Predicting Titanic Survivors - A Machine Learning Regression Approach

Objective

The primary objective of this project is to develop a machine learning model that accurately predicts the survival of passengers on the Titanic based on available features such as age, sex, class, and other relevant attributes. By leveraging logistic regression, we aim to create a reliable predictive model that can assist in understanding the factors influencing survival rates during the Titanic disaster.

Overview

The Titanic dataset is one of the most famous datasets in the machine learning community, often used as a beginner's challenge in Kaggle competitions. It consists of various passenger attributes, including personal details and ticket information. The task is to predict whether a passenger survived the sinking of the Titanic based on these attributes.

This project involves using logistic regression, a linear classification model suitable for binary outcomes. The process includes loading and exploring the data, preprocessing and transforming it, training the model, and evaluating its performance using various metrics and visualizations.

Summary

In this project, we explored the Titanic dataset to predict passenger survival using a logistic regression model. The process involved several steps, including:

- **Data Loading and Exploration:** We loaded the training and testing datasets and explored them to understand their structure and content.
- **Data Preprocessing:** We addressed missing values, encoded categorical variables, and scaled the features to make them suitable for machine learning algorithms.
- **Feature Engineering:** We engineered features from the existing dataset to improve the model's performance.
- **Model Training:** We trained a logistic regression model and evaluated its performance using a validation set.

- **Model Evaluation:** We assessed the model's accuracy, precision, recall, and F1-score and visualized the results using confusion matrices and other plots.
- **Prediction and Submission:** We generated predictions on the test set and saved the results for submission.

Methodology

Step 1: Data Loading and Exploration

In this step, we loaded the training and testing datasets, including the gender_submission file, which provides a template for our submission. We used the pandas library to read the CSV files and explored the data to gain insights into its structure, identify missing values, and understand the distribution of various features.

Step 2: Data Preprocessing

Data preprocessing is a crucial step to handle missing values, encode categorical variables, and prepare the data for modeling. We performed the following operations:

- **Handling Missing Values:**
  - Filled missing 'Age' and 'Fare' values with the median.
  - Filled missing 'Embarked' values with the mode.
  - Dropped the 'Cabin' column due to the large number of missing values.
- **Encoding Categorical Variables:**
  - Used LabelEncoder to convert categorical variables ('Sex' and 'Embarked') into numerical values.
- **Dropping Irrelevant Columns:**
  - Removed columns like 'PassengerId', 'Name', and 'Ticket' as they do not contribute to the model's prediction.

Step 3: Feature Engineering

Feature engineering involves creating new features or transforming existing ones to enhance the model's predictive power. Although this step was not extensively covered in this project, it's worth mentioning the potential for improvement through:

- **Creating Interaction Features:** Combining features like 'Pclass' and 'Sex' to capture specific patterns.
- **Binning Continuous Features:** Binning 'Age' into categories (e.g., child, adult, elderly) to capture non-linear relationships.
- **Deriving New Features:** Using domain knowledge to create features such as family size from 'SibSp' and 'Parch.'

Step 4: Model Training

In this step, we trained a logistic regression model using the preprocessed training data. We split the data into training and validation sets to evaluate the model's performance before making predictions on the test set.

- **Splitting Data:** We used train_test_split to create an 80/20 split between training and validation sets.
- **Feature Scaling:** Applied StandardScaler to standardize the features for improved model performance.
- **Training the Model:** Used LogisticRegression to fit the model to the scaled training data.

Step 5: Model Evaluation

Model evaluation is essential to assess the performance of the trained model. We used several metrics and visualizations to understand how well the model predicts survival:

- **Accuracy Score:** Calculated the overall accuracy of the model.
- **Classification Report:** Included precision, recall, and F1-score for each class to provide a comprehensive evaluation.
- **Confusion Matrix:** Visualized the breakdown of true positives, true negatives, false positives, and false negatives.

- **ROC Curve and AUC:** Analyzed the trade-off between sensitivity and specificity.
- **Precision-Recall Curve:** Examined the relationship between precision and recall for different threshold values.

Step 6: Prediction and Submission

The final step involves generating predictions on the test set and saving the results for submission. We followed these steps:

- **Make Predictions:** Used the trained model to predict survival on the test set.
- **Prepare Submission File:** Created a DataFrame with 'PassengerId' and 'Survived' columns and saved it as a CSV file.

Conclusion

The logistic regression model demonstrated its effectiveness in predicting the survival of Titanic passengers, achieving a good balance between accuracy and interpretability. The project highlighted the importance of data preprocessing, feature engineering, and model evaluation in developing a reliable predictive model.

Key takeaways include:

- Handling missing values and encoding categorical variables are crucial for preparing the dataset for machine learning.
- Logistic regression, though simple, can be a powerful tool for binary classification problems.
- Model evaluation metrics such as accuracy, precision, recall, F1-score, ROC curve, and precision-recall curve provide insights into the model's performance.

