COM S 673: Advanced Topics in Computational Models of Learning
Assignment #1

1. You need to submit a report and your code to Canvas. Your hard-copy report should include (1) answers to the non-programming part, and (2) analysis and results of the programming part. Please put all your code files and report into a compressed file named "HW#_FirstName_LastName.zip"
2. Unlimited number of submissions are allowed and the latest one will be timed and graded.
3. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
4. All students are required to typeset their reports using latex.
5. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE

Linear Models for Handwritten Digits Classification: In this assignment, you will implement the binary logistic regression model and multi-class logistic regression model on a partial dataset from MNIST. In this classification task, the model will take a 16×16 image of handwritten digits as inputs and classify the image into different classes. For the binary case, the classes are 1 and 2 while for the multi-class case, the classes are 0, 1, and 2. The "data" fold contains the dataset which has already been split into a training set and a testing set. All data examples are saved in dictionary-like objects using "npz" file. For each data sample, the dictionary key 'x' indicates its raw features, which are represented by a 256-dimensional vector where the values between $[-1, 1]$ indicate grayscale pixel values for a 16×16 image. In addition, the key 'y' is the label for a data example, which can be 0, 1, or 2. The "code" fold provides the starting code. You must implement the models using the starting code.

1. Data Preprocessing [15 points]: In this problem, you need to finish "code/DataReader.py".

   (a) Explain what the function *train_valid_split* does and why we need this step.
   Answer. As its name suggests, this function tries to split the training data into two parts: the training dataset and the validation dataset. It does so by considering the raw data's first entry and corresponding labels to the split index for training purposes and from the split index + 1 to the end for validation purposes. Since these two subsets of training data are derived from the original dataset, both training and validation procedures are considered part of the training process. However, the training dataset, typically the larger portion of the data, is the portion the machine learning model encounters first to figure out the model's parameters. An example of such parameters in linear regression would be the coefficients of each independent feature. Furthermore, the validation set helps fine-tune these parameters by comparing the predicted value for a certain label of this dataset based on its derived features. This process helps the model become more generalized and robust and prevents it from over-fitting, making it

applicable to datasets different from the one we're currently dealing with but serving the same purposes.

(b) Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

Answer. Assuming that the training set includes both the training and validation datasets (training set = training + validation dataset) while addressing this question, there are two possible approaches:

i. If we consider the current task—classifying digits 0, 1, and 2—then retraining the model on the entire training set is not valid. The validation dataset must remain untouched until the end of the training phase and should be used solely to evaluate the model's accuracy. This evaluation helps optimize the model and prevents overfitting. If we retrain the model using the validation set, we would no longer have an independent dataset to assess its performance, making it an improper practice. Therefore, the validation set must remain separate and unused during training.

ii. In general cases, if labeled data is limited, some techniques cautiously merge the validation set into training after hyperparameter tuning is complete. However, this requires a separate, unseen test set to properly assess the model's final performance.

(c) In this assignment, we use two hand-crafted features:

The first feature is a measure of symmetry. For a 16×16 image x, it is defined as

$$F_{symmetry} = -\frac{\sum_{pixel} |x - flip(x)|}{256},$$

where 256 is the number of pixels and $flip(\cdot)$ means left and right flipping.

The second feature is a measure of intensity. For a 16×16 image x, it is defined as

$$F_{intensity} = \frac{\sum_{pixel} x}{256},$$

which is simply the average of pixel values.

Implement them in the function *prepare_X*.

Answer. Please look at the function *prepare_X*.

(d) In the function *prepare_X*, there is a third feature which is always 1. Explain why we need it.

Answer. The value 1 corresponds to the bias weight w_0, which allows the model to make a prediction even when the weighted sum of its inputs is not sufficient on its own. The bias weight introduces a level of adaptability that ensures the model can learn and make predictions effectively. For instance, in this problem, without a bias, the model might only activate if at least one of the symmetry or intensity values is exactly at a certain threshold. However, by introducing a bias, we allow the model to make a prediction even if these features are slightly below or above the threshold. The actual value of bias determines how much a model is lenient or stringent in prediction.

(e) The function *prepare_y* is already finished. Note that the returned indices stores the indices for data from class 1 and 2. Only use these two classes for binary classification and convert the labels to +1 and -1 if necessary.

