Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Learning from Data and Related Challenges and Classification

Abeysinghe D.U. 210011X

Submitted in partial fulfillment of the requirements for the module
EN 3150 Pattern Recognition

2nd October 2024

1 Logistic Regression

1.1 Load the data

1.2 Purpose of `"y_encoded = le.fit_transform(df_filtered['species'])"`

This line of code converts the categorical values in the 'species' column of the DataFrame `df_filtered` into numerical format. This is done using the label encoding.
The method `fit_transform()` identifies the unique values in the species column and assigns numerical values to them.

1.3 Purpose of `"X = df.drop(['species','island','sex'],axis=1)"`

This code is used to drop the features 'species', 'island' and 'sex' from the DataFrame. Since we are encoding the 'species' feature using a label encoder, we don't need the feature itself to perform the regression.

1.4 Why we cannot use 'island' and 'sex' features?

The features 'island' and 'sex' are categorical data. If we are to use them in regression, they have to be encoded.

1.5 Training the logistic regression model using the code in listing 2

The following output was observed.

```
Accuracy: 0.5813953488372093
[[ 2.75979760e-03 -8.34554386e-05  4.61284208e-04 -2.86548082e
   -04]] [-8.51898705e-06]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
   _sag.py:349: ConvergenceWarning: The max_iter was reached
   which means the coef_ did not converge
  warnings.warn(
```

1.6 What is the usage of `"random_state=42"`?

Using this argument allows us to get the same train and test data sets regardless of the number of times the code is run. This is helpful in comparing model performance with different models.

1.7 Why does the saga solver perform poorly?

According to the output we cans see that the solver has not converged to the optimum point (global minimum). According to the documentation the convergence of 'saga' solver is only guaranteed on features with the same scale. This can be the main reason that this model doesn't converge to an optimum point.

1.8 Using the 'liblinear' solver

The following output was observed.

```
Accuracy: 1.0
[[ 1.45422752 -0.93943994 -0.16571368 -0.00398663]]
   [-0.04793176]
```

According to the output, the accuracy is 1.0 .

1.9 Why does the 'liblinear' solver perform better than the 'saga' solver?

The liblinear solver is tailored for small to medium-sized datasets and performs better in binary or multinomial logistic regression scenarios. It employs a coordinate descent approach, which is effective for these sizes and achieves efficient convergence. Since the dataset is neither large nor sparse, the liblinear solver is more suitable. In contrast, the saga solver is a stochastic method designed for large-scale problems, and it tends to underperform on smaller datasets due to potentially slower or less efficient convergence.

1.10 Performance of 'liblinear' and 'saga' with feature scaling

The following code was used to scale the train and test datasets and then perform regression with the 'saga' solver.

```
from sklearn.preprocessing import StandardScaler

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size
    =0.2, random_state = 42)

#re-train the logistic regression model saga using standard
    scaling

scaler = StandardScaler()

logreg = LogisticRegression(solver='saga')

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

logreg.fit(X_train_scaled, y_train)

y_pred = logreg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:" , accuracy)

print(logreg.coef_, logreg.intercept_)
```

The following output was observed after this code was run.

```
Accuracy: 0.9767441860465116
[[ 3.90453326 -0.82352552  0.18541027 -0.73564462]]
    [-1.96737274]
```

The obtained accuracy is 0.9767 which is a significantly high amount compared to the initial accuracy value of 0.5814. The main reason for this change can be seen as the successful convergence of the 'saga' solver after the dataset was scaled.
Similarly, we can check the accuracy of the 'liblinear' solver after scaling as well.The following output was observed.

```
Accuracy: 0.9767441860465116
[[ 3.77819685 -0.75341497  0.17248526 -0.71597049]]
    [-1.72205563]
```

The accuracy has reduced by a small amount from 1.0 (without scaling) to 0.9767. We can say that an accuracy of 1.0 is a sign of overfitting which can occur due to the regression model relying on unbalanced feature scales in a way that tightly fits the training data. After the features have been scaled the model will generalize better due to balanced feature contribution leading to a slight drop in accuracy.

1.11 Code in listing 3

When running the code as it is the following error is observed.

```
ValueError: could not convert string to float: 'Dream'
```

This has happened due to some categorical features not being dropped from the DataFrame. This can be solved by dropping the 'island' and 'sex' features from the DataFrame. Else we have to use a one hoy encoder to convert the categorical data before moving on to regression.

