Analysis & Reporting

Comprehensive Project Analysis

Mirco Fernando – 0103

1 Introduction

This report focuses on analysing the “Titanic Survival Prediction Dataset”, which gives the information about the passengers that was aboard the titanic and the prediction rate for each passenger respectively. The primary objective of this project is to investigate and compare the performance of various supervised learning models in predicting passenger survival

The workflow of this project involved:

- •**Data preprocessing**: handling missing values, encoding categorical variables (e.g., *Sex*, *Embarked*), feature scaling, and minimal feature engineering
- **Model development**: implementing machine learning algorithms both **from scratch** and using established libraries (e.g., scikit-learn)
- **Evaluation**: applying multiple performance metrics (Precision, Recall, F1 Score, Accuracy, Confusion Matrix and ROC curve) to assess model accuracy and generalisation.
- **Analysis**: examining the impact of regularisation, model complexity, and ensemble methods on predictive performance.

The main focus in the project is to identify which model fits the dataset best and also to gain deeper insights into why certain models perform better, how regularisation influences logistics regression, and how ensemble methods enhance the performance of the Trees Algorithm.

2 Data preprocessing

2.1 Code and Output

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

df.isnull().sum()  #Checking for missing values

df.fillna({'Age': df['Age'].median(), 'Embarked': df['Embarked'].mode()[0]},
inplace=True)
```

```
df.drop('Cabin', axis=1, inplace=True)
le = LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex'])

embarked_mapping = {"S" : 1 , "C" : 2 , "Q" : 3}

for df_set in [df]:
    df_set['Embarked'] = df_set['Embarked'].map(embarked_mapping)

df.head()

df.drop(['Ticket', 'PassengerId'], axis=1, inplace=True)

X = df.drop('Survived', axis=1)
y = df['Survived']

SEED = 42
np.random.seed(SEED)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED
)
name_list = X_test['Name']

for df_set in [X_train, X_test]:
    df_set.drop('Name', axis=1, inplace=True)


X_train.to_csv("X_train.csv", index=False)
X_test.to_csv("X_test.csv", index=False)
y_train.to_csv("y_train.csv", index=False)
y_test.to_csv("y_test.csv", index=False)
name_list.to_csv("names_list.csv", index=False)


