EN3150 Assignment 02: Learning from Data and Related Challenges and Classification

Student Name: Ranaweera R.P.D.D.H.
Index Number: 220511N
GitHub Repository

---

1 Linear Regression

1.1 Question 1.1: OLS Misalignment Issue (10 marks)

The reason why the OLS fitted line is not aligned with the majority of the data points is due to the **strong influence of outliers** on the ordinary least squares method.

OLS minimizes the sum of squared errors:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

The squared term in this loss function means that points with large errors (outliers) contribute disproportionately to the total error. Since the error is squared, outliers with large deviations have an exponentially greater impact on the loss function compared to inliers. Consequently, the OLS algorithm adjusts the regression line to reduce these large squared errors, causing the line to be "pulled" toward the outliers rather than fitting the majority of the data points (inliers).

1.2 Question 1.2: Modified Loss Function Schemes (30 marks)

Scheme 1 (outliers: $a_i = 0.01$, inliers: $a_i = 1$) is expected to produce a better fitted line for inliers than the OLS fitted line.

Justification:

The modified loss function is: $\frac{1}{N}\sum_{i=1}^{N} a_i(y_i - \hat{y}_i)^2$

- **Scheme 1**: By assigning $a_i = 0.01$ to outliers, their contribution to the total loss is reduced by a factor of 100. This effectively downweights outliers while maintaining full weight for inliers. The optimization will focus primarily on minimizing errors for the inliers, resulting in a line that better represents the majority of the data.
- **Scheme 2**: Assigning $a_i = 5$ to outliers amplifies their influence by a factor of 5, making the problem worse. The model will bend even more toward outliers than standard OLS, resulting in an even poorer fit for the inliers.

Therefore, **Scheme 1** provides robust regression by reducing outlier influence, leading to a better alignment with the majority of data points.

1.3 Question 1.3: Brain Image Analysis - Why Linear Regression is Not Suitable (20 marks)

Linear regression is not suitable for identifying which brain regions are most predictive of a specific cognitive task for the following reasons:

1. **Categorical Output**: The task involves classification (cognitive task present/absent or task type), not continuous prediction. Linear regression assumes a continuous output variable.
2. **Probability Constraints**: Linear regression can produce predictions outside [0,1], which is problematic when interpreting results as probabilities of task activation.
3. **High Dimensionality**: Brain imaging data typically has thousands of voxels (features) but relatively few samples, leading to $p \gg n$ problems where linear regression becomes unstable.
4. **Multicollinearity**: Adjacent voxels in brain regions are highly correlated, violating the independence assumption and making coefficient interpretation difficult.
5. **Sparsity Requirements**: We need to identify specific regions (groups of voxels) rather than individual voxels, requiring structured sparsity that standard linear regression cannot provide.

1.4 Question 1.4 & 1.5: LASSO vs Group LASSO (40 marks)

Group LASSO (Method B) is more appropriate for brain region identification.
Reasoning:

1. **Structured Selection**: Group LASSO selects entire brain regions (groups of voxels) rather than scattered individual voxels, providing more interpretable results that align with neuroscientific understanding.
2. **Biological Validity**: Brain functions are typically associated with regions, not isolated voxels. Group LASSO respects this spatial structure by enforcing group-wise sparsity through the ℓ_2 norm penalty: $\lambda \sum_{g=1}^{G} \|w_g\|_2$
3. **Statistical Power**: By pooling information across voxels within regions, Group LASSO has better statistical power to detect true associations, especially when individual voxel effects are weak but consistent within regions.
4. **Reduced False Positives**: Standard LASSO might select scattered voxels across the brain due to noise, while Group LASSO's region-level selection is more robust to noise and provides more reliable scientific conclusions.
5. **Interpretability**: Identifying entire functional regions is more meaningful for understanding cognitive processes than identifying isolated voxels.