(f) Test your code in "code/main.py" and visualize the training data from class 1 and 2 by implementing the function *visualize_features*. The visualization should not include the third feature. Therefore it is a 2-D scatter plot. Include the figure in your submission.
Answer. Please look at Figure 1.

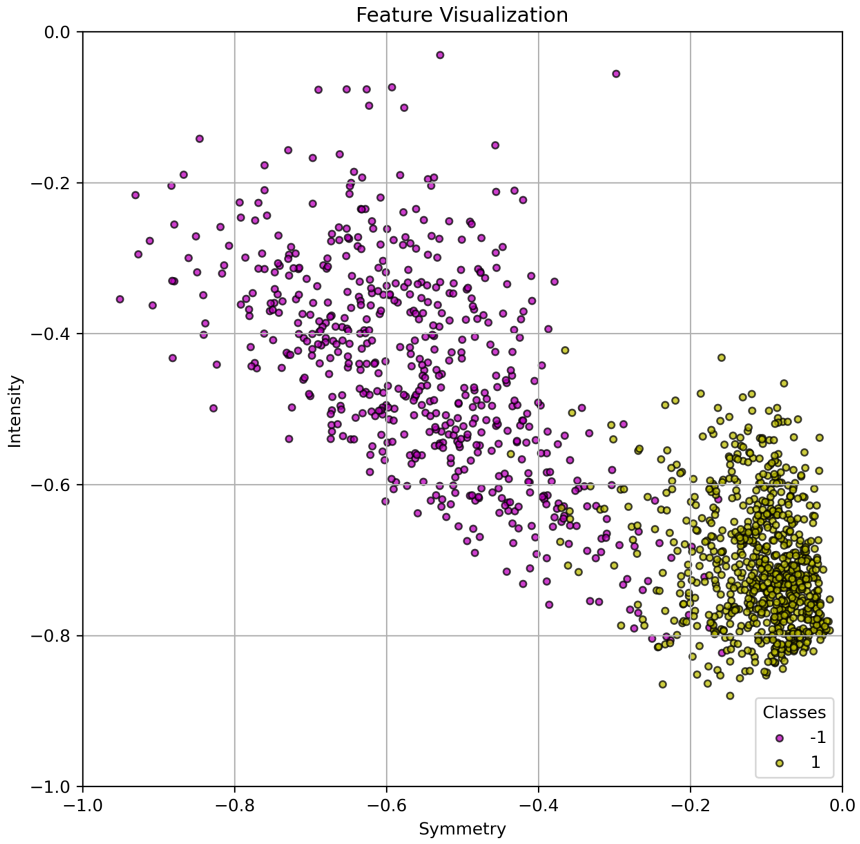


Figure 1: Visualization of training features where each data point's color represents the corresponding value of y.

2. Cross-entropy loss [20 points]: In logistic regression, we use the cross-entropy loss.

(a) Write the loss function $E(w)$ for one training data sample (x, y). Note that the binaray labels are 1 and -1.
Answer. Data likelihood for one training sample is defined as

$$p(y|\mathbf{x}, \mathbf{w}) = \begin{cases} \sigma(\mathbf{w}^\top \mathbf{x}), & y = 1 \\ 1 - \sigma(\mathbf{w}^\top \mathbf{x}), & y = -1 \end{cases}$$

We will show:

$$p(y|\mathbf{x}, \mathbf{w}) = \sigma(y\mathbf{w}^\top \mathbf{x})$$

Proof. i. Suppose $y = 1$; Then:

$$p(y|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(1 \cdot \mathbf{w}^\top \mathbf{x}) = \sigma(y\mathbf{w}^\top \mathbf{x})$$

ii. Suppose $y = -1$; Then:

$$p(y|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^\top \mathbf{x}) = 1 - \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}} = \frac{1 + e^{\mathbf{w}^\top \mathbf{x}} - e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}} = \frac{1}{1 + e^{\mathbf{w}^\top \mathbf{x}}}$$
$$= \frac{1}{1 + e^{-(-\mathbf{w}^\top \mathbf{x})}} = \sigma(-1 \cdot \mathbf{w}^\top \mathbf{x}) = \sigma(y\mathbf{w}^\top \mathbf{x})$$