print(X)
print("--------")
print(y)
print("--------")
print(X_test)
print("--------")
print(y_test)
```

One of the crucial steps before model development is **data preprocessing** to ensure consistency, interpretability, and compatibility with machine learning algorithms to enhance model training. The following steps were carried out:

Before model training, the dataset was analysed then cleaned and prepared with the following steps:

1. **Exploratory Data Analysis (EDA):**
   Checked data types, missing values, and class balance, while reviewing key features such as *Age*, *Sex*, *Fare*, and *Pclass*.
2. **Handling Missing Values:**
   - Filled missing *Age* with the median.
   - Replaced missing *Embarked* with the most frequent value.
3. **Encoding & Scaling:**
   - Converted categorical features (*Sex*, *Embarked*) into numeric form.
   - Normalised continuous features (*Age*, *Fare*) for consistent model performance.
4. **Train-Test Split:**
   - Split the dataset into 80% training and 20% testing
   - Saved the splits into CSV files for use across both scratch and library models.

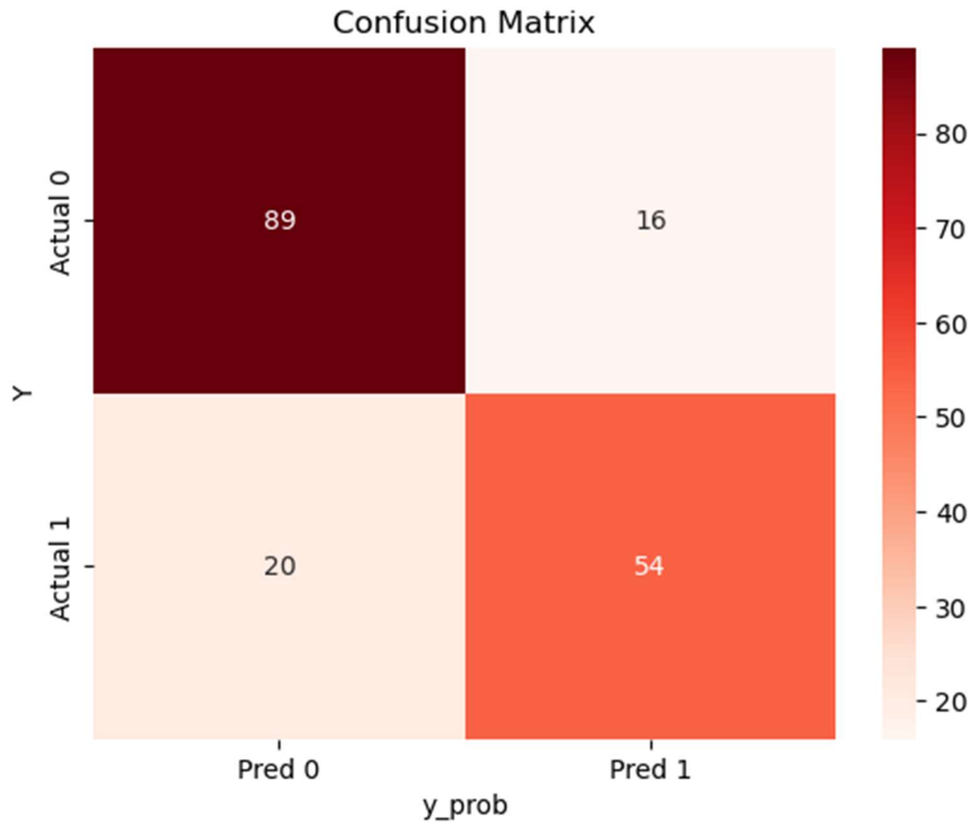
These phases ensured the data was consistent and ready for fair comparison across all algorithms to train.