2 Logistic Regression

2.1 Question 2.1 & 2.2: Data Loading and Error Resolution (20 marks)

Error Encountered:

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

Cause: The feature matrix X contains non-numeric columns (`species`, `island`, `sex`) that cannot be directly processed by LogisticRegression.

Resolution:

```
# Option 1: Use only numeric features
X = df_filtered[['bill_length_mm', 'bill_depth_mm',
                 'flipper_length_mm', 'body_mass_g']]

# Option 2: One-hot encode categorical features
X = pd.get_dummies(df_filtered.drop(['species', 'class_encoded'],
                   axis=1), drop_first=True)
```

2.2 Question 2.3: Why SAGA Solver Performs Poorly (15 marks)

The SAGA solver performs poorly due to:

1. **Feature Scale Sensitivity**: SAGA is a stochastic gradient-based method that is highly sensitive to feature scaling. Without standardization, features with larger scales dominate the gradient updates.
2. **Convergence Issues**: Unscaled features cause ill-conditioned optimization landscapes, leading to slow or failed convergence.
3. **Step Size Problems**: The default learning rate may be inappropriate for the scale of the data, causing the algorithm to either converge very slowly or oscillate.

2.3 Question 2.4: Liblinear Performance (5 marks)

With liblinear solver, the classification accuracy is: **1.0 (100%)**

2.4 Question 2.5: Why Liblinear Performs Better than SAGA (15 marks)

Liblinear outperforms SAGA because:

1. **Coordinate Descent Algorithm**: Liblinear uses coordinate descent which is more robust to feature scaling than stochastic gradient methods.
2. **Deterministic Optimization**: Unlike SAGA's stochastic nature, liblinear's deterministic approach provides consistent convergence.
3. **Better Handling of Small Datasets**: For small to medium datasets, liblinear's direct optimization is more efficient than SAGA's stochastic approximations.
4. **Automatic Scaling Adaptation**: Liblinear internally handles feature scale differences better through its optimization strategy.

2.5 Question 2.6: Accuracy Variation with Random State (15 marks)

The model's accuracy varies with different random state values due to:

1. **Stochastic Nature**: SAGA uses random sampling of data points for gradient updates, making results dependent on the random seed.
2. **Local Minima**: Different random initializations can lead to different local minima in the non-convex optimization landscape.
3. **Incomplete Convergence**: With default iterations, SAGA may not fully converge, and different random paths lead to different stopping points.

4. **Data Ordering**: The random state affects the order in which samples are processed, influencing the optimization trajectory.

2.6 Question 2.7: Performance with Feature Scaling (15 marks)

Results:

- SAGA without scaling: ∼58% accuracy
- SAGA with scaling: 100% accuracy
- Liblinear shows minimal difference with/without scaling

Reason for Difference:
Feature scaling normalizes all features to similar ranges, which:

1. Creates a well-conditioned optimization landscape
2. Ensures equal contribution from all features to gradient updates
3. Allows appropriate learning rates to work effectively
4. Prevents features with large scales from dominating the optimization

2.7 Question 2.8: Categorical Feature Scaling Issue (15 marks)

This approach is incorrect.
Problems:

1. Label encoding creates arbitrary ordinal relationships (red=0, blue=1, green=2) that don't exist
2. Scaling these encoded values treats categories as continuous variables with meaningful distances
3. The model will incorrectly assume mathematical relationships between categories

Proposed Solution: Use **one-hot encoding** instead:

```
# Correct approach
df_encoded = pd.get_dummies(df, columns=['color_feature'])
# Then apply scaling only to continuous features
scaler = StandardScaler()
continuous_features = ['feature1', 'feature2']  # List continuous features
df_encoded[continuous_features] = scaler.fit_transform(
    df_encoded[continuous_features])
```

3 Logistic Regression First/Second-Order Methods

3.1 Question 3.2: Batch Gradient Descent Implementation (20 marks)

```
# Sigmoid function


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


