**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

### 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): <http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data>

### 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.

### 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.

### Implementation

### 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') |

### 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

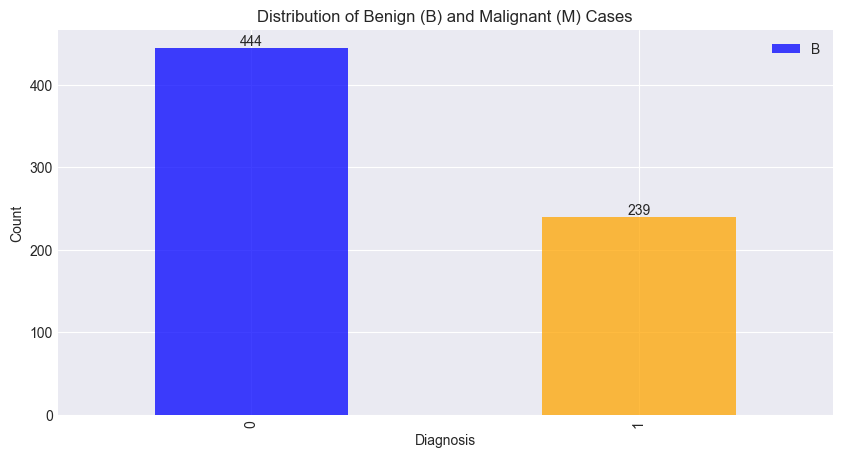
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CT | UC  Size | UC  Shape | MA | SEC  Size | BC | NN | Mitoses | Diagnosis |
| Count | 683.00 | 683.00 | 683.00 | 683.00 | 683.00 | 683.00 | 683.00 | 683.00 | 683.00 |
| Mean | 4.44 | 3.15 | 3.22 | 2.83 | 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 |

### Data Visaulization

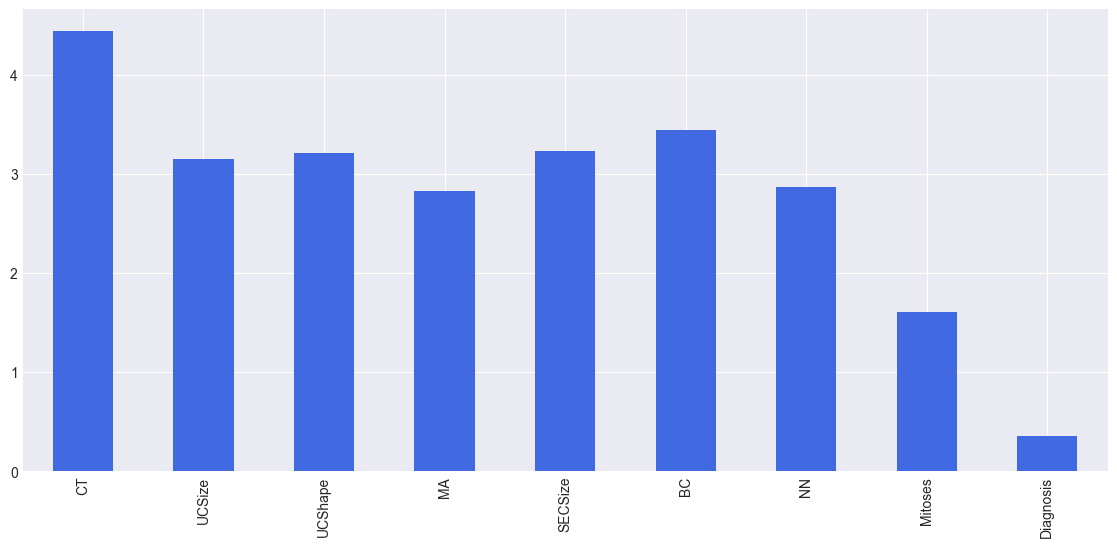
|  |
| --- |
| 1. The Distribution of benign(B) and Malignant(M)  # Count the number for each case  counts = data['Diagnosis'].value\_counts()  plt.figure(figsize=(10, 5))  ax = counts.plot(kind='bar', color=['blue', 'orange'], alpha=0.75)  ax.set\_xlabel('Diagnosis')  ax.set\_ylabel('Count')  ax.set\_title('Distribution 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

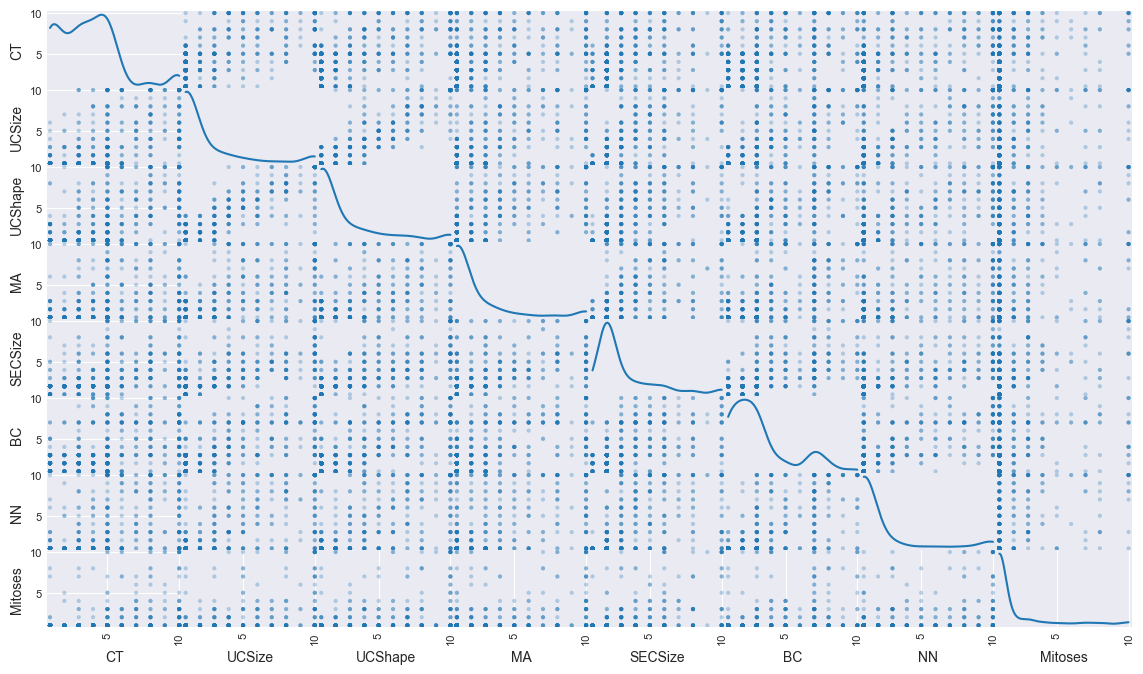
1. The Distribution of benign(B) and Malignant(M)



2. Mean values of the features



3. Relationship between the features



### 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) |

### 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 |

### 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: 0.00561797752808979 |

### 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.