1.12 Using standard scaling or min-max scaling after using label encoding

Label encoding transforms categorical values into integer values. It can sometimes imply an ordinal relationship(a ranking/order) between categories. This does not make sense for unordered categories. Standard scaling or Min-Max scaling are used with continuous numerical features. Applying these scalers to integers from label encoding may not have meaningful outcomes and could mislead the model.
Instead of using a label encoder, we can use one hot encoder. This method transforms categorical feature values into binary columns (of 0s and 1s) without implying any ordinal relationship.

Question 02

- x_1: Number of hours studied
- x_2: Undergraduate GPA
- y: Indicator for whether the student received an A+
- $w_0 = -5.9$: Estimated intercept coefficient
- $w_1 = 0.06$: Estimated coefficient for x_1
- $w_2 = 1.5$: Estimated coefficient for x_2

1. What is the estimated probability that a student, who has studied for 50 hours and has an undergraduate GPA of 3.6, will receive an A + in the class?

- x_1: 50
- x_2: 3.6
- $w_0 = -5.9$
- $w_1 = 0.06$
- $w_2 = 1.5$

$$
\begin{aligned}
p(y_i = 1 \mid x_i, \mathbf{w}) &= sigm(w_0 + w_1 x_1 + w_2 x_2) \\
&= \frac{e^{w_0 + w_1 x_1 + w_2 x_2}}{e^{w_0 + w_1 x_1 + w_2 x_2} + 1} \\
&= \frac{e^{-5.9 + 0.06 * 50 + 1.5 * 3.6}}{e^{-5.9 + 0.06 * 50 + 1.5 * 3.6} + 1} \\
&= 0.9241
\end{aligned}
$$

2. To achieve a 60% chance of receiving an A+ in the class, how many hours of study does a student like the one in part 1 need to complete?

$$
\begin{aligned}
0.6 &= sigm(w_0 + w_1 x_1 + w_2 x_2) \\
&= \frac{e^{w_0 + w_1 x_1 + w_2 x_2}}{e^{w_0 + w_1 x_1 + w_2 x_2} + 1} \\
&= \frac{e^{-5.9 + 0.06 * h + 1.5 * 3.6}}{e^{-5.9 + 0.06 * h + 1.5 * 3.6} + 1} \\
h &= 15.09
\end{aligned}
$$

2 Logistic regression on real world data

2.1 Importing a dataset

I imported the 'Iris' dataset from the UCI Machine Learning Repository

2.2 Correlation matrix and pair plots

The obtained correlation matrix and the pair plots are given below.

	sepal length	sepal width	petal length	petal width
sepal length	1.000000	-0.109369	0.871754	0.817954
sepal width	-0.109369	1.000000	-0.420516	-0.356544
petal length	0.871754	-0.420516	1.000000	0.962757
petal width	0.817954	-0.356544	0.962757	1.000000