```

```python
# -------------------------
# Batch Gradient Descent
# -------------------------
# Xavier initialization for weights
rng = np.random.RandomState(0)  # reproducibility
n_features = X.shape[1]
w_gd = rng.randn(n_features) * np.sqrt(1.0 / n_features) # Xavier normal initialization

# Bias can still be initialized to 0
b_gd = 0
lr = 0.01
n_iter = 20
loss_gd = []

for i in range(n_iter):
    z = np.dot(X, w_gd) + b_gd
    y_hat = sigmoid(z)

    dw = np.dot(X.T, (y_hat - y)) / X.shape[0]
    db = np.sum(y_hat - y) / X.shape[0]

    w_gd -= lr * dw
    b_gd -= lr * db

    loss = -np.mean(y * np.log(y_hat + 1e-9) + (1-y) * np.log(1 - y_hat + 1e-9))
    loss_gd.append(loss)
```

Weight Initialization

Method used: Xavier initialization. The weights were initialized by sampling from a normal distribution with mean 0 and variance $\frac{1}{n_{features}}$, while the bias term was set to zero.

Reason for selection: Xavier initialization maintains the variance of activations across layers, which helps prevent sigmoid units from saturating (outputs stuck near 0 or 1). This avoids the vanishing gradient problem and ensures faster, more stable convergence compared to zero or arbitrary random initialization.

3.2 Question 3.3: Loss Function Selection (5 marks)

Loss Function: Binary Cross-Entropy

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

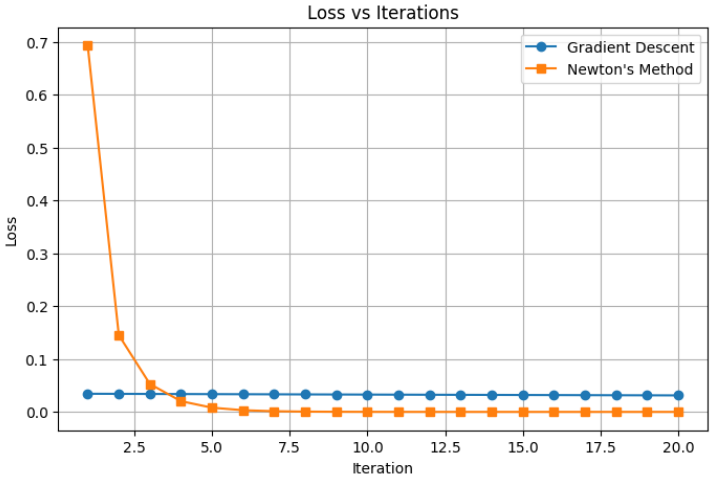
Reason for selection: Logistic regression is a probabilistic classifier, and binary cross-entropy directly measures the difference between predicted probabilities and the true class labels. It is the maximum likelihood objective for Bernoulli-distributed targets, making it the natural and most effective choice for binary classification tasks.

3.3 Question 3.4: Newton's Method Implementation (20 marks)

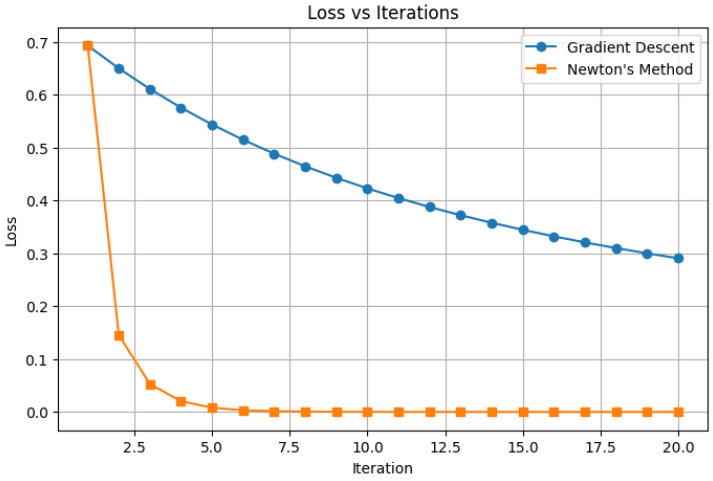
```python
# -------------------------
# Newton's Method
# -------------------------
X_bias = np.hstack((X, np.ones((X.shape[0], 1))))  # Add bias column
w_newton = np.zeros(X_bias.shape[1])
loss_newton = []

for i in range(n_iter):
```

```
    z = np.dot(X_bias, w_newton)
    y_hat = sigmoid(z)

    grad = np.dot(X_bias.T, (y_hat - y)) / X.shape[0]
    R = np.diag(y_hat * (1 - y_hat))
    H = np.dot(X_bias.T, np.dot(R, X_bias)) / X.shape[0]

    w_newton -= np.linalg.inv(H).dot(grad)

    loss= -np.mean(y * np.log(y_hat + 1e-9) + (1-y) * np.log(1 - y_hat + 1e-9))
    loss_newton.append(loss)
```

