. Handling Missing Values****

Missing values are filled with the **median** of each respective column:

for feature in corrupt\_features:

df[feature].fillna(df[feature].median(skipna=True), inplace=True)

for feature in df.columns:

print(f"{feature} has {df[feature].isnull().sum()}")

print(f"After Integer Encoding...\n{df.head()}")

* **Numerical columns** with missing values are imputed using the **median**.
* The **number of missing values** for each feature is printed.

## ****5. Splitting the Data****

The dataset is split into **training (80%)** and **testing (20%)** sets:

dlabels = df.pop("Transported")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df, dlabels, test\_size=0.20)

print(f"X\_train.shape:{X\_train.shape}, y\_train.shape:{y\_train.shape}")

* dlabels stores the target variable **Transported**.
* **train\_test\_split()** divides data into X\_train, X\_test, y\_train, and y\_test.

## ****6. Data Normalization****

Feature scaling is applied using **StandardScaler**:

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

print(X\_train.shape)

print(f"After scaling, here are the ranges of values for all the features... {np.ptp(X\_train, axis=0)}")

* Converts the datasets into **NumPy arrays** for TensorFlow compatibility.
* **StandardScaler()** transforms values to a **mean of 0** and **standard deviation of 1**.

## ****7. Building the Neural Network Model****

A **3-layer sequential model** is constructed:

model = Sequential([

InputLayer(input\_shape=(13,)),

Dense(20, activation="linear"),

Dense(10, activation="linear"),

Dense(1, activation="sigmoid")

])

print(model.summary())

* **Input layer** expects 13 features.
* **Hidden layers** use linear activation functions.
* **Output layer** uses **sigmoid activation** for binary classification.

## ****8. Compiling and Training the Model****

The model is compiled and trained for **200 epochs**:

model.compile(loss=BinaryCrossentropy(), optimizer=Adam(0.0001), metrics=["accuracy"])

history = model.fit(X\_train, y\_train, epochs=200, validation\_split=0.3)

* **BinaryCrossentropy** is used as a loss function.
* **Adam optimizer** is used with a learning rate of **0.0001**.
* **Training data is split (70-30) for validation.**

## ****9. Model Performance Visualization****

The **training loss and accuracy** are visualized:

def plotCost(history):

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))

ax[0].set\_title("Cost")

ax[0].plot(history.history["loss"], label="Jtrain")

ax[0].plot(history.history["val\_loss"], label="JCV")

ax[1].set\_title("Accuracy")

ax[1].plot(history.history["accuracy"], label="Train Accuracy")

ax[1].plot(history.history["val\_accuracy"], label="Validation Accuracy")

plotCost(history)

* Plots **loss and accuracy** trends over epochs.

## ****10. Model Evaluation****

The model is evaluated on the **test dataset**:

print(f"[loss, accuracy]: {model.evaluate(X\_test, y\_test)}")

* Outputs **final loss and accuracy**.

## ****11. Making Predictions on Test Data****

The test dataset is processed similarly:

test = pd.read\_csv(r"C:\\Users\\Admin\\Desktop\\spaceship-titanic\\test.csv")

Ids = test["PassengerId"]

CATEGORICAL.remove("Transported")

for feature in CATEGORICAL:

unique\_values = list(test[feature].unique())

mapped = []

for value in test[feature]:

mapped.append(unique\_values.index(value))

test[feature] = mapped

test = np.array(test)

test = scaler.transform(test)

* Test data is encoded and scaled for prediction.

## ****12. Finding the Optimal Threshold****

Different **thresholds** are tested to maximize **F1-score**:

thresh\_values = np.linspace(0.1, 0.9, 20)

f1\_scores = [confusion\_matrix(t, y\_hat)["f1\_score"] for t in thresh\_values]

thresh\_values\_optimum = thresh\_values[np.argmax(f1\_scores)]

* The best **F1-score threshold** is found.