Figure 1: Correlation matrix

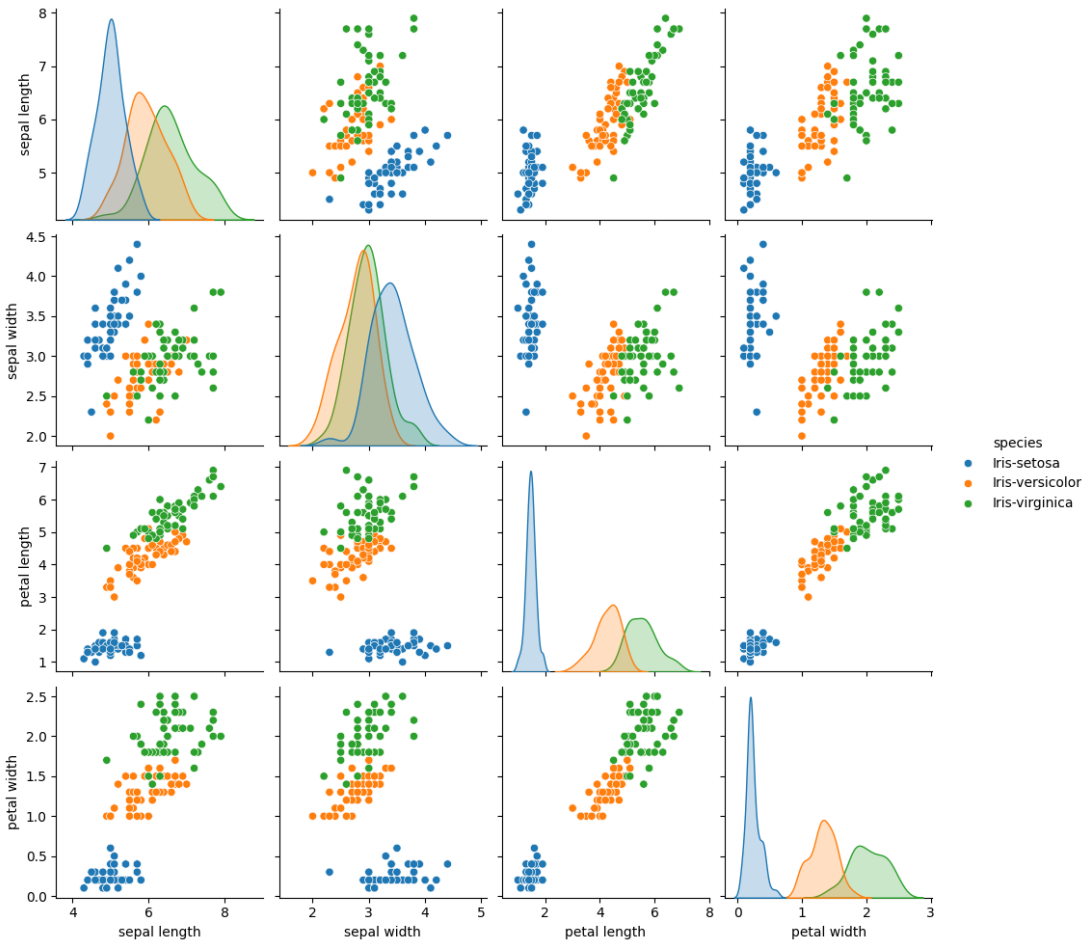

Figure 2: Pair plots

- **Strong Positive Correlations:**
  - Petal Length and Petal Width: $r = 0.96$
  - Sepal Length and Petal Length: $r = 0.87$
  - Sepal Length and Petal Width: $r = 0.82$
- **Moderate Negative Correlations:**
  - Sepal Width and Petal Length: $r = -0.42$
  - Sepal Width and Petal Width: $r = -0.36$
- **Weak Correlations:**
  - Sepal Length and Sepal Width: $r = -0.11$
- **General Observations:**
  - Petal dimensions (length and width) are more correlated and separable among species.
  - Sepal dimensions show weaker correlations and less clear differentiation between species.

2.3 Evaluation of the model

I trained a logistic regression model on the Iris dataset using a 'saga' solver. The feature data were scaled using the `StandardScaler`. A binary classification was done between the classes `'Iris-setosa'` and `'Iris-versicolor'`.
I evaluated the model's accuracy and it had achieved an accuracy of 1.0. This means it predicts the class correctly 100% of the time.

2.4 Determining if any features can be discarded

3 Logistic regression First/Second-Order Methods

3.1 Load the data

3.2 Implementing batch gradient descent to update the weights

The weights were initialized using `np.random.rand()` to avoid symmetry in the optimization process and allow the gradient descent to converge more effectively.

3.3 Loss function

$$\text{NLL(w)} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Binary cross-entropy works well because logistic regression gives a probability between 0 and 1, and cross-entropy measures the difference between these probabilities and the actual class labels (0 or 1). It shows how well the logistic regression model is predicting the correct class labels.

