Report on Logistic Regression for Diabetes Prediction

1. Introduction

Diabetes is a chronic disease that affects millions worldwide. Early detection and prediction play a crucial role in preventive healthcare. In this project, we use a medical dataset to build and evaluate a logistic regression model for predicting the likelihood of diabetes.

The dataset is a processed version of the Pima Indians Diabetes Dataset, with the following features:
- times_pregnant – Number of times the patient has been pregnant
- glucose_conc – Plasma glucose concentration
- Diastolic_BP – Diastolic blood pressure
- Triceps_thk – Triceps skin fold thickness
- 2_hr_insulin – 2-hour serum insulin level
- BMI – Body mass index
- Pedigree – Diabetes pedigree function (family history)
- Age – Patient's age
- Diabetes (target variable) – Encoded as 1 (diabetic) and 0 (non-diabetic)

The target variable was originally encoded as -1 and 1, which caused issues with binary classification metrics. We resolved this by remapping -1 to 0.

2. Exploratory Data Analysis (EDA)

Steps Performed:
1. Data Loading: The dataset was loaded into pandas from '../Data/proc_pima_2_withheader.csv_preprocessed.csv'.
2. Missing Values: No missing values were present.
3. Statistical Summary: We inspected mean, median, standard deviation, and ranges to identify scale differences.
4. Feature Scaling: Features were standardized using StandardScaler.
5. Target Distribution: Balanced representation ensured fair evaluation.
6. Feature Engineering: Final dataset consisted of scaled features + binary target.

Exploratory Data Analysis (EDA)

Summary statistics

```python
print("Summary Statistics:")
print(df.describe())
```

[4]

```
Summary Statistics:
         Diabetes  times_pregnant  glucose_conc  Diastolic_BP  Triceps_thk
count  394.000000      394.000000    394.000000    394.000000   394.000000
mean    -0.335025        3.302030    122.380711     70.675127    29.137056
std      0.943407        3.208235     31.422626     12.465567    10.503919
min     -1.000000        0.000000      0.000000     24.000000     7.000000
25%     -1.000000        1.000000     99.000000     62.000000    21.000000
50%     -1.000000        2.000000    119.000000     70.000000    29.000000
75%      1.000000        5.000000    143.000000     78.000000    36.750000
max      1.000000       17.000000    198.000000    110.000000    63.000000

       2_hr_insulin         BMI    Pedigree         Age
count    394.000000  394.000000  394.000000  394.000000
mean     155.322335   33.073858    0.522741   30.888325
std      118.987181    7.015055    0.344833   10.232549
min        0.000000   18.200000    0.085000   21.000000
25%       76.000000   28.400000    0.270250   23.000000
50%      125.000000   33.200000    0.449500   27.000000
75%      190.000000   37.075000    0.685750   36.000000
max      846.000000   67.100000    2.420000   81.000000
```

3. Logistic Regression Implementation

We implemented logistic regression using gradient descent optimization from scratch. The main steps were:
- Sigmoid Function: To map linear combinations of inputs to probabilities.
- Loss Function: Binary cross-entropy loss was minimized.
- Gradient Descent: Used to iteratively update weights until convergence.
- Thresholding: Predictions converted to 0 or 1 based on a 0.5 cutoff.

The scratch implementation was compared against scikit-learn's LogisticRegression for validation.

```python
class LogisticRegressionScratch:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = self.sigmoid(linear_model)

            dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
            db = (1 / n_samples) * np.sum(y_pred - y)

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict_proba(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        return self.sigmoid(linear_model)

    def predict(self, X):
        y_pred_prob = self.predict_proba(X)
        return np.where(y_pred_prob >= 0.5, 1, 0)
```

4. Model Evaluation

We split the dataset into 80% training and 20% testing. Both implementations were evaluated on accuracy, precision, recall and F1 score

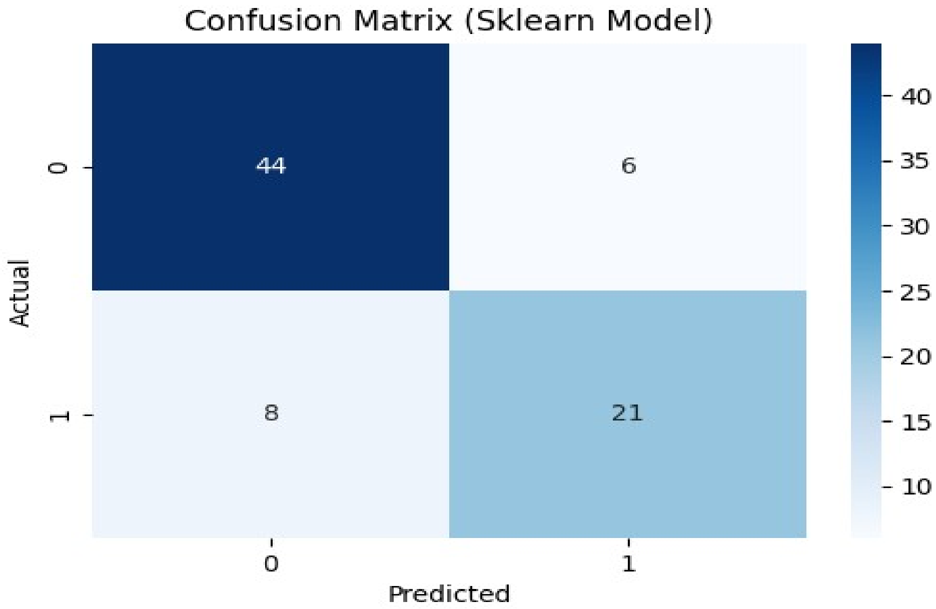
Results:

[Scratch Logistic Regression]
- Accuracy: 0.8228
- Precision: 0.7778
- Recall: 0.7241
- F1 Score: 0.7500

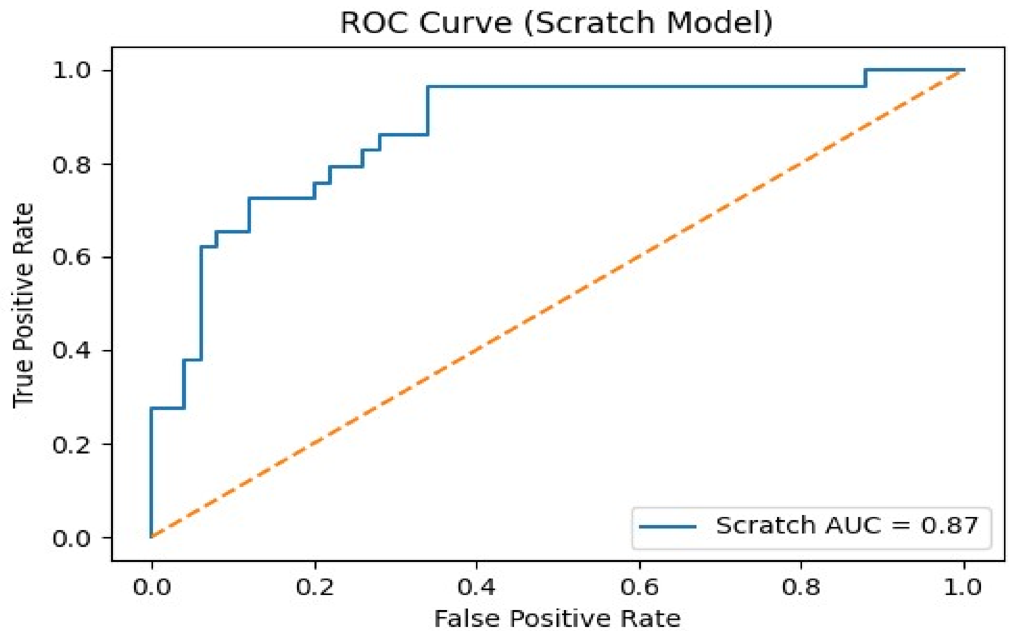
[Scikit-learn Logistic Regression]
- Accuracy: 0.8228
- Precision: 0.7778
- Recall: 0.7241
- F1 Score: 0.7500

The results were identical, confirming the correctness of our scratch implementation.

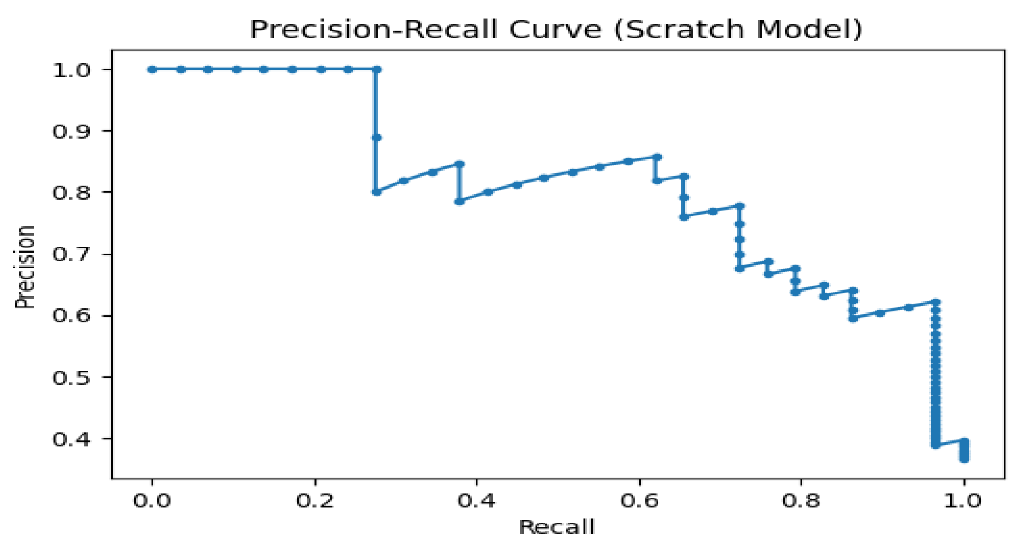
- Confusion Matrix



- ROC Curve



- Precision-Recall Curve



5. Conclusion

The project demonstrated that logistic regression can be successfully implemented from scratch and achieve the same performance as scikit-learn's library implementation.

Key takeaways:
- Proper preprocessing (feature scaling, binary target encoding) is essential.
- Gradient descent optimization produced stable convergence.
- Evaluation metrics showed good predictive performance (~82% accuracy).
- Confusion matrix and ROC/PR curves confirmed balanced trade-offs between precision and recall.

This project reinforces logistic regression as a robust baseline classifier for binary medical datasets and provides a foundation for experimenting with more advanced machine learning algorithms.