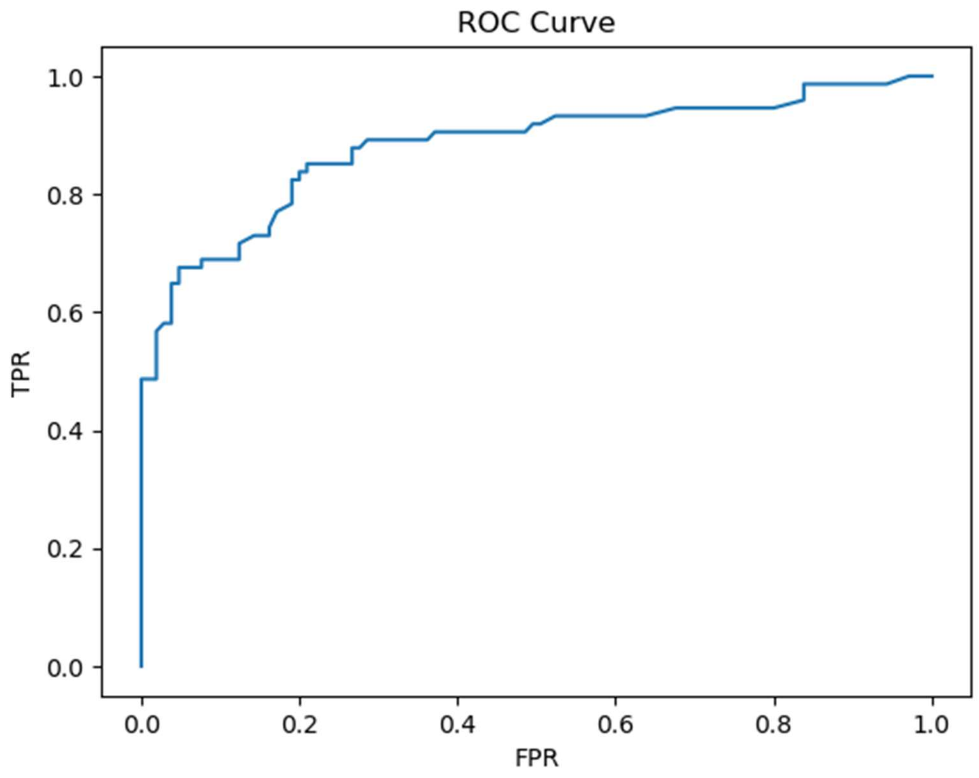
3 Logistics Regression Variants Performance

I implemented Logistics Regression variants, logistics regression with no penalty, L1 and L2 these models, lets analyse the predicted values using evaluation metrics.

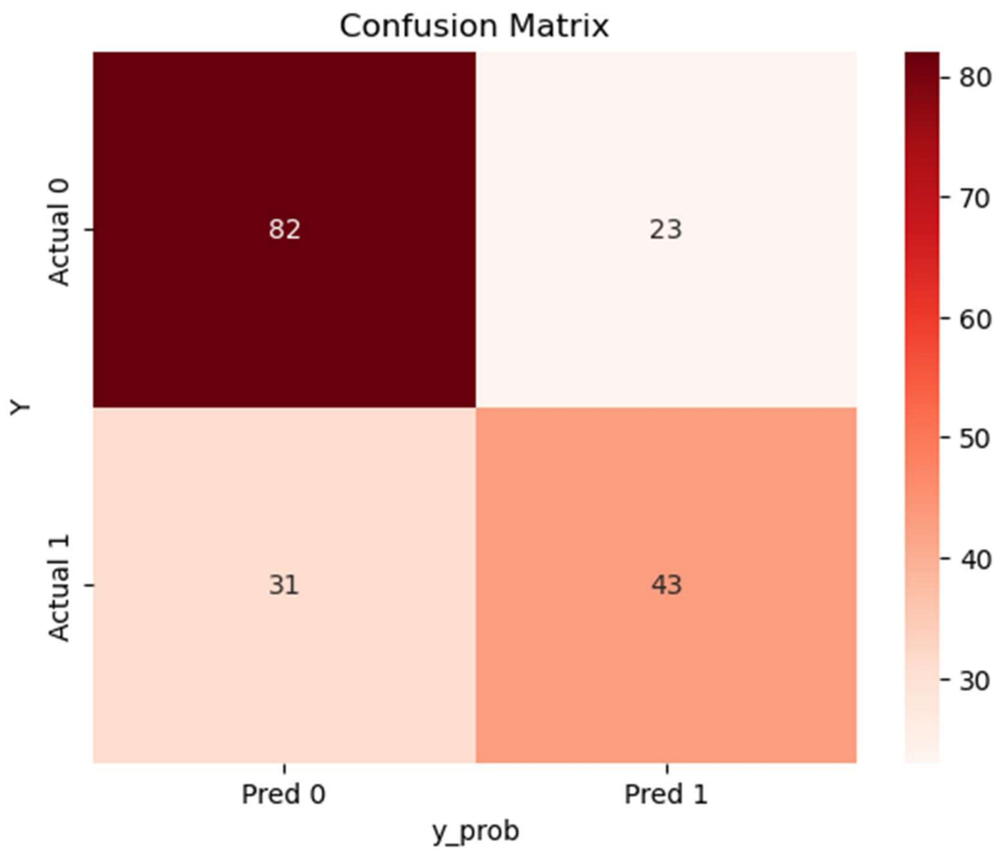
Model Variant	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Notes**
Logistic Regression (No Penalty)	0.77	0.72	0.75	0.79	Baseline model, decent balance but prone to overfitting.
Logistic Regression (L1)	0.65	0.58	0.61	0.69	L1 shrinks some coefficients to zero → adds feature selection effect. Weaker performance
Logistic Regression (L2)	0.77	0.74	0.75	0.80	L2 improves generalisation, handles multicollinearity effectively. More robust less prone to overfitting

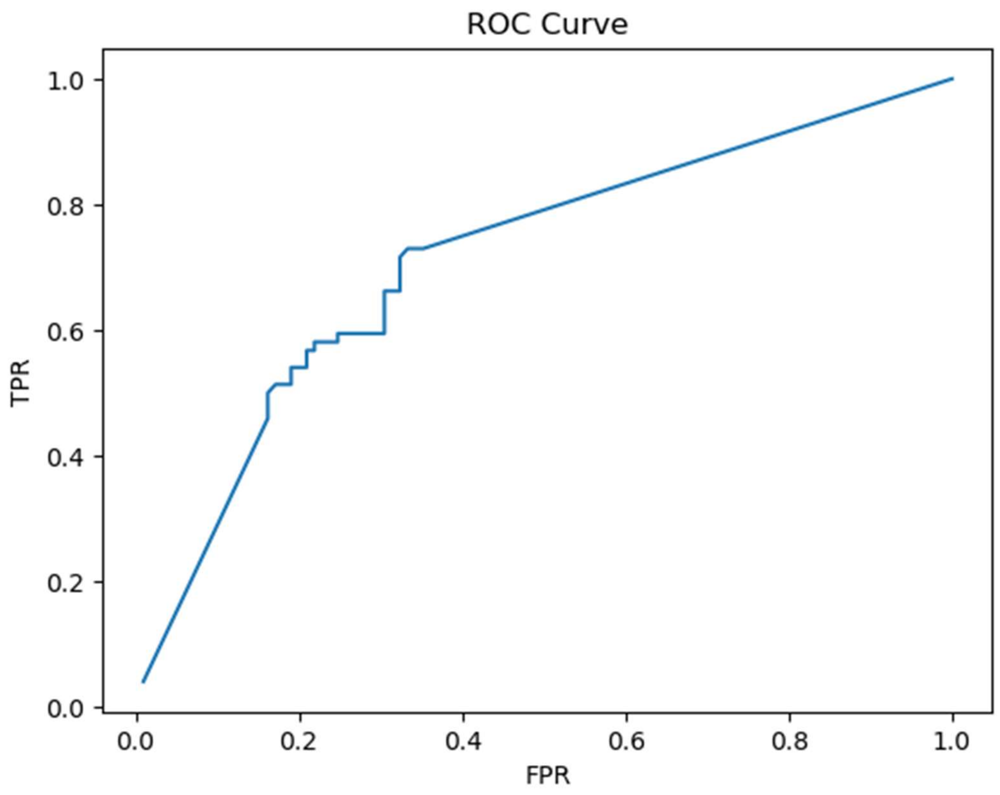
Logistics regression (No penalties) – Confusion matrix and ROC



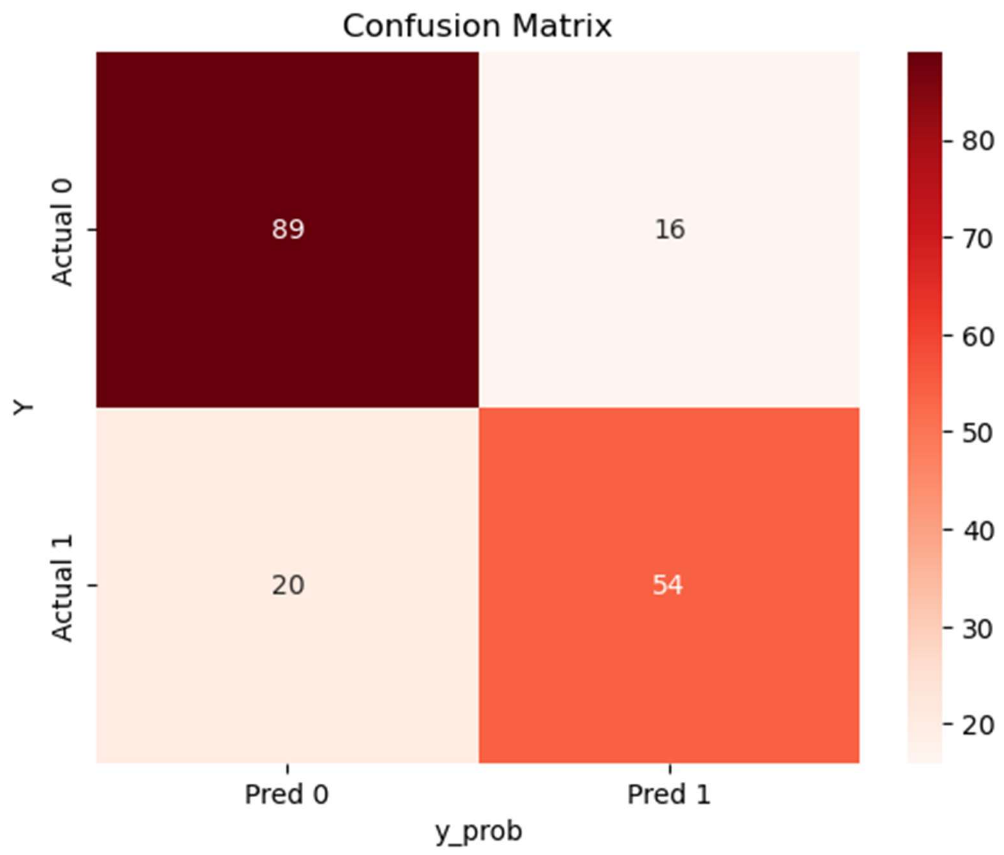


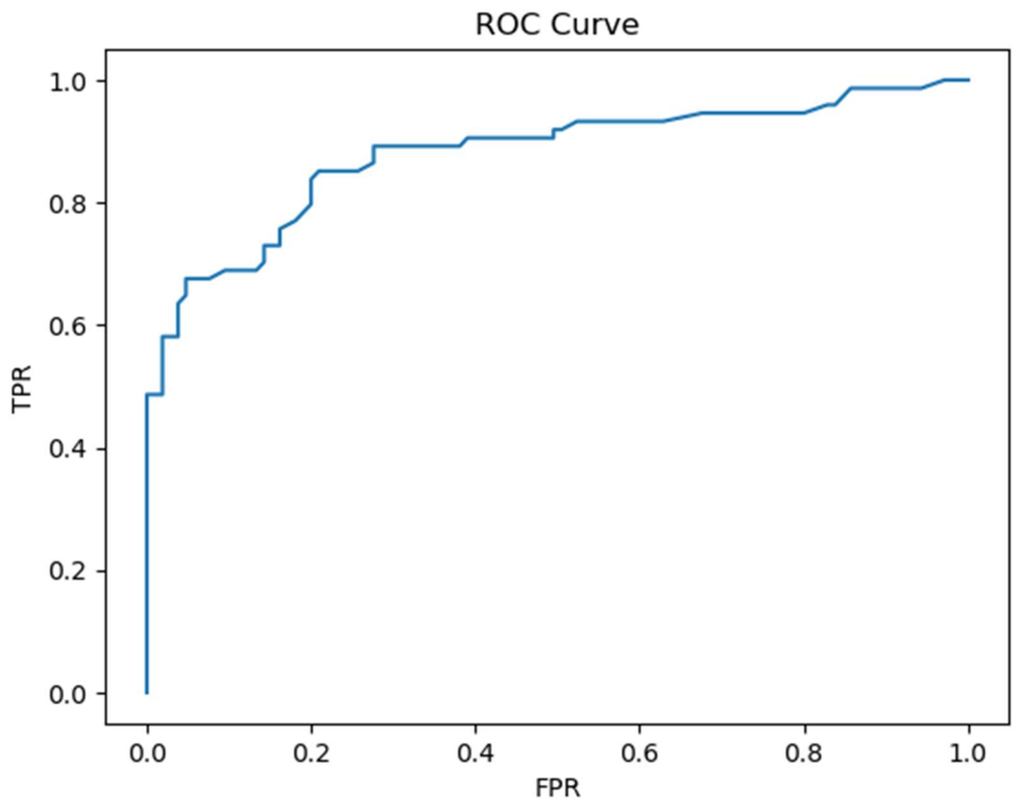
Logistics regression (L1) – Confusion matrix and ROC





Logistics regression (L2) – Confusion matrix and ROC





I trained all three models with 10000 as the number of iterations and the same penalty value of 1000 for both L1 and L2. After experimenting with different settings, these values gave the best fit.