3.4 Plotting the loss against iterations

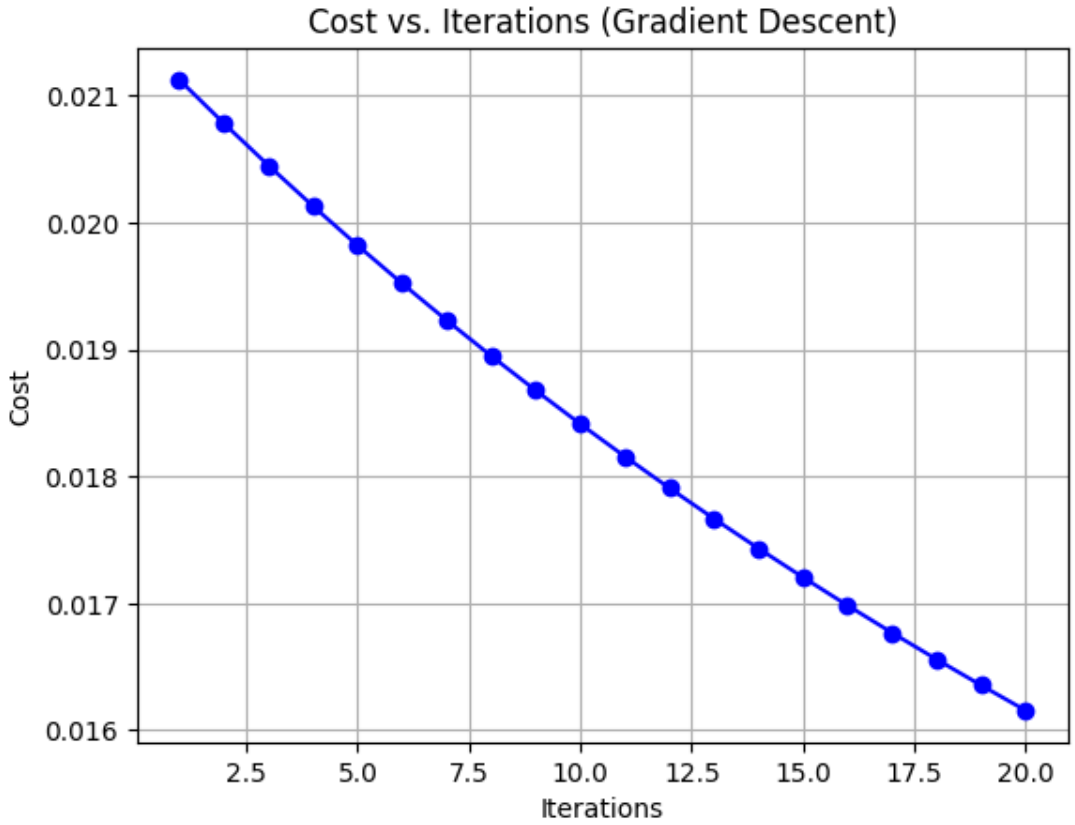

Figure 3: Cost vs Iterations

3.5 Using stochastic Gradient descent

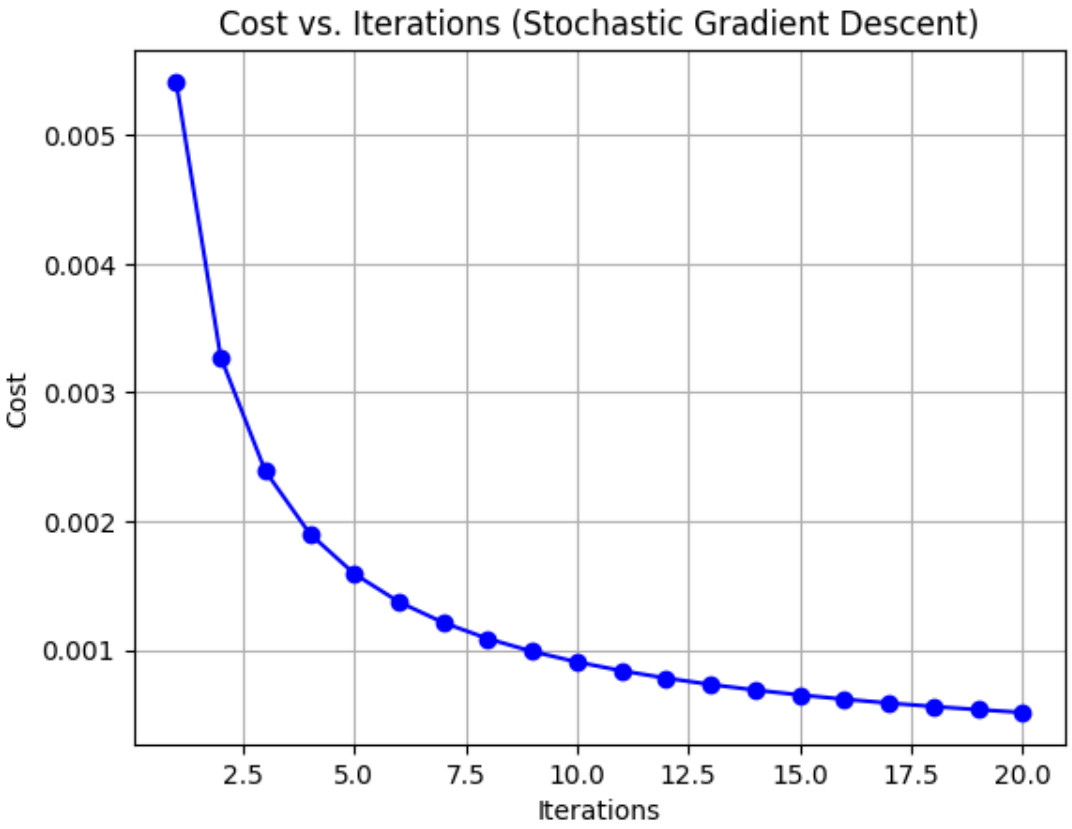

Figure 4: Cost vs Iterations stochastic gradient

3.6 Implementing Newton's method

When using the Newton-Raphson method the loss function converges to 0 before the number of iterations reach 20. (Included in the code.)