□

We know Data likelihood for N training data is:

$$L(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} p(y_n|\mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} \sigma(y_n \mathbf{w}^\top \mathbf{x}_n)$$

Hence, by assigning $N = 1$ we have:

$$L(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{1} p(y|\mathbf{x}, \mathbf{w}) = \prod_{n=1}^{1} \sigma(y\mathbf{w}^\top \mathbf{x}) = \sigma(y\mathbf{w}^\top \mathbf{x})$$

Hence, the cross-entropy error (negative log-likelihood) would be:

$$\mathcal{E}(\mathbf{w}) = -\log L(\mathcal{D}|\mathbf{w}) = -\sum_{n=1}^{1} \log \Big[\sigma(y\mathbf{w}^\top \mathbf{x})\Big] = -\log \Big[\sigma(y\mathbf{w}^\top \mathbf{x})\Big]$$

(b) Compute the gradient $\nabla E(w)$. Please provide the intermediate steps.
Answer. $\nabla E(w)$ is a composite function of $f(x) = \log(x)$, $g(x) = \sigma(x)$ and $t(x) = \mathbf{x}^\top$ which are all differentiable on their domain. Hence, based on the chain rule:

$$f(g(t(x_0)))' = f'(g(t(x_0))) \cdot g'(t(x_0)) \cdot t'(x_0)$$
$$= \frac{1}{g(t(x_0))} \cdot \sigma(t(x_0)) \cdot (1 - \sigma(t(x_0)))$$

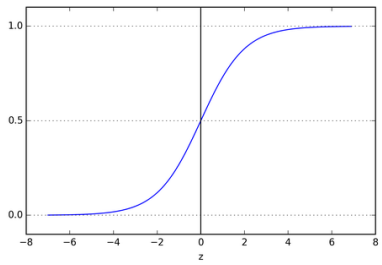
Hence:

$$\nabla E(w) = (-\log \Big[\sigma(y\mathbf{w}^\top \mathbf{x})\Big])' = -(\log' \Big[\sigma(y\mathbf{w}^\top \mathbf{x})\Big]) \cdot \sigma'(y\mathbf{w}^\top \mathbf{x}) \cdot (y\mathbf{w}^\top \mathbf{x})'$$
$$= -\frac{1}{\sigma(y\mathbf{w}^\top \mathbf{x})} \cdot \sigma(y\mathbf{w}^\top \mathbf{x}) \cdot (1 - \sigma(y\mathbf{w}^\top \mathbf{x}))y\mathbf{x}$$
$$= (\sigma(y\mathbf{w}^\top \mathbf{x}) - 1)y\mathbf{x}$$
$$= (\frac{1}{1 + e^{-y\mathbf{w}^\top \mathbf{x}}} - 1)y\mathbf{x}$$
$$= (\frac{1 - 1 - e^{-y\mathbf{w}^\top \mathbf{x}}}{1 + e^{-y\mathbf{w}^\top \mathbf{x}}})y\mathbf{x}$$
$$= -(\frac{e^{-y\mathbf{w}^\top \mathbf{x}}}{1 + e^{-y\mathbf{w}^\top \mathbf{x}}})y\mathbf{x}$$
$$= -y\sigma(-y\mathbf{w}^\top \mathbf{x})\mathbf{x}$$
$$= -y\frac{1}{1 + e^{y\mathbf{w}^\top \mathbf{x}}}\mathbf{x}$$

(c) Once the optimal w is obtained, it can be used to make predictions as follows:

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases}$$

where the function $\theta(z) = \frac{1}{1+e^{-z}}$ looks like



However, this is not the most efficient way since the decision boundary is linear. Why? Explain it. When will we need to use the sigmoid function in prediction?

Answer. Using the sigmoid function for predictions is inefficient since it adds extra computation.

Instead, we can use this simpler method:

$$\hat{y} = \begin{cases} 1, & \text{if } \mathbf{w}^T \mathbf{x} \geq 0 \\ -1, & \text{if } \mathbf{w}^T \mathbf{x} < 0 \end{cases}$$

The sigmoid function is useful during training because it is easy to differentiate, helping with gradient descent and loss minimization.