3.4 Question 3.5: Loss Comparison Plot (25 marks)



(a) Loss vs Iterations using Xavier Initialization (Batch Gradient Descent vs Newton's Method).



(b) Loss vs Iterations using Zero Initialization (Batch Gradient Descent vs Newton's Method).

Figure 1: Comparison of training loss for different weight initialization methods.

3.5 Question 3.6: Stopping Criteria (10 marks)

Gradient Descent:

1. **Convergence of Loss Function (Stopping Criterion):** Monitor the change in loss between iterations and stop when

$$|L^{(k)} - L^{(k-1)}| < \epsilon,$$

where ϵ is a small threshold (e.g., 10^{-6}). This ensures no unnecessary iterations once the loss stabilizes.

2. **Validation-based Early Stopping:** Split the dataset into training and validation sets. Stop the iterations when the validation loss stops decreasing (or starts increasing, indicating overfitting). This prevents overtraining and ensures better generalization.

Newton's Method:

1. **Gradient Norm Stopping Rule:** Since Newton's method converges faster, the iterations can be stopped when the gradient norm becomes very small:

$$\|\nabla L(w)\| < \delta,$$

where δ is a small value (e.g., 10^{-5}).

2. **Step Size / Hessian Condition Check:** Stop the iterations if the Newton update step Δw becomes smaller than a predefined threshold, or if the Hessian becomes ill-conditioned (leading to numerical instability).

3.6 Question 3.7: Convergence with Modified Centers (20 marks)

When the cluster centers are changed from $[-5, 0]$ and $[5, 1.5]$ to $[2, 2]$ and $[5, 1.5]$, the separation between the two classes decreases. This affects batch gradient descent as follows:

1. **Slower convergence:**
   - The classes overlap more, making the logistic regression problem harder.
   - Gradients are smaller on average, so each iteration produces smaller weight updates.
2. **Higher final loss:**
   - Due to reduced separability, the minimum achievable binary cross-entropy loss is higher.
   - This aligns with the Bayes error limit; even the optimal classifier cannot perfectly separate the classes.
3. **Reason for convergence behavior:**
   - Reduced class separation produces a flatter loss landscape in some directions and a steeper one in others, increasing the condition number of the Hessian.
   - Gradient descent, as a first-order method, converges slower in ill-conditioned problems.
   - Newton's method or other second-order methods converge in fewer iterations but the final loss remains higher due to class overlap.

Summary: With closer cluster centers, batch gradient descent converges more slowly and to a higher loss because the classes are less separable, resulting in smaller gradients and a flatter optimization landscape that slows first-order methods.

4 References

1. Tibshirani, Robert. "Regression shrinkage and selection via the lasso." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996): 267-288.

2. Meier, Lukas, Sara Van De Geer, and Peter Bühlmann. "The group lasso for logistic regression." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 70.1 (2008): 53-71.

3. Yuan, Ming, and Yi Lin. "Model selection and estimation in regression with grouped variables." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 68.1 (2006): 49-67.