3.7 Plotting the cost against iteration in the Newton-Raphson method

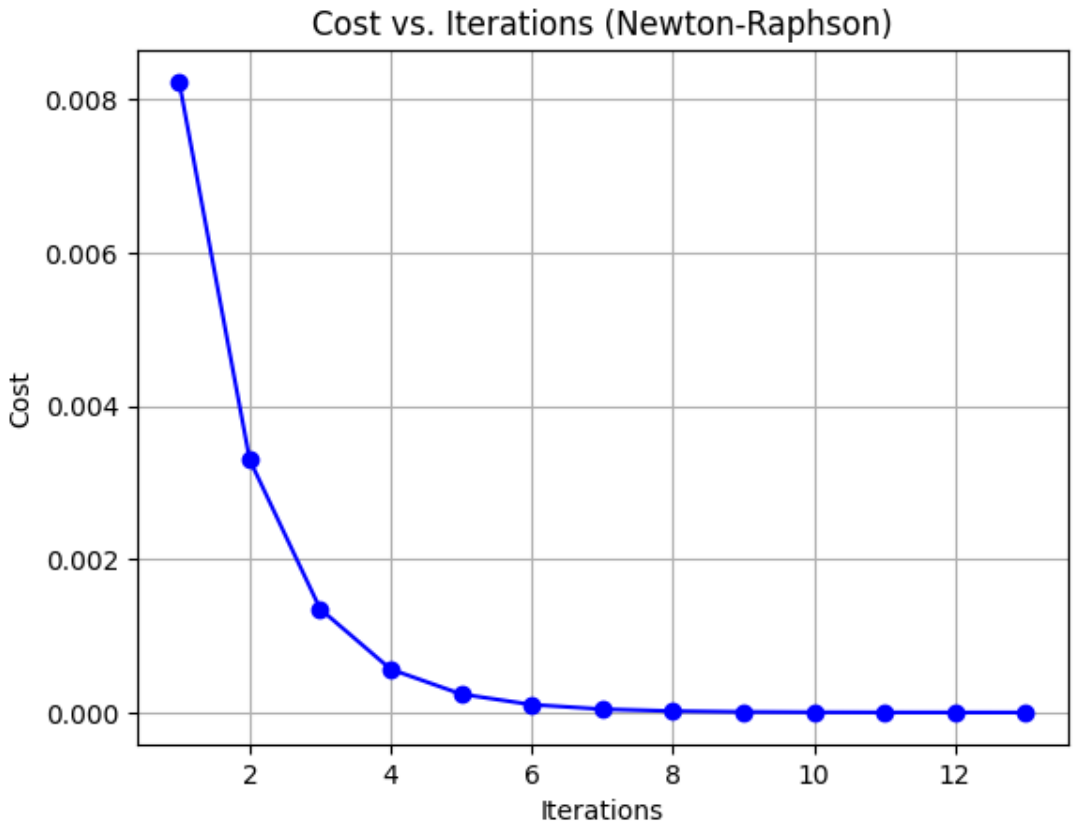

Figure 5: Cost vs Iterations Newton-Raphson method

3.8 All in one plot

- **Newton-Raphson (Blue)**:
  - The Newton-Raphson method quickly reduces the loss within the first few iterations, reaching near-zero cost rapidly.
  - This is typical of Newton's method, which converges quickly when the initial guess is close to the solution due to its second-order nature (using curvature information).
  - However, it may face difficulties with large datasets or complex, high-dimensional problems due to expensive Hessian matrix computations.
- **Stochastic Gradient Descent (Green)**:
  - Stochastic Gradient Descent (SGD) shows rapid convergence within the first few iterations and stabilizes at a low cost quickly.