(d) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted label of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

Answer. Yes. Based on the definition, the decision boundary is $\theta(w^T x) = 0.9$. Thus, we can write

$$\sigma(\mathbf{w}^\top \mathbf{x}) = 0.9 \Rightarrow \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}} = 0.9 \Rightarrow 1 = 0.9 + 0.9e^{-\mathbf{w}^\top \mathbf{x}} \Rightarrow \frac{1-0.9}{0.9} = e^{-\mathbf{w}^\top \mathbf{x}}$$

which corresponds to the linear equation of

$$\mathbf{w}^\top \mathbf{x} = \ln\left(\frac{0.9}{1-0.9}\right) = \ln\left(\frac{0.9}{0.1}\right) = \ln(9) = 2.2$$

resulting in a linear decision boundary.

(e) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

Answer. The essential property of logistic regression that results in a linear decision boundary is the use of the sigmoid function as the activation function. Logistic regression assumes that the relationship between the input features $\mathbf{x}$ and the log-odds (logit) of

the target is linear. This means the decision boundary is defined by a linear equation, $\mathbf{w}^\top \mathbf{x} + b = 0$, where $\mathbf{w}$ represents the weight vector, and b is the bias term.

The sigmoid function, defined as $\sigma(z) = \frac{1}{1+e^{-z}}$, maps the linear combination of the features and weights, $\mathbf{w}^\top \mathbf{x}$, into a probability space $(0, 1)$. This function is monotonic with $\text{Domain}_\sigma = \mathbb{R}$ and $\text{Range}_\sigma = (0, 1)$, ensuring that for every value of $\mathbf{w}^\top \mathbf{x} \in \mathbb{R}$, there exists a unique threshold $T \in (0, 1)$ such that $\sigma(\mathbf{w}^\top \mathbf{x}) = T$. This bijective property ensures that the decision boundary is determined by solving:

$$\sigma(\mathbf{w}^\top \mathbf{x}) = T \Rightarrow \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} = T \Rightarrow 1 = T + Te^{-\mathbf{w}^\top \mathbf{x}} \Rightarrow \frac{1-T}{T} = e^{-\mathbf{w}^\top \mathbf{x}}$$

which corresponds to the linear equation of

$$\mathbf{w}^\top \mathbf{x} = \ln\left(\frac{T}{1-T}\right)$$

3. Sigmoid logistic regression [25 points]: In this problem, you need to finish "code/LogisticRegression.py". **Please follow the instructions in the starting code. Please use data from class 1 and 2 for the binary classification.**

   (a) Based on (b) in the last problem, implement the function *_gradient*.
   Answer. Please look at the function *_gradient*.

   (b) There are different ways to train a logistic regression model. In this assignment, you need to implement gradient descent, stochastic gradient descent and batch gradient descent in the functions *fit_GD*, *fit_SGD* and *fit_BGD*, respectively. Note that GD and SDG are actually special cases of BGD.
   Answer. Please look at the functions *fit_GD*, *fit_SGD*, and *fit_BGD*, respectively.
   While implementing *fit_GD*, we iteratively update the weight vector in the negative gradient direction to minimize the cross-entropy loss. Specifically, we compute the gradient using one sample at a time and update the weights accordingly, ensuring that all training samples contribute to the optimization process. Additionally, there's a variable called *tol*. This parameter is used as the convergence threshold. The optimization stops when the ℓ_2-norm of the gradient ($\|\nabla J(\mathbf{W})\|_2$) becomes less than or equal to *tol*.
   Regarding *fit_BGD* and *fit_SGD*, the stopping rule is reaching the number of iterations.

   (c) Implement the functions *predict* and *score* for prediction and evaluation, respectively. Additionally, please implement the function *predict_proba* which outputs the probabilities of both classes.
   Answer. Please look at the functions *predict*, *score*, and *predict_proba*, respectively.
   While implementing *score*, up to two significant figures will be shown for better readability.

   (d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results*. Include the figure in your submission.
   Answer. Please look at Figure 2. The scatter plot represents data points, where colors indicate class labels. The dashed black line represents the decision boundary, computed from the logistic regression weights. The x-axis corresponds to Symmetry, and the y-axis corresponds to Intensity. The legend in the lower left corner denotes the class labels.

   (e) Implement the testing process and report the test accuracy of your best logistic regression model.

