UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

GROUP 17 MIDTERM REPORT

Members:

1. Name: Nguyễn Huy Hùng / ID: BI12-183
2. Name: Dương Hải Nam / ID: BI12-300
3. Name: Phùng Đức Minh / ID: BI12-277
4. Name: Đỗ Hữu Nam / ID: BI12-305
5. Name: Đỗ Trịnh Đại Hải / ID: BI12-147

Contents:

TOPIC: TITANIC DISASTER PREDICTION

I. Introduction to dataset

- The Titanic Passenger Survival Dataset is a collection of information about passengers aboard the Titanic, including whether they survived the disaster or not.
- We have 4 different datasets for training and testing the model:
  - For training the model:
    - **train_X** dataset contains 8 columns and 891 rows of data.
    - t**rain_Y** dataset contains 2 columns and 891 rows of data.
  - For testing the model:
    - **test_X** dataset contains 8 columns and 418 rows of data.
    - **test_Y** dataset contains 2 columns and 418 rows of data.

- X dataset (train_x, test_X) is a dataset containing the input features used to predict the output variable:
  - **Id:** A unique identifier for each passenger.
  - **PClass:** The passenger class, which is a proxy for socio-economic status:
    - 1 = 1st class (upper class)
    - 2 = 2nd class (middle class)
    - 3 = 3rd class (lower class)
  - **Sex:** The gender of the passenger:
    - 0 = Male
    - 1 = Female
  - **Age:** The age of the passenger
  - **SibSp:** The number of siblings and spouses the passengers had aboard the Titanic.
  - **Parch:** The number of parents and children the passenger had aboard the Titanic.
  - **Fare:** The fare paid by the passenger for the ticket.
  - **Embarked:** The port of embarkation:
    - 0 = Cherbourg
    - 1 = Queenstown
    - 2 = Southampton

Id	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	0	22	1	0	7.25	1
1	1	1	38	1	0	71.2833	0
2	3	1	26	0	0	7.925	1
3	1	1	35	1	0	53.1	1
4	3	0	35	0	0	8.05	1
5	3	0	20	0	0	8.4583	2
6	1	0	54	0	0	51.8625	1
7	3	0	2	3	1	21.075	1
8	3	1	27	0	2	11.1333	1
9	2	1	14	1	0	30.0708	0
10	3	1	4	1	1	16.7	1
11	1	1	58	0	0	26.55	1
12	3	0	20	0	0	8.05	1
13	3	0	39	1	5	31.275	1

- Y dataset (train_Y, test_Y) is a dataset containing the output variable that we want to predict based on the input features. In the Titanic survival prediction example, Y would represent whether a passenger survived (1) or did not survive (0) the Titanic disaster.

Id	Survived
0	0
1	1
2	1
3	1
4	0
5	0
6	0
7	0
8	1
9	1

II. Building Models.

- Our group performed two machine learning models: a Random Forest Classifier and a Logistic Regression model to predict whether the passengers survived the Titanic disaster or not.
- Below is a detailed breakdown of each section of the code:

A. Libraries:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, confusion_matrix, ConfusionMatrixDisplay

```

- The code uses several key Python libraries:
  - **NumPy** for numerical operations.
  - **Pandas** for data manipulation.
  - **Matplotlib** for data visualization.
  - **Scikit-Learn** for machine learning models and evaluation metrics.

B. Random Forest Classifier:

1. Data import:

```python
X_train_rf = pd.read_csv("train_X.csv").drop("Id", axis = 1).values
Y_train_rf = pd.read_csv("train_Y.csv").drop("Id", axis = 1).values.ravel()
X_test_rf = pd.read_csv("test_X.csv").drop("Id", axis = 1).values
Y_test_rf = pd.read_csv("test_Y.csv").drop("Id", axis = 1).values.ravel()

```

- Training and testing data for the Random Forest model are loaded from CSV files. The 'Id' column is dropped, and the data is converted to NumPy arrays. The target variables are flattened using **ravel()**.

2. Model Training

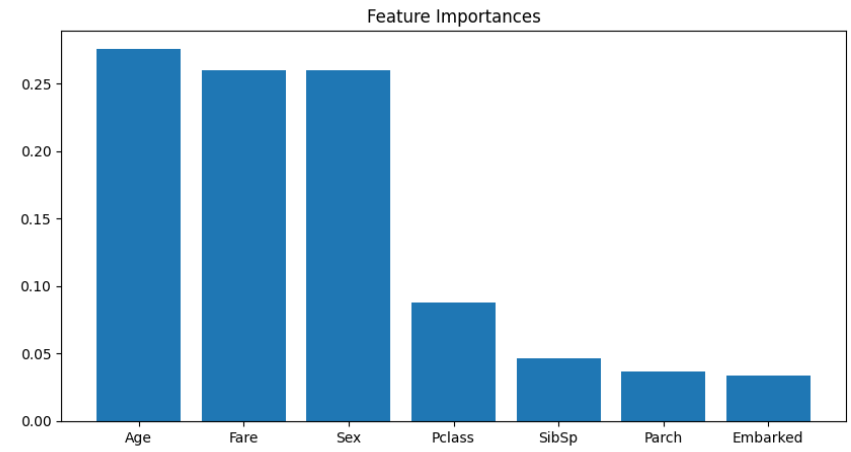
```
rf_model = RandomForestClassifier(n_estimators = 100, random_state = 42)
rf_model.fit(X_train_rf, Y_train_rf)
Y_pred_rf = rf_model.predict(X_test_rf)

```

- A Random Forest Classifier is instantiated with 100 estimators and trained on the training data. Predictions are made on the test data.

3. Feature Importance Plot

```
# Feature Importance Plot
feature_names = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]
importances = rf_model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 5))
plt.title("Feature Importances")
plt.bar(range(X_train_rf.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train_rf.shape[1]), [feature_names[i] for i in indices], rotation='horizontal')
plt.show()

```

- The feature importances from the Random Forest model are extracted and plotted to visualize which features contribute the most to the model's predictions.
- Importances Feature plotting:



C. Logistic Regression:

1. Data Preparation

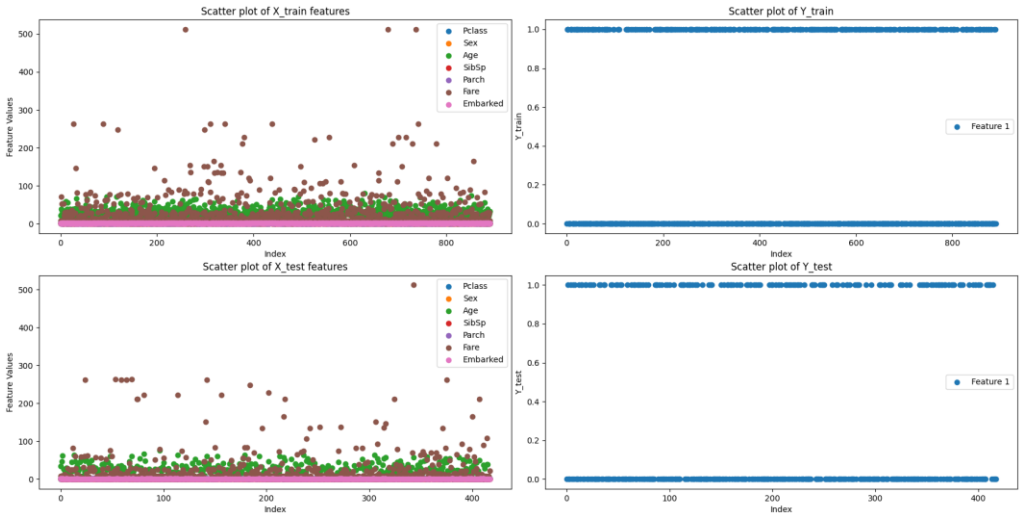
```
X_train_lr = pd.read_csv("train_X.csv").drop("Id", axis = 1).values.T
Y_train_lr = pd.read_csv("train_Y.csv").drop("Id", axis = 1).values.reshape(1, -1)
X_test_lr = pd.read_csv("test_X.csv").drop("Id", axis = 1).values.T
Y_test_lr = pd.read_csv("test_Y.csv").drop("Id", axis = 1).values.reshape(1, -1)

```

- Training and testing data for the Logistic Regression model are loaded similarly to the Random Forest data, but the feature matrices are transposed.

2. Data Visualization