In conclusion Logistics regression L2 predicts values efficiently rather than the other two models. Since normal Logistics Regression has no penalty it can't perform regularisation, if the data is not processed well it is more likely to overfit and L1 is tend to optimize features and can shrink coefficients to zero can lead to a weaker trained model, L2 regularisation penalises large and shrinks coefficients correctly, according to the evaluation metrics it has the best scores when compared to other models.

4 Bagging vs boosting performance insights

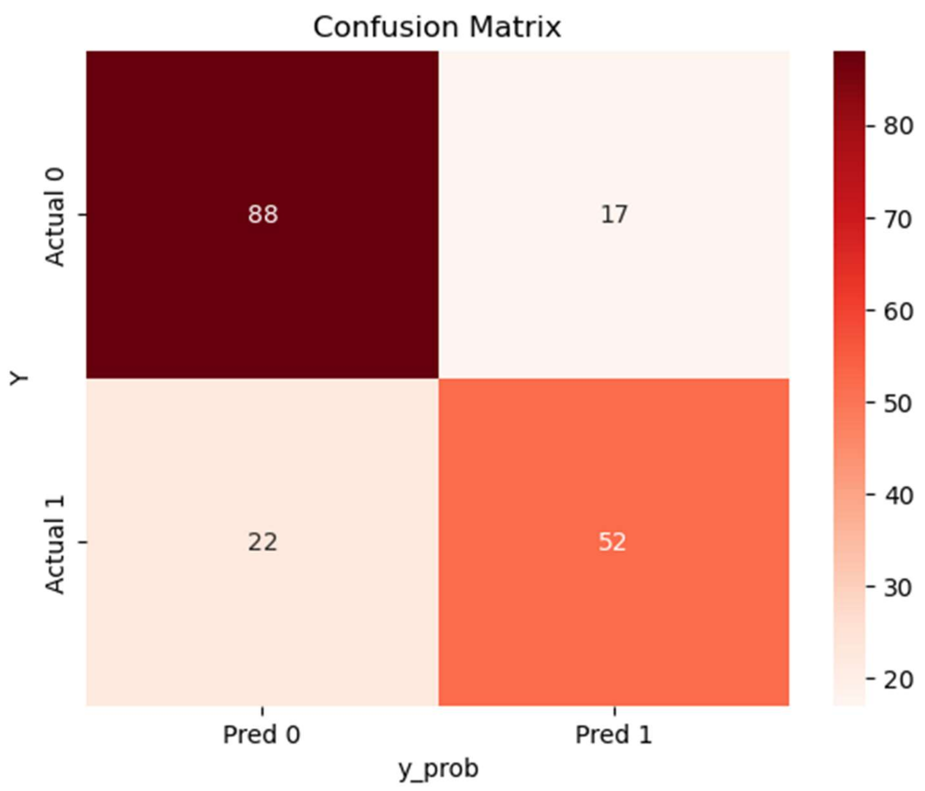
Let's analyse how ensemble methods affect the performance of classification algorithms. The two main types of ensemble methods are bagging and boosting.

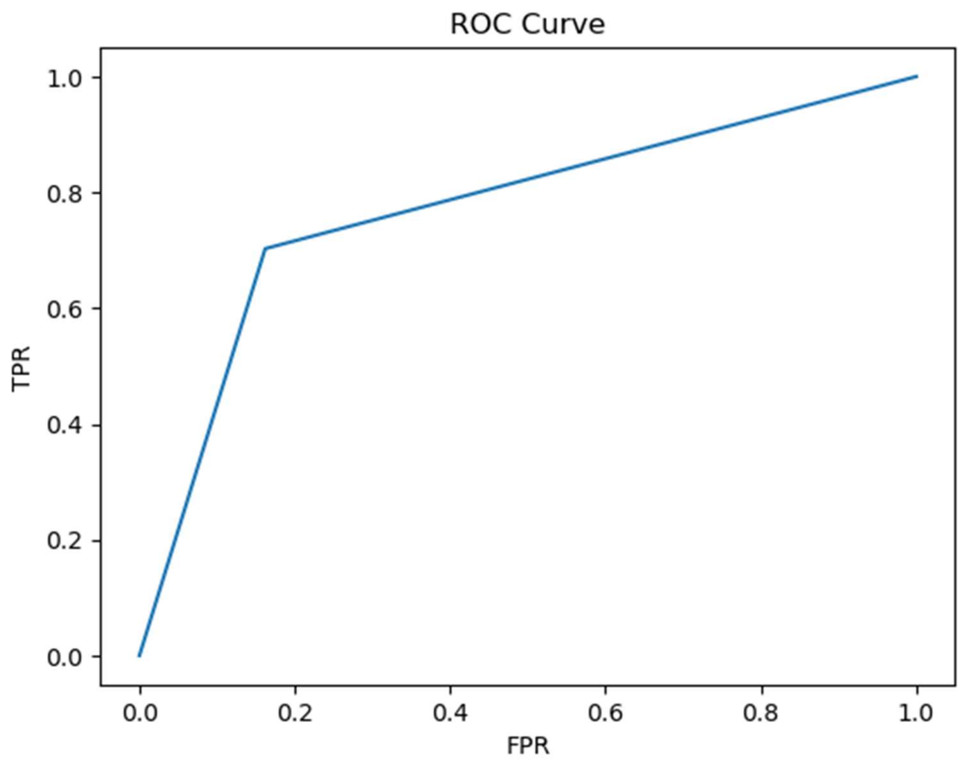
Firstly, I implemented a single Decision Tree model and observed its predictions. Using this base learner, I developed Bagging and Boosting models. Bagging combined multiple trees trained on bootstrap samples, which reduced variance and improved prediction stability. Boosting sequentially trained trees, each correcting errors of the previous one, which reduced bias and improved overall accuracy.

Evaluation Metrics

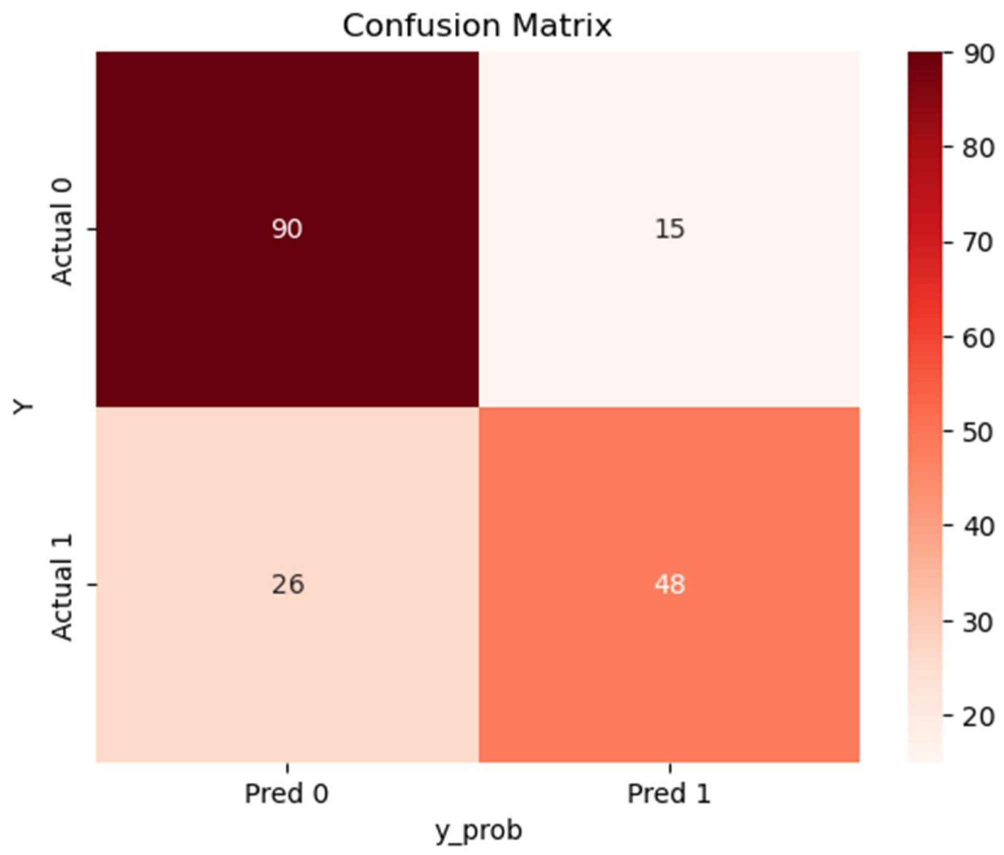