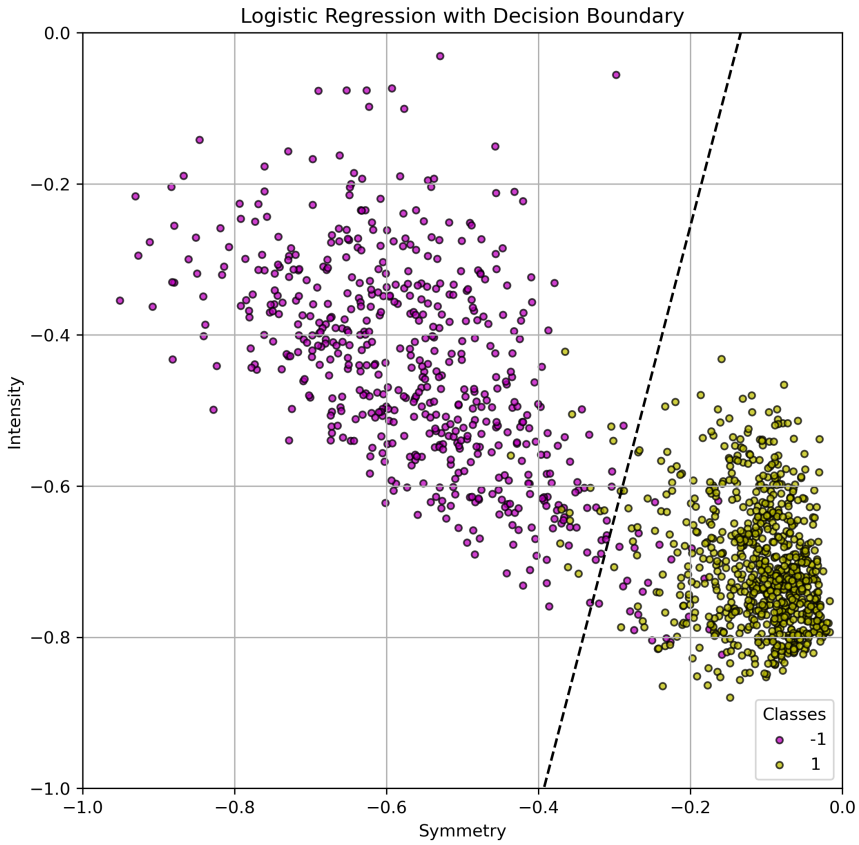


Figure 2: Decision boundary visualization of the logistic regression model. The dashed black line represents the decision boundary, computed from the logistic regression weights.

Answer.
For hyperparameter tuning for different gradient descent algorithms GD, BGD, and SGD were iterating over learning rates of ('$0.1, 0.3, 0.5, 0.7$'), maximum iterations of ('$50, 100, 150$'), and batch sizes of ('$50, 100, 150, 200, 250$') (only for BGD).
Best Hyperparameters: (BGD, 0.7, 150, 50)
Best Logistic Regression for Binary Classification Results:

- Best Logistic Regression Sigmoid weights: $\mathbf{W} = \begin{bmatrix} 2.59647319 & 19.43675937 & -5.03712078 \end{bmatrix}$
- Best Logistic Regression Sigmoid train accuracy: 97.19%
- Best Logistic Regression Sigmoid validation accuracy: 97.19%
- Test Accuracy: 94.37%

4. Softmax logistic regression [20 points]: In this problem, you need to finish "code/LRM.py". **Please follow the instructions in the starting code.**

   (a) Based on the course notes, implement the function *_gradient*.

   (b) In this assignment, you only need to implement batch gradient descent in the function *fit_BGD*.
   Answer.
   Please look at the functions *fit_BGD*. The stopping rule is reaching the number of iterations.

(c) Implement the functions *predict* and *score* for prediction and evaluation, respectively.
Answer.
Please look at the functions *predict* and *score* respectively.
While implementing *score*, up to two significant figures will be shown for better readability.

(d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results_multi*. Include the figure in your submission.
Answer.
Please look at Figure 3. The scatter plot represents data points, where colors indicate class labels. The dashed black line represents the decision boundary, computed from the logistic regression weights. The x-axis corresponds to Symmetry, and the y-axis corresponds to Intensity. The legend in the lower left corner denotes the class labels.