  - SGD typically fluctuates more than batch gradient descent since it updates the weights based on individual samples, but it appears to have stabilized effectively in this case.
  - It is faster for larger datasets because it performs weight updates more frequently (after each data point), making it more efficient for online learning or large-scale optimization.
- **Batch Gradient Descent (Red)**:

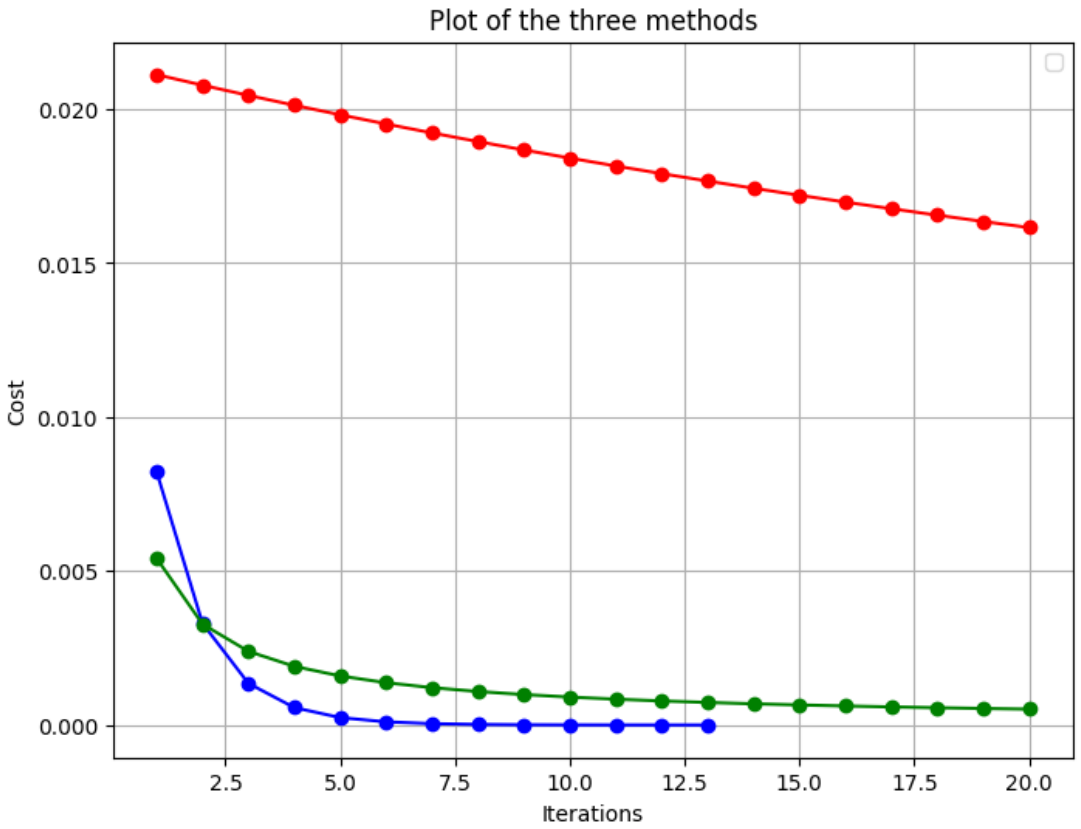

Figure 6: Cost vs Iterations All 3 methods

- Batch Gradient Descent is slower compared to Newton-Raphson and SGD, with a more gradual reduction in loss.
- It maintains a consistent reduction in loss over the iterations but doesn't reach as low a cost as quickly as the other methods.
- This method updates weights after computing the gradient over the entire dataset, which makes it slower and more computationally expensive for large datasets.