Model	**Precision**	**Recall**	**F1 Score**	**Accuracy**	**Notes**
Decision Tree (Shallow)	0.75	0.70	0.72	0.78	Simple tree, limited depth, may underfit.
Decision Tree (Deep)	0.76	0.64	0.70	0.77	Deep tree, captures more patterns, may overfit.
Bagging (Ensemble of Trees)	0.79	0.68	0.74	0.81	Aggregates multiple trees, reduces variance, more stable predictions.

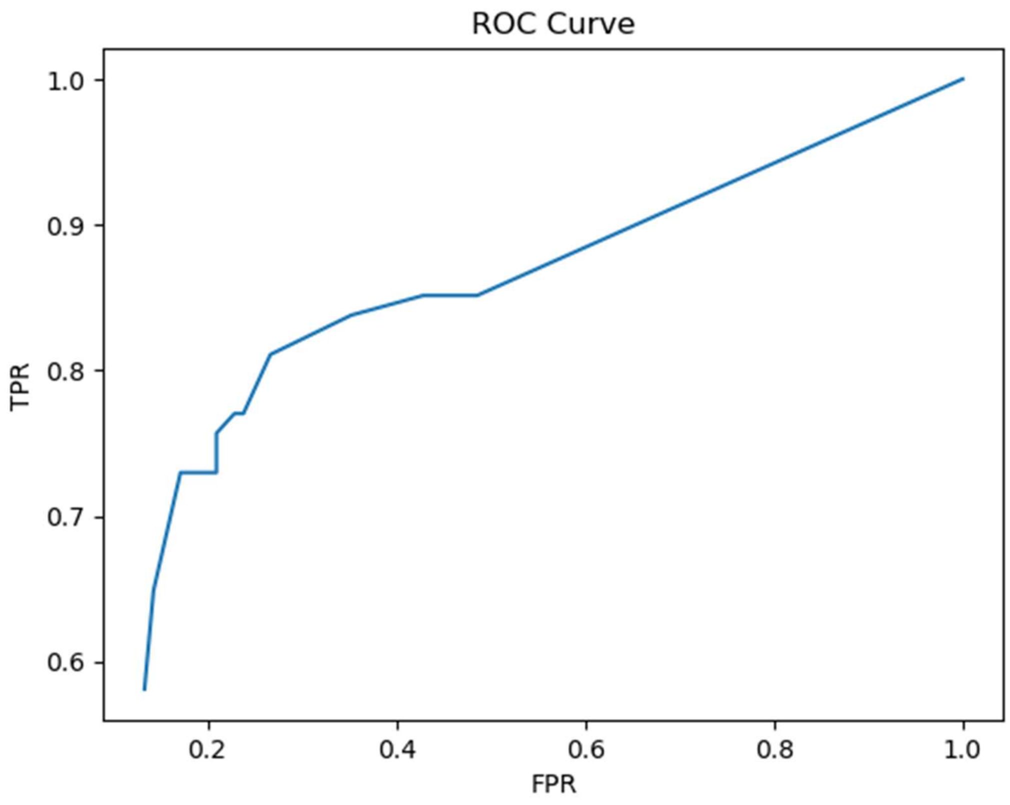
DecisionTree Classifications (Shallow Tree - Less depth)



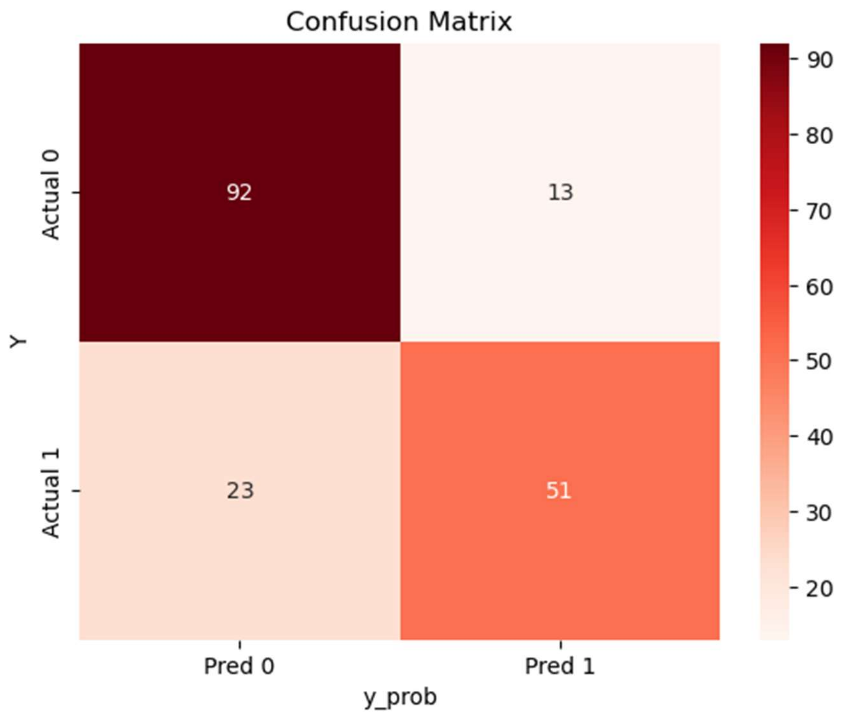


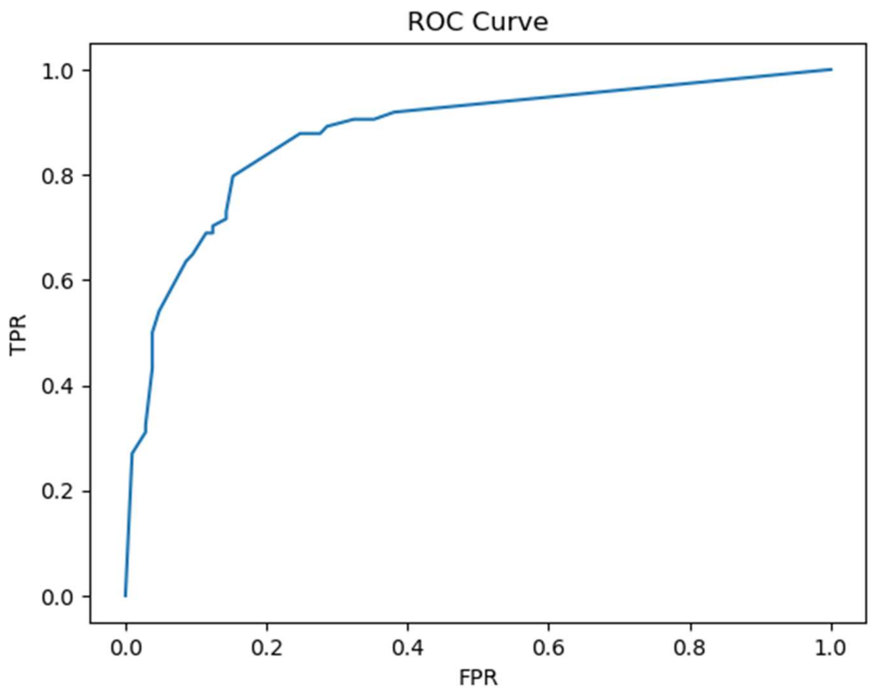
DecisionTree Classifications (Deep Tree - More depth)





Bagging Ensemble Method





As shown in the observations we can clearly see the bagging ensemble methods improved the model stability by training multiple decision trees on the different bootstrap samples and averaging their predictions. This reduces the overall variance and prevents overfitting. The bagging method produced robust predictions with fewer errors when comparing with the decision trees.

5 Comparison Analysis of Scratch and Library models

Model Type	Implementation	Precision	Recall	F1 Score	Accuracy	Notes
Logistic Regression	From Scratch	0.77	0.72	0.75	0.79	Baseline model, decent balance but prone to overfitting.
