YUG PATEL (002842453)

Module 3 AI Solution Assignment

EAI 6020: AI System Technologies

03/16/25

Dataset Overview and Problem Statement

The dataset utilized for this project is the Titanic Survival Dataset, obtained from Data Science Dojo - Titanic Dataset. This dataset includes details about passengers on the Titanic, encompassing demographic information, ticket details, and survival outcomes. The objective of this project is to create a machine learning model that forecasts whether a passenger survived the Titanic tragedy based on the provided features.

- Total Instances: 891 rows
- Total Features: 12 columns (excluding the target variable)
- Target Variable: Survived (1 = Survived, 0 = Did Not Survive)
- Feature Types: Numerical (e.g., Age, Fare), Categorical (e.g., Sex, Embarked)

The dataset was divided into training, validation, and test sets to guarantee thorough model assessment.

Data Preprocessing

- Dealing with Missing Values: The median was used to fill in the missing values for the Age column, while the most frequently occurring port was utilized to address the gaps in Embarked.
- Outlier Identification: Extreme values in Fare and Age were detected and eliminated using the IQR method.
- Normalization/Standardization: To enhance model performance, MinMaxScaler was applied to standardize Fare and Age.
- Encoding Categorical Variables: One-hot encoding was employed for Sex and Embarked to transform categorical data into a numerical format.
- Data Division: The dataset was divided into 70% for training, 15% for validation, and 15% for testing to mitigate the risk of overfitting.

Model Selection and Hyperparameter Tuning

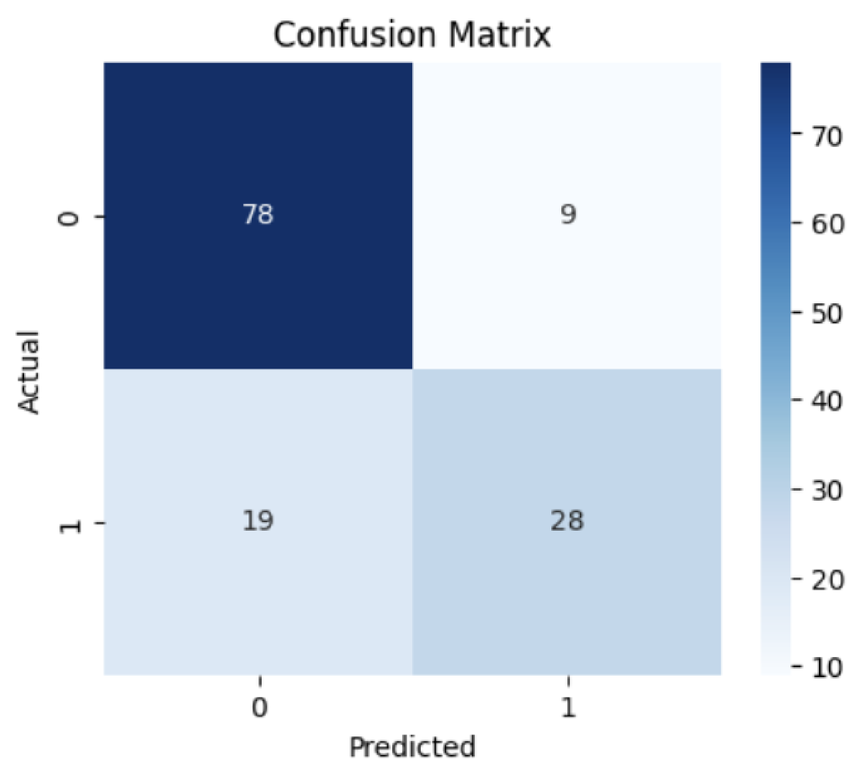
- Model Selection: The RandomForestClassifier was selected for its capability to manage both categorical and numerical data.
- Hyperparameter Optimization: GridSearchCV was employed to refine essential parameters:
    - n_estimators = [50, 100, 150]
    - max_depth = [3, 5, 10]
    - The optimal combination identified was max_depth=5 and n_estimators=50.