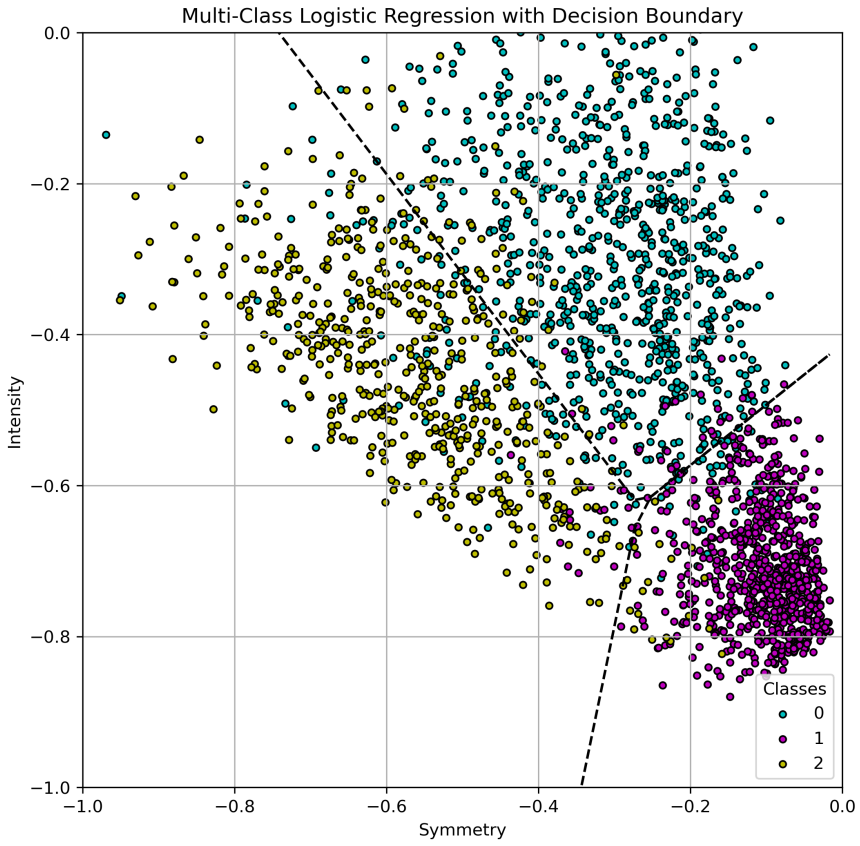


Figure 3: Decision boundary visualization of the multi-class logistic regression model. The dashed black line represents the decision boundary, computed from the logistic regression weights.

(e) Implement the testing process and report the test accuracy of your best logistic regression model.
Answer.
For hyperparameter tuning batch gradient descent algorithms BGD was iterating over learning rates of ('$0.1, 0.5$'), maximum iterations of ('$100, 1000$'), and batch sizes of ('25'). Best Hyperparameters: $(0.1, 1000, 10)$

- Best Logistic Regression Softmax Weights:

$$\mathbf{W} = \begin{bmatrix} 6.7332 & 0.4344 & 10.5483 \\ -1.1498 & 15.7370 & -8.5398 \\ -5.5834 & -16.1714 & -2.0085 \end{bmatrix}$$

- Best Logistic Regression Softmax Train Accuracy: 89.65%
- Best Logistic Regression Softmax Validation Accuracy: 87.3%
- Best Logistic Regression Test Accuracy for Multi-Class Classification: 86.48%

5. Softmax logistic vs Sigmoid logistic [20 points]: In this problem, you need to experimentally compare these two methods. **Please follow the instructions in the starting code.** Use data examples from class 1 and 2 for classification.

   (a) Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.
   Answer.
   Based on table 1, both the softmax and sigmoid classifiers were trained using the same dataset for 1000 iterations. The training, validation, and test accuracies were recorded. The results show that both classifiers achieve almost the same test accuracy of 93.1%. This suggests that either method can be used interchangeably without significant differences in performance.

Table 1: Comparison of Softmax and Sigmoid Classifier Accuracy

Accuracy Metric	Softmax Classifier	Sigmoid Classifier
Train Accuracy	59.48%	96.96%
Validation Accuracy	60.58%	96.83%
Test Accuracy	93.07%	93.72%

   (b) Explore the training of these two classifiers and monitor the graidents/weights. How can we set the learning rates so that $w_1 