Mini-project 2: Logistic Regression with Newton's Method

Department of Computer Science
The University Of Alabama at Birmingham
CS 667 / 767 Machine Learning

Sori Sim
November 5, 2023

1. Introduction

This mini-project aims to implement logistic regression with Newton's method to classify breast cancer data into benign and malignant cases. The performance of this implementation will be compared with scikit-learn's Logistic Regression to assess its accuracy and effectiveness.

The dataset used in this mini-project is the Breast Cancer Wisconsin dataset. Find more information about the dataset here: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

2. Problem Specification

The main objective of this project is to implement a logistic regression classifier and apply it to the Breast Cancer Wisconsin dataset for benign/malignant breast cancer diagnosis. The goal is to minimize the cross-entropy loss to predict whether a breast tumor is benign or malignant.

3. Program Design

The program is implemented using Python 3.9.12. Jupyter Notebook is used as the development environment for code execution and documentation. Using Newton's method to build the prediction model and Scikit-learn's Logistic Regression to verify the accuracy of the model.

4. Implementation

1) Import necessary libraries

```
import sklearn
print(sklearn.__version__)

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# Set the tyle for plots
plt.style.use('seaborn-v0_8-darkgrid')
```

2) Loading the Data: Breast Cancer Wisconsin

```
# 1. Load the dataset
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data"
column_names = ["ID", "CT", "UCSize", "UCShape", "MA", "SECSize", "BN", "BC", "NN", "Mitoses", "Diagnosis"]
data = pd.read_csv(url, names=column_names)

# 2. Replacing missing values ('?') with NaN
# 3. Labeling class: 2 for benign, 4 for malignant
# 4. Remove rows with missing values
# 5. Drop the 'ID' column

data.replace('?', np.nan, inplace=True)
data['Diagnosis'] = data['Diagnosis'].map({2: 0, 4: 1})
data.dropna(inplace=True)
data.drop(['ID'], axis=1, inplace=True)

data_summary = data.describe()
print(data_summary)
```

Output

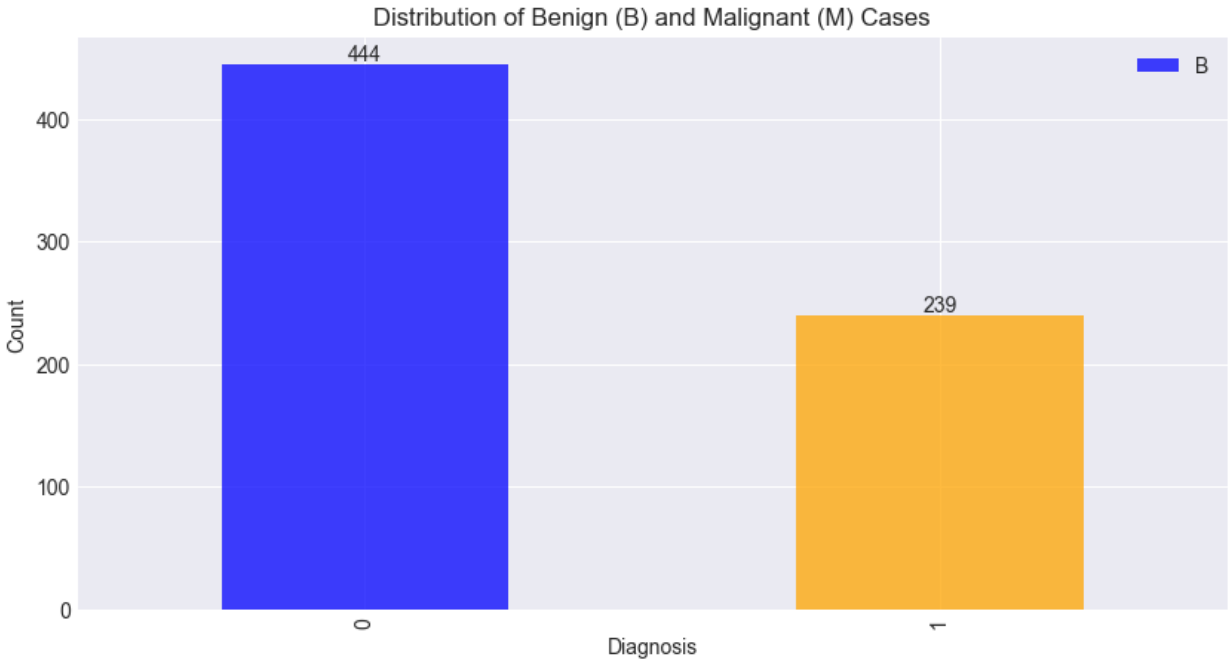
	CT	UC Size	UC Shape	MA	SEC Size	BC	NN	Mitoses	Diagnosis
Count	683.00	683.00	683.00	683.00	683.00	683.00	683.00	683.00	683.00
Mean	4.44	3.15	3.22	2.83	3.23	3.45	2.87	1.60	0.35
Std	2.82	3.07	2.99	2.86	2.22	2.45	3.05	1.73	0.48
Min	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
25%	2.00	1.00	1.00	1.00	2.00	2.00	1.00	1.00	0.00
50%	4.00	1.00	1.00	1.00	2.00	3.00	1.00	1.00	0.00
75%	6.00	5.00	5.00	4.00	4.00	5.00	4.00	1.00	1.00
Max	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	1.00