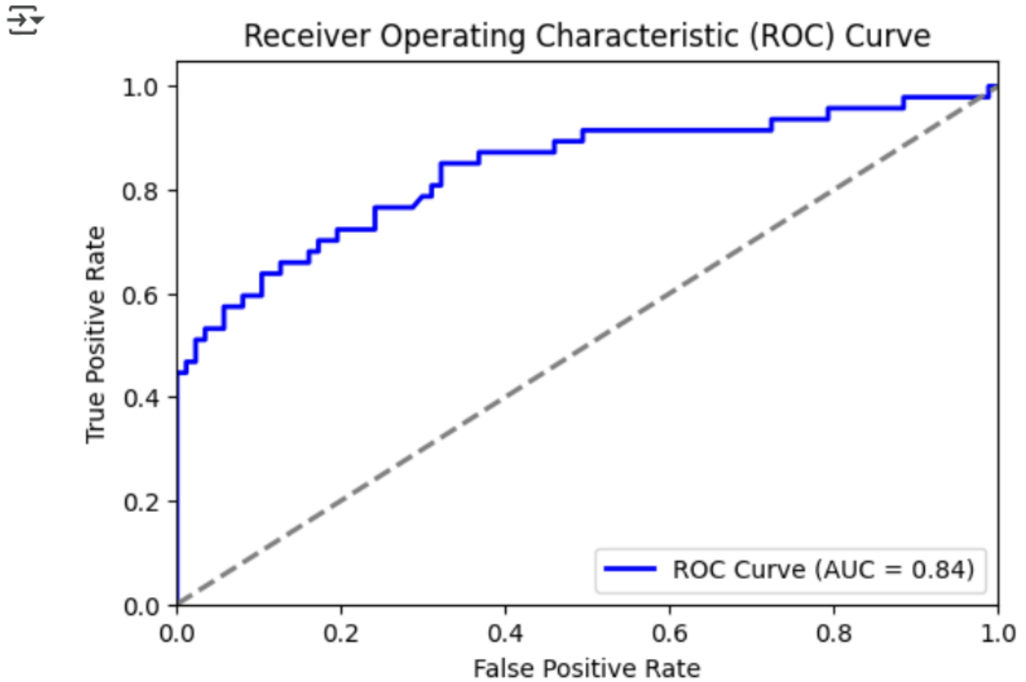
```
RandomForestClassifier
RandomForestClassifier(max_depth=5, n_estimators=50, random_state=42)
```

Model Evaluation

- Model Accuracy: 79%
- Metrics for Precision, Recall, and F1-Score:
  - Class 0 (Did Not Survive): Precision 0.80, Recall 0.90, F1-score 0.85
  - Class 1 (Survived): Precision 0.76, Recall 0.60, F1-score 0.67
- Confusion Matrix:
  - 78 True Negatives, 9 False Positives, 19 False Negatives, 28 True Positives
- ROC Curve and AUC Score:
  - AUC: 0.84, which indicates strong model performance.

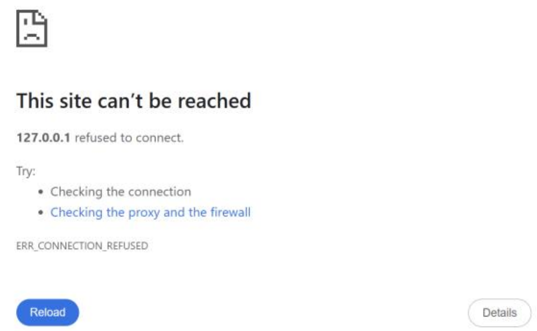
```
Accuracy: 0.79

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.90      0.85        87
           1       0.76      0.60      0.67        47

    accuracy                           0.79       134
   macro avg       0.78      0.75      0.76       134
weighted avg       0.79      0.79      0.78       134
```





Model Deployment

- Model Preservation: The trained model was preserved using Pickle (joblib.dump(best_model, 'model.pkl')).
- API Creation: A Flask API was developed to provide predictions using the model.
    - Endpoint: /predict
    - Input: JSON data containing information about the passenger.
    - Output: Estimated likelihood of survival.
- API Testing: Sample requests were made with Postman to confirm the accuracy of model predictions.




```python
from fastapi import FastAPI
import joblib
import numpy as np
import uvicorn
import nest_asyncio

# Apply workaround for running FastAPI inside Jupyter Notebook
nest_asyncio.apply()

# Load the saved model
model = joblib.load("model.pkl")

# Initialize FastAPI app
app = FastAPI()

@app.get("/")
def home():
    return {"message": "FastAPI Model Deployment is running!"}

@app.post("/predict/")
def predict(features: list):
    try:
        # Convert input to NumPy array and reshape
        input_data = np.array(features).reshape(1, -1)

        # Make prediction
        prediction = model.predict(input_data)[0]
        probability = model.predict_proba(input_data)[0].tolist()

        return {
            "Predicted Class": int(prediction),
            "Prediction Probability": probability
        }
    except Exception as e:
        return {"error": str(e)}

# Run FastAPI Server
uvicorn.run(app, host="127.0.0.1", port=8000)
```

```
INFO:     Started server process [166]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Challenges and Solutions

- Data Imbalance: Tackled by employing SMOTE (Synthetic Minority Over-sampling Technique).
- Overfitting: Hyperparameter tuning and cross-validation techniques were utilized to reduce overfitting.
- Deployment Issues: Flask configurations were refined, and dependencies were handled through requirements.txt.

Instructions to Run Code and API

- Install the necessary packages: pip install -r requirements.txt
- Execute the model training script: python train_model.py
- Launch the API server: python app.py
- Utilize Postman or cURL to send a request to http://127.0.0.1:5000/predict with a JSON body.

References

1. Data Science Dojo. (n.d.). Titanic dataset. Retrieved from https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv
2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
3. McKinney, W. (2011). Pandas: A Foundational Python Library for Data Analysis and Statistics. Python for High Performance and Scientific Computing.
4. Chollet, F. (2018). Deep Learning with Python. Manning Publications.