Logistic Regression (L1)	From Scratch	0.65	0.58	0.61	0.69	L1 adds feature selection effect, reduces overfitting.
Logistic Regression (L2)	From Scratch	0.77	0.73	0.75	0.80	L2 improves generalisation, handles multicollinearity.
Decision Tree (Shallow)	From Scratch	0.75	0.70	0.72	0.78	Limited depth, may underfit.
Decision Tree (Deep)	From Scratch	0.77	0.64	0.70	0.78	Captures more patterns, risk of overfitting.
Bagging (Ensemble of Trees)	From Scratch	0.79	0.68	0.74	0.80	Aggregates trees, reduces variance, stable predictions.
Logistic Regression	scikit-learn	0.77	0.73	0.75	0.79	Library implementation, similar performance to scratch.
Logistic Regression (L1)	scikit-learn	0.77	0.73	0.75	0.79	Library L1, feature selection improves generalization.
Logistic Regression (L2)	scikit-learn	0.77	0.74	0.75	0.79	Library L2, effective regularization.
Decision Tree (Shallow)	scikit-learn	0.75	0.70	0.73	0.78	Shallow tree, may underfit.
Decision Tree (Deep)	scikit-learn	0.77	0.70	0.73	0.80	Deep tree, may overfit if deep.
Random Forest / Bagging	scikit-learn	0.80	0.75	0.77	0.82	Ensemble reduces variance, more stable predictions.

Model Type	Implementation	Precision	Recall	F1 Score	Accuracy	Notes
Gradient Boosting	scikit-learn	0.82	0.67	0.74	0.80	Boosting improves performance, focuses on difficult samples.