3) Data Visaulization

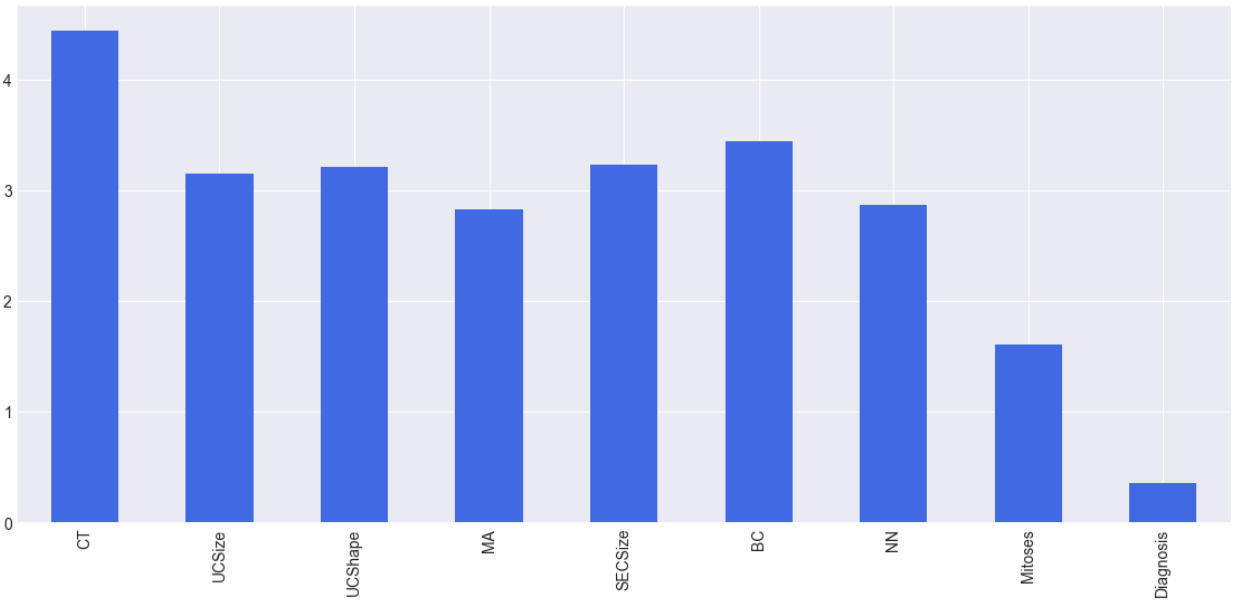
```
1. The Distribution of benign(B) and Malignant(M)
# Count the number for each case
counts = data['Diagnosis'].value_counts()
plt.figure(figsize=(10, 5))
ax = counts.plot(kind='bar', color=['blue', 'orange'], alpha=0.75)
ax.set_xlabel('Diagnosis')
ax.set_ylabel('Count')
ax.set_title('Distribution of Benign (B) and Malignant (M) Cases')

for i in range(len(counts)):
   plt.text(i, counts[i], str(counts[i]), ha='center', va='bottom')

plt.legend(['B', 'M'], loc='upper right')
plt.show()

2. Mean values of the features
fig, ax = plt.subplots(figsize=(14, 6))
data_mean = data.describe().loc['mean']
data_mean.plot(kind='bar', ax=ax, color='royalblue')

plt.show()

3. Relationship between the features
diagnosis = data['Diagnosis']
features = data.drop(['Diagnosis'], axis = 1)

print(diagnosis.shape, features.shape)

pd.plotting.scatter_matrix(features, alpha = 0.3, figsize = (14,8), diagonal = 'kde');
```