Conclusions

- **Newton-Raphson** is the fastest at reducing the loss initially, but its use is limited to smaller problems due to computational cost.
- **Stochastic Gradient Descent** is efficient for large datasets, showing rapid convergence with slight fluctuations early on.
- **Batch Gradient Descent** is slower in convergence but provides more stable updates over time.

3.9 New approaches to decide number of iterations

1. **Early Stopping with Cross-Validation:** Training is halted when the validation set's loss stops improving. This helps to prevent overfitting, ensuring that the model generalizes well to unseen data.

2. **Convergence Tolerance:** Training stops once the change in loss between iterations falls below a specified threshold (e.g., $\epsilon = 10^{-6}$). This indicates that the model has converged, and further training is unlikely to give significant improvements.

3.10 Changing the centers of the code

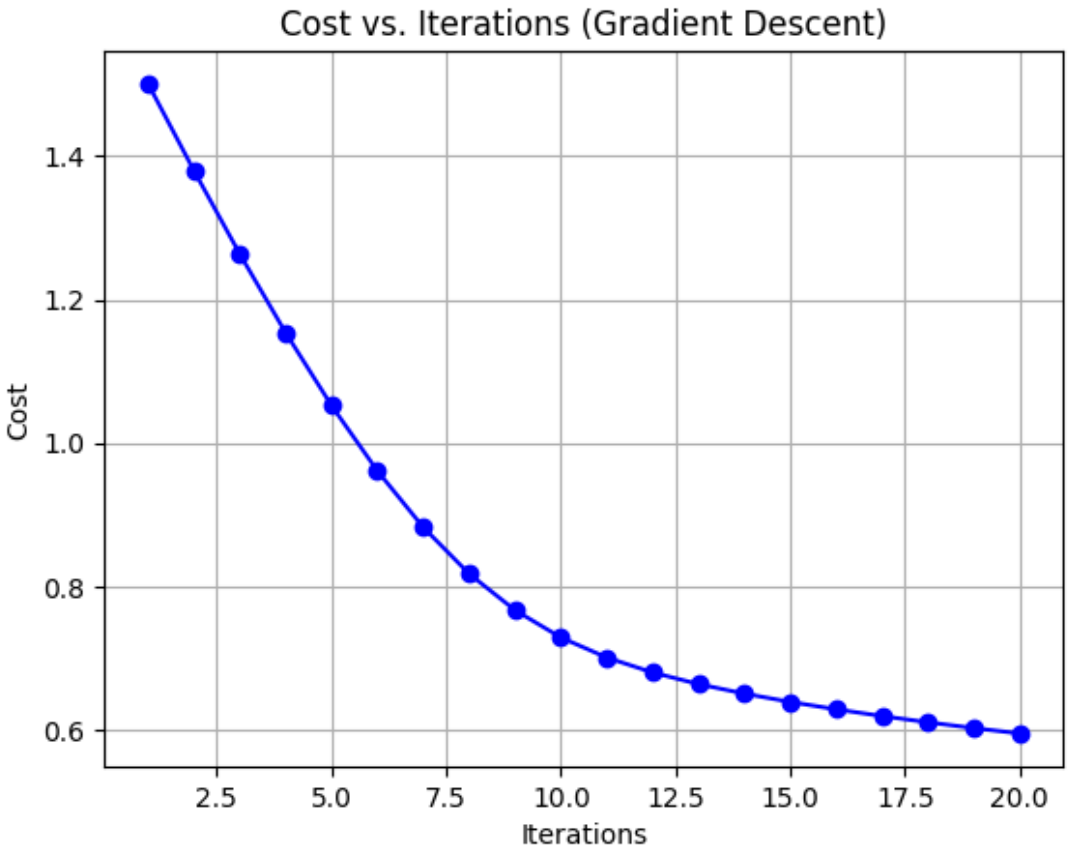

Figure 7: Batch gradient descent with new centers

The data indicates that with the new centers [[3,0],[5,1.5]], convergence is noticeably slower compared to the original configuration. In the earlier setup, the cost decreased quickly, showing rapid progress toward establishing the decision boundary. However, with the new centers, the cost shows only a slight reduction, reflecting a much slower decline in loss.
This slower convergence is likely due to the closer placement of the new centers, which increases overlap between the classes and makes it more challenging for the model to differentiate between them. Consequently, the gradient descent algorithm takes longer to reach an optimal solution, as the decision boundary becomes more complex, resulting in a more gradual decrease in the cost function.