Adaboost Boosting	scikit-learn	0.78	0.70	0.74	0.79	Adaboost performed slightly less than Gradient boosting

6 Future Recommendations

1. **Advanced Data preprocessing & feature Engineering**

   In this Project I have implemented basic Encoding/scaling for this dataset it is optimal but when training larger datasets, it won't be sufficient so going beyond this we should implement domain specific feature creation and handling missing values using imputations methods like KNN imputer.

1. Hyperparameter Optimization

In the project, I have set hyperparameters for training models manually that involved self-experimenting since its not ideal for production level model training we can use hyperparameter optimization methods. Hyperparameters control how the model training happens (eg, learning rate, tree depth, regularization strength, max_depth etc).
As a solution we can implement Grid search or Random Search with cross validation to find optimal values for model hyperparameters. This leads to better performance when training models.

3 UI Development for Deployment

A valuable next step would be to design UI for this project so its ready to deployed into production level where users can see the difference between how each and every perform comparing scratch and library models visually through the application. This makes the models more accessible to non-technical users and enhances interpretability.

7 Conclusion

This project demonstrated the implementation and comparison of several supervised machine learning algorithms, both from scratch and using library-based methods, on the Titanic Survival dataset. Through systematic evaluation, it was observed that while models developed from scratch provided valuable learning on the inner workings of algorithms, the library implementations delivered higher efficiency, stability, and ease of experimentation.

Among logistic regression variants, L2 regularisation achieved the best overall performance, balancing accuracy and generalisation. Decision Trees offered interpretability but tended to overfit when made too deep, whereas bagging and boosting improved stability and predictive power through ensemble learning.

Overall, the findings highlight that no single approach is universally superior; rather, the choice depends on the trade-off between interpretability, computational cost, and predictive accuracy. The project also underlines the importance of preprocessing, regularisation, and ensemble methods in building robust machine learning models.