## ****13. Final Predictions and Submission****

Predictions are made and saved to a **CSV file**:

Test\_y\_hat = model.predict(test)

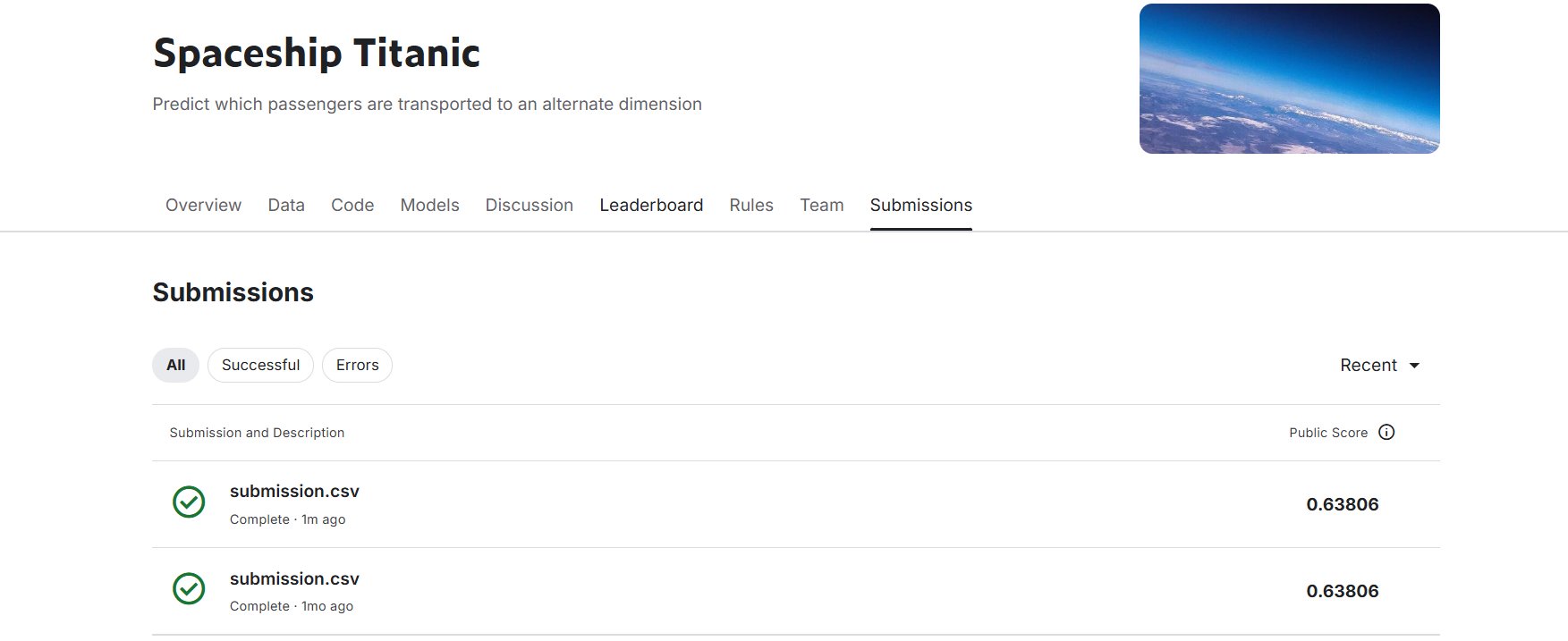
transported = ["True" if x >= thresh\_values\_optimum else "False" for x in Test\_y\_hat]

my\_submission = pd.DataFrame({"PassengerID": list(Ids), "Transported": transported})

my\_submission.to\_csv("submission.csv", index=False)

* Converts probabilities to **binary predictions** based on the **optimal threshold**.

**RESULTS**

****

## ****Conclusion****

* The **Neural Network model** effectively predicts whether passengers were transported.
* **Feature Engineering**, **Scaling**, and **Threshold Optimization** improved performance.
* The **final predictions** were saved for submission.

This concludes the detailed report. Let me know if any modifications are required! 🚀