```python
# Scatter Plots
def scatter_plot_data(X, Y, title, subplot_index, xlabel = "Index", ylabel = "Feature Values", labels = None):
    plt.subplot(2, 2, subplot_index)
    for i in range(X.shape[0]):
        plt.scatter(np.arange(X.shape[1]), X[i, :], label = labels[i] if labels else f"Feature {i+1}")
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()

plt.figure(figsize=(20, 10))
scatter_plot_data(X_train_lr, Y_train_lr, "Scatter plot of X_train features", 1, labels = feature_names)
scatter_plot_data(Y_train_lr, Y_train_lr, "Scatter plot of Y_train", 2, ylabel = "Y_train")
scatter_plot_data(X_test_lr, Y_test_lr, "Scatter plot of X_test features", 3, labels = feature_names)
scatter_plot_data(Y_test_lr, Y_test_lr, "Scatter plot of Y_test", 4, ylabel = "Y_test")
plt.tight_layout()
plt.show()

```

- Several scatter plots are created to visualize the features and target variables for both the training and test datasets.
- Data visualize:



3. Sigmoid Function

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

```

- The sigmoid function is defined, which will be used to map predictions to probabilities.

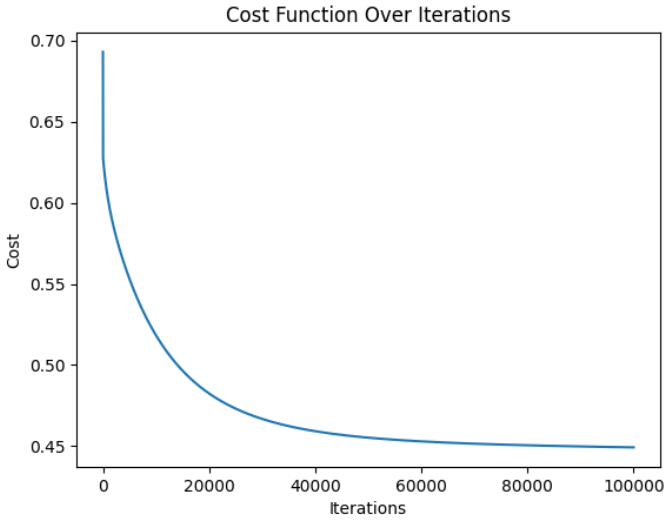
4. Model Training

```python
def model_lr(X, Y, learning_rate, iterations):
    m, n = X.shape[1], X.shape[0]
    W, B = np.zeros((n, 1)), 0
    cost_list = []
    for i in range(iterations):
        Z = np.dot(W.T, X) + B
        A = sigmoid(Z)
        # Cost Function
        cost = -(1/m) * np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A))
        # Gradient Descent
        W -= learning_rate * (1/m) * np.dot(A - Y, X.T).T
        B -= learning_rate * (1/m) * np.sum(A - Y)
        # Tracking cost value over iterations
        if i % (iterations // 10) == 0:
            print(f"Cost after {i} iterations: {cost}")
        cost_list.append(cost)

    return W, B, cost_list

W_lr, B_lr, cost_list_lr = model_lr(X_train_lr, Y_train_lr, 0.001, 100000)

# Plot cost function
plt.plot(np.arange(100000), cost_list_lr)
plt.title("Cost Function Over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.show()

```

- A function to train the Logistic Regression model is defined.
- The weights and bias are initialized to zero, and gradient descent is used to optimize the cost function.
- The cost is printed every 10% of the total iterations:

```
Cost function in Linear Regression model after 0 iterations: 0.6931471805599454
Cost function in Linear Regression model after 10000 iterations: 0.518318504952069
Cost function in Linear Regression model after 20000 iterations: 0.48245351031982525
Cost function in Linear Regression model after 30000 iterations: 0.46674899820135907
Cost function in Linear Regression model after 40000 iterations: 0.4591687984641674
Cost function in Linear Regression model after 50000 iterations: 0.45517763749795936
Cost function in Linear Regression model after 60000 iterations: 0.45289001499072096
Cost function in Linear Regression model after 70000 iterations: 0.45146059794861
Cost function in Linear Regression model after 80000 iterations: 0.45048756833994
Cost function in Linear Regression model after 90000 iterations: 0.4497708938942782
```

- The cost function's value over iterations is plotted to visualize the convergence of the model training process:

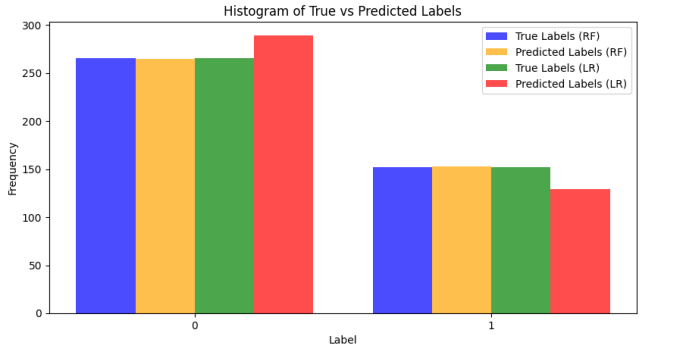


D. Model evaluation:

1. Predictions and Visualization

```python
# Predictions
def predict_lr(W, B, X):
    Z = np.dot(W.T, X) + B
    A = sigmoid(Z)
    return (A > 0.5).astype(int)
predictions_lr = predict_lr(W_lr, B_lr, X_test_lr)

# Visualize the True vs Predicted Labels for Each Model
plt.figure(figsize = (10, 5))
plt.hist([Y_test_rf, Y_pred_rf, Y_test_lr.flatten(), predictions_lr.flatten()],
         bins = np.arange(-0.5, 2, 1),
         label = ["True Labels (RF)", "Predicted Labels (RF)", "True Labels (LR)", "Predicted Labels (LR)"],
         alpha = 0.7, color=["blue", "orange", "green", "red"])
plt.xticks([0, 1])
plt.xlabel("Label")
plt.ylabel("Frequency")
plt.title("Histogram of True vs Predicted Labels")
plt.legend(loc = "upper right")
plt.show()

```

- A prediction function is defined to implement a simple binary classifier.
- A histogram is plotted to compare the true labels and predicted labels for both the Random Forest and Logistic Regression models.
- This visualization helps in understanding the performance and distribution of predictions for both models.
- The histogram:

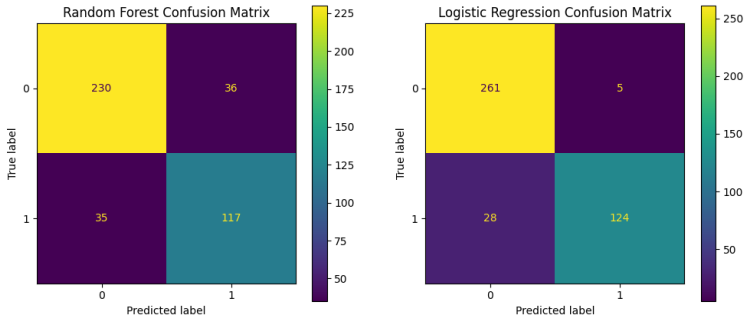


2. Evaluation Metrics

a. Confusion Matrix

```
cm_rf = confusion_matrix(Y_test_rf, Y_pred_rf)
cm_lr = confusion_matrix(Y_test_lr.flatten(), predictions_lr.flatten())

fig, ax = plt.subplots(1, 2, figsize = (12, 5))
ConfusionMatrixDisplay(cm_rf, display_labels = [0, 1]).plot(ax = ax[0])
ax[0].set_title("Random Forest Confusion Matrix")
ConfusionMatrixDisplay(cm_lr, display_labels = [0, 1]).plot(ax = ax[1])
ax[1].set_title("Logistic Regression Confusion Matrix")
plt.show()
```

- Confusion matrices for both models are generated and displayed.
- These matrices provide insights into the performance of each model by showing the counts of true positive, true negative, false positive, and false negative predictions:



b. Precision, Recall, F-score, Specificity, and Accuracy

```python
def specificity(cm):
    tn, fp, fn, tp = cm.ravel()
    return (tn / (tn + fp)) * 100

def accuracy_lr(X, Y, W, B):
    predictions = predict_lr(W, B, X)
    return np.mean(predictions == Y) * 100

# Random Forest Metrics
precision_rf = precision_score(Y_test_rf, Y_pred_rf) * 100
recall_rf = recall_score(Y_test_rf, Y_pred_rf) * 100
f1_rf = f1_score(Y_test_rf, Y_pred_rf) * 100
specificity_rf = specificity(cm_rf)
accuracy_rf = accuracy_score(Y_test_rf, Y_pred_rf) * 100

# Logistic Regression Metrics
precision_lr = precision_score(Y_test_lr.flatten(), predictions_lr.flatten()) * 100
recall_lr = recall_score(Y_test_lr.flatten(), predictions_lr.flatten()) * 100
f1_lr = f1_score(Y_test_lr.flatten(), predictions_lr.flatten()) * 100
specificity_lr = specificity(cm_lr)
accuracy_lr = accuracy_lr(X_test_lr, Y_test_lr.flatten(), W_lr, B_lr)

# Print Metrics
print("---")
print(f"""Random Forest Metrics:
      - Precision: {precision_rf:.2f}%
      - Recall: {recall_rf:.2f}%
      - F-Score: {f1_rf:.2f}%
      - Specificity: {specificity_rf:.2f}%
      - Accuracy: {accuracy_rf:.2f}%""")
print("---")
print(f"""Logistic Regression Metrics:
      - Precision: {precision_lr:.2f}%
      - Recall: {recall_lr:.2f}%
      - F-Score: {f1_lr:.2f}%
      - Specificity: {specificity_lr:.2f}%
      - Accuracy: {accuracy_lr:.2f}%""")

```

- Specificity Calculation: We have a function to calculate the specificity metric for both models from a confusion matrix.
- An accuracy function is defined to evaluate the Logistic Regression model, predictions are made using the learned weights and bias, and accuracy is calculated by comparing predictions to the true labels.
- Precision, recall, F-score, Specificity and Accuracy are calculated and printed for both models to provide a comprehensive evaluation:
- Printing Metrics:

```
Random Forest Metrics:
      - Precision: 76.47%
      - Recall: 76.97%
      - F-Score: 76.72%
      - Specificity: 86.47%
      - Accuracy: 83.01%
---
Logistic Regression Metrics:
      - Precision: 96.12%
      - Recall: 81.58%
      - F-Score: 88.26%
      - Specificity: 98.12%
      - Accuracy: 92.11%
```

III. Synthesize and analyze.

- Comparing the performance metrics of the Logistic Regression and Random Forest models reveals interesting insights into their strengths and weaknesses.

1. **Model Metrics**

    - **Precision, Recall, F-Score, Specificity and Accuracy:**
        - **Logistic Regression:**
            - **Precision:** 96.12%
            - **Recall:** 81.58%
            - **F-Score:** 88.26%
            - **Specificity:** 98.12%
            - **Accuracy:** 92.11%.
        - **Random Forest:**
            - **Precision:** 76.47%
            - **Recall:** 76.97%
            - **F-Score:** 76.72%
            - **Specificity:** 86.47%
            - **Accuracy:** 83.01%.

2. **Analysis**
    - **Logistic Regression Dominates Precision:**
        - The Logistic Regression model exhibits higher precision compared to the Random Forest model. This suggests that when it predicts positive instances, it is more likely to be correct.
    - **Recall Superiority of Linear Regression:**
        - Linear Regression shows more advantages in recall. This implies that Linear Regression tends to capture a higher proportion of actual positive instances.
    - **Balanced F-Score of Logistic Regression:**
        - Logistic Regression achieves a higher F1 score, indicating a better balance between precision and recall compared to Random Forest.
    - **Logistic Regression Outperforms Random Forest:**
        - Logistic Regression shows higher specificity and accuracy compared to Random Forest. Higher specificity means Logistic Regression is better at correctly identifying negative instances, while higher accuracy indicates better overall performance in correctly classifying both positive and negative instances.
    - **Overfitting and Underfitting:**
        - Logistic Regression is a simpler model and less prone to overfitting, especially with a limited amount of data. Random Forests can overfit if not properly regularized or if the data is not sufficient to capture the complexity of the model.

IV. Conclusion

- Overall, the code provides a comprehensive analysis of two machine learning models, showcasing their strengths and weaknesses in terms of predictive accuracy and other evaluation metrics. It also demonstrates how to visualize and interpret the results to gain insights into model performance and behavior.
- Therefore, based on these synthesize and analyze factors, it can be concluded that Logistic Regression demonstrates superior performance across all evaluated metrics. It achieves higher precision, recall, F-Score, specificity, and accuracy, making it a more effective model for correctly classifying instances in this dataset. Additionally, its simplicity reduces the risk of overfitting, further enhancing its reliability compared to the more complex Random Forest model.

IV. References

https://github.com/Jaimin09/Coding-Lane-Assets/tree/main/Logistic%20Regression%20in%20Python%20from%20Scratch

- We have changed a bit.