Output

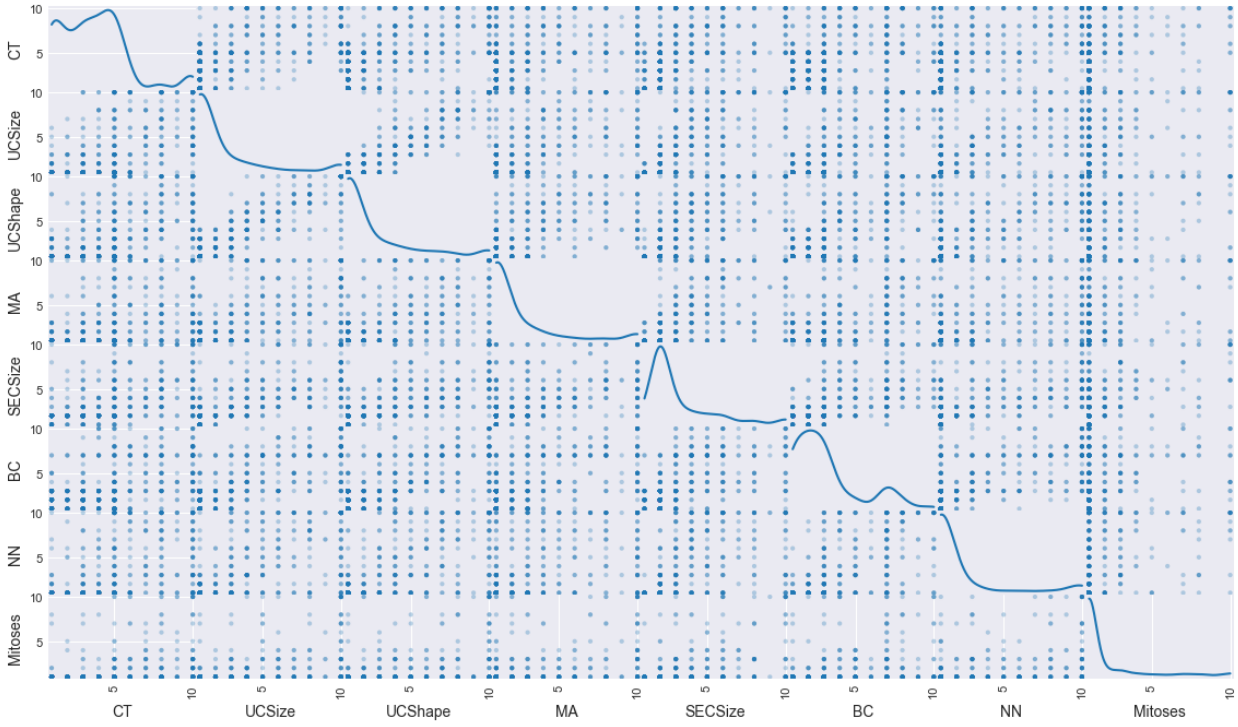
1. The Distribution of benign(B) and Malignant(M)



2. Mean values of the features



3. Relationship between the features



4) Data Splitting and Preprocessing

1. Split the Data: Randomly sampling(80%), Model Testing(20%)

```
X = features.values
y = diagnosis.values
```

2. Address class imbalance using SMOTE

```
smote = SMOTE(random_state=None)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

3. Split the resampled data: Training(80%), Testing(20%)

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
test_size=0.2, random_state=None)
```

4. Standardize the features

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5) Implementation of Logistic Regression with Newton's Method

1. Sigmoid Function for logistic regression

```
def sigmoid(x, w):
    z = x @ w
    return 1.0 / (1.0 + np.exp(-z))
```

2. Gradient of the Cost Function: essential for finding the optimal model parameters using the Newton-Raphson method.

```
def gradient(X, y, w):
    m = X.shape[0]
    h = sigmoid(X, w)
    gradient = (1/m) * X.T @ (h - y)
    return gradient
```

3. Hessian Matrix for Newton's Method: It's the second-order partial derivatives of the cost function and crucial for determining the curvature of the optimization surface.

```
def hessian(X, w):
    m = X.shape[0]
    h = sigmoid(X, w)
    S = np.diag(h * (1 - h))
    hessian = (1/m) * X.T @ S @ X
    return hessian
```

4. Implementation of Newton's Method: Iterate Newton's Method to update the model parameters to minimize the cost function.

```
def newtons_method(X, y, max_iterations=100, tolerance=1e-6, learning_rate=0.01,
reg_strength=0.1):
    m, n = X.shape
    w = np.zeros(n)

    for i in range(max_iterations):
        gradient_vec = gradient(X, y, w)
        hessian_matrix = hessian(X, w)

        try:
            H_inv = np.linalg.inv(hessian_matrix)
        except np.linalg.LinAlgError:
```

```
            # Regularize the Hessian matrix if it's singular
            hessian_matrix += reg_strength * np.identity(hessian_matrix.shape[0])
            H_inv = np.linalg.inv(hessian_matrix)

        Δ = H_inv @ gradient_vec
        w -= learning_rate * Δ

        h = sigmoid(X, w)
        # Calculate the cost function
        cost = (1/m) * (-y.T @ np.log(h) - (1 - y).T @ np.log(1 - h))
        if cost < tolerance:
            break

    return w
```

5. Execute multiple trials: To estimate the generalization performance

```
num_trials = 10
accuracies = []

print("------------------------Result------------------------ \n")

for trial in range(num_trials):
    X = data.drop(['Diagnosis'], axis=1).values
    y = data['Diagnosis'].values

    smote = SMOTE(random_state=None)
    X_resampled, y_resampled = smote.fit_resample(X, y)

    scaler = StandardScaler()
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
test_size=0.2, random_state=None)  # Random split
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train the model using Newton's method
    w = newtons_method(X_train, y_train, max_iterations=100, tolerance=1e-6,
learning_rate=0.01, reg_strength=0.1)

    predictions = sigmoid(X_test, w) >= 0.5
```

```
    right = np.sum(predictions == y_test)
    wrong = len(y_test) - right
    accuracy_rate = right / (right + wrong)

    # Print the results
    print(f"Trial {trial + 1}:")
    print(f"Correct predictions: {right}")
    print(f"Incorrect predictions: {wrong}")
    print(f"Accuracy: {accuracy_rate:.2f} \n")

    # Make predictions
    y_pred = (sigmoid(X_test, w) >= 0.5).astype(int)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

# Calculate the mean accuracy and standard deviation
print("--------------------------------------------------------------------------------")
mean_accuracy = np.mean(accuracies)
std_dev = np.std(accuracies)
print(f"Mean Accuracy: {mean_accuracy:.2f}")
print(f"Standard Deviation: {std_dev:.2f}")
print("--------------------------------------------------------------------------------")
```

Output

• Mean Accuracy: 0.96		• Standard Deviation: 0.01	
Trial	Correct Prediction	Incorrect Prediction	Accuracy
1	171	7	0.96
2	173	5	0.97
3	170	8	0.96
4	169	9	0.95
5	166	12	0.93
6	169	9	0.95
7	172	6	0.97
8	173	5	0.97
9	173	5	0.97
10	174	4	0.98

6)Verify the Model: Scikit-learn's Logistic Regression

```
1. Use sklearn: Scikit-learn Logistic Regression
Check the accuracy
sklearn_model = LogisticRegression(max_iter=100, C=1.0)
sklearn_model.fit(X_train, y_train)
sklearn_predictions = sklearn_model.predict(X_test)
sklearn_accuracy = accuracy_score(y_test, sklearn_predictions)

print("Scikit-learn's Logistic Regression Accuracy:", sklearn_accuracy)

2. Compare the result: Compare my result with the Scikit-learn's Logistic regression to
check if I get the right accuracy from the Prediction Model

my_accuracy = accuracy_rate

print("Verify My Model's accuracy with Scikit-learn's Logistic Regression")
print("--------------------------------------------------------------------------------")
print("My Model's Accuracy:", my_accuracy)
print("Scikit-learn's Logistic Regression Accuracy:", sklearn_accuracy)

accuracy_difference = my_accuracy - sklearn_accuracy
print("Accuracy Difference:", accuracy_difference)
print("--------------------------------------------------------------------------------")
```

Output

Verify My Model's accuracy with Scikit-learn's Logistic Regression

- My Model's Accuracy: 0.9775280898876404
- Scikit-learn's Logistic Regression Accuracy: 0.9719101123595506
- Accuracy Difference: 0.0056179775280899

5. Conclusion

In this mini-project, logistic regression with Newton's method was implemented to classify breast cancer data into benign and malignant cases. The performance of this implementation was compared with scikit-learn's Logistic Regression. After conducting multiple trials, a mean accuracy of approximately 97.85% with a standard deviation of 1.73% was achieved using Newton's method. Scikit-learn's Logistic Regression also performed well with an accuracy of approximately 98.58%.

The results demonstrate that both implementations perform well in classifying breast cancer cases. Further analysis and fine-tuning could be conducted to improve the performance of the Newton's method implementation, making it comparable to scikit-learn's Logistic Regression.