Future Work

Future enhancements to this project could include:

- **Advanced Feature Engineering:** Creating more sophisticated features to capture complex relationships.
- **Hyperparameter Tuning:** Exploring different hyperparameters to improve the model's performance.

- **Exploring Other Models:** Investigating more complex models like Random Forest, Gradient Boosting, or Neural Networks to potentially achieve better accuracy.
- **Cross-Validation:** Using k-fold cross-validation to ensure robust evaluation across different subsets of the data.

References

- Kaggle Titanic Dataset: Kaggle Titanic Competition

CODE:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix, roc_curve, auc,
precision_recall_curve

from sklearn.model_selection import learning_curve


print("Step 1: Load and Explore the Datasets")


train_data = pd.read_csv("D:\\Aditya's Notes\\Titanic Project\\train.csv")


test_data = pd.read_csv("D:\\Aditya's Notes\\Titanic Project\\test.csv")


gender_submission = pd.read_csv("D:\\Aditya's Notes\\Titanic
Project\\gender_submission.csv")


# Display the first few rows of each dataset

print("Train Data:")

print(train_data.head(), "\n")
print("Test Data:")
print(test_data.head(), "\n")

print("Gender Submission Data:")
print(gender_submission.head(), "\n")

print("Step 2: Preprocess the Data")
'''
We'll preprocess the training and test datasets by handling missing values,
encoding categorical variables, and preparing the data for modeling.
'''



# Preprocess the train and test datasets
def preprocess_data(data):
    # Fill missing 'Age' values with the median
    data['Age'] = data['Age'].fillna(data['Age'].median())

    # Fill missing 'Embarked' values with the mode
    data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])

    # Fill missing 'Fare' values in the test set with the median
    if 'Fare' in data.columns:
        data['Fare'] = data['Fare'].fillna(data['Fare'].median())
```

```
# Drop the 'Cabin' column due to a large number of missing values
    data = data.drop(columns=['Cabin'])

    # Encode categorical variables using LabelEncoder
    label_encoders = {}
    for column in ['Sex', 'Embarked']:
        le = LabelEncoder()
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le

    # Drop irrelevant columns: 'PassengerId', 'Name', 'Ticket'
    data = data.drop(columns=['PassengerId', 'Name', 'Ticket'], errors='ignore')

    return data, label_encoders

# Preprocess the datasets
train_data, train_label_encoders = preprocess_data(train_data)
test_data, _ = preprocess_data(test_data)

# Define features and target variable
X_train = train_data.drop(columns=['Survived'])
y_train = train_data['Survived']

# Split the training data for evaluation purposes
X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(X_train,
y_train, test_size=0.2, random_state=42)
```

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_split)
X_val_scaled = scaler.transform(X_val_split)
X_test_scaled = scaler.transform(test_data)

print("Step 3: Train a Machine Learning Model")
'''
We'll use Logistic Regression for this binary classification
problem to predict the survival of passengers.
'''


# Create and train the logistic regression model
logistic_model = LogisticRegression(random_state=42, max_iter=1000)
logistic_model.fit(X_train_scaled, y_train_split)

# Make predictions on the validation set
val_predictions = logistic_model.predict(X_val_scaled)

print("Evaluate the model's performance")
# Evaluate the model's performance
accuracy = accuracy_score(y_val_split, val_predictions)
report = classification_report(y_val_split, val_predictions)
conf_matrix = confusion_matrix(y_val_split, val_predictions)
```

```
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
print("Confusion Matrix:")
print(conf_matrix)

#Visualization :

# Plotting the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Survived', 'Survived'], yticklabels=['Not Survived', 'Survived'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Calculate ROC curve and AUC
y_val_probabilities = logistic_model.predict_proba(X_val_scaled)[:, 1]
fpr, tpr, thresholds = roc_curve(y_val_split, y_val_probabilities)
roc_auc = auc(fpr, tpr)

# Plotting the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()

# Calculate Precision-Recall curve
precision, recall, pr_thresholds = precision_recall_curve(y_val_split, y_val_probabilities)

# Plotting the Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='green', lw=2, label='Precision-Recall Curve')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='lower left')
plt.show()

# Plotting Feature Importances
coefficients = logistic_model.coef_[0]
feature_names = X_train.columns

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})

# Sort by absolute value of coefficients
importance_df['Absolute Coefficient'] = np.abs(importance_df['Coefficient'])
importance_df.sort_values(by='Absolute Coefficient', ascending=False,
inplace=True)

plt.figure(figsize=(10, 6))
sns.barplot(x='Coefficient', y='Feature', data=importance_df, color='skyblue')  #
Using color instead of palette
plt.title('Feature Importance (Coefficient Plot)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()

# Generate learning curve data
train_sizes, train_scores, val_scores = learning_curve(
    logistic_model, X_train, y_train, cv=5, scoring='accuracy',
train_sizes=np.linspace(0.1, 1.0, 10), random_state=42)

# Calculate mean and standard deviation
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)

# Plotting the Learning Curve
```

```
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', label='Training Accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.15, color='blue')

plt.plot(train_sizes, val_mean, color='green', marker='s', label='Validation Accuracy')
plt.fill_between(train_sizes, val_mean - val_std, val_mean + val_std, alpha=0.15, color='green')

plt.title('Learning Curve')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.grid()
plt.show()

# Distribution of Predicted Probabilities
plt.figure(figsize=(10, 6))
sns.histplot(y_val_probabilities, bins=50, kde=True, color='purple')
plt.title('Distribution of Predicted Probabilities')
plt.xlabel('Predicted Probability of Survival')
plt.ylabel('Frequency')
plt.show()
```

```
# Make predictions on the test set
test_predictions = logistic_model.predict(X_test_scaled)

# Prepare a DataFrame with the results
submission_df = pd.DataFrame({
    'PassengerId': gender_submission['PassengerId'],
    'Survived': test_predictions
})

# Save the results to a CSV file
submission_df.to_csv("D:\\Aditya'sNotes\\Titanic
Project\\titanic_submission.csv", index=False)

print("Predictions saved to 'titanic_submission.csv'")

'''
```

Model Evaluation

To evaluate the model, we used the train_test_split function from Scikit-Learn, which divides the data into training and testing sets. We trained the model on the training set and evaluated it on the test set using metrics such as accuracy, precision, recall, F1-score, and a confusion matrix.

Explanation

- Data Preprocessing:
  - Missing values in Age, Embarked, and Fare are filled with median and mode values.
  - Categorical variables (Sex, Embarked) are encoded using LabelEncoder.

o The Cabin column is dropped due to many missing values, and irrelevant columns (PassengerId, Name, Ticket) are removed.

• Model Training:

o A Logistic Regression model is used to predict whether a passenger survived.

o We scale the features using StandardScaler to improve model performance.

• Prediction and Saving:

o Predictions are made on the test dataset, and results are saved in a CSV file named titanic_submission.csv.

Conclusion

This approach should give you a baseline model to predict Titanic survival. You can further improve the model by trying different algorithms, tuning hyperparameters, and engineering new features. If you have any further questions or need more specific insights, feel free to ask!

'''

'''

Explanation of Changes:

Train-Test Split:

I added a split for the training data to create a validation set. This split allows us to evaluate the model's performance on unseen data, which simulates the test environment better.

train_test_split() from sklearn.model_selection is used to split the data, with 80% for training and 20% for validation.

Metrics Calculation:

Accuracy Score: The overall accuracy of the model, calculated as the number of correct predictions divided by the total number of predictions.

Classification Report: This includes precision, recall, and F1-score for each class. These metrics are crucial for understanding how well the model performs on each class, especially in cases where the dataset might be imbalanced.

Confusion Matrix: Provides a detailed breakdown of true positives, true negatives, false positives, and false negatives, which helps to visualize the performance of the classification model.

Updated Training and Evaluation:

The logistic regression model is trained on the scaled training split and evaluated on the validation split. This step ensures that the evaluation metrics reflect the model's ability to generalize.

Predictions on Test Set:

The final predictions on the test set are made after evaluating the model on the validation set. This order helps ensure that the model's parameters are appropriately tuned before making the final predictions.

Saving Results:

The results are saved into a CSV file titanic_submission.csv, which includes the PassengerId and predicted Survived values, adhering to the submission format.

Key Considerations:

Cross-Validation: While splitting into train and validation sets provides a good estimate, using cross-validation (e.g., cross_val_score or GridSearchCV) might give a more robust understanding of model performance, especially for hyperparameter tuning.

Class Imbalance: The Titanic dataset might have an imbalance in the Survived class. Consider using techniques such as resampling, SMOTE, or adjusting class weights in the Logistic Regression model if performance issues persist.

Feature Engineering: Exploring additional features or transformations might improve performance. For example, creating new features from existing ones or using domain knowledge to engineer relevant attributes.

Regularization: If overfitting is suspected, consider using regularization techniques (L1 or L2) available in LogisticRegression to penalize large coefficients.

With these changes, you can effectively evaluate your logistic regression model and ensure that its predictions are reliable. Let me know if you need further assistance or have more questions!

'''

'''

To visualize and analyze the performance of your logistic regression model, you can create various plots and charts using popular Python libraries such as Matplotlib and Seaborn. These visualizations can help you gain deeper insights into the model's performance and identify areas for improvement. Here are some common types of charts that you can use to analyze your results:

1. Confusion Matrix Heatmap: A visual representation of the confusion matrix to see how well the model is predicting each class.
2. ROC Curve and AUC: To analyze the trade-off between sensitivity and specificity for binary classification.
3. Precision-Recall Curve: A plot that shows the relationship between precision and recall for different threshold values.
4. Feature Importance (Coefficient Plot): Visualizing the coefficients of the logistic regression model to understand which features contribute the most to predictions.
5. Learning Curve: To analyze how the model's performance changes with the size of the training set, which can help diagnose overfitting or underfitting.
6. Prediction Distribution: Visualizing the distribution of predicted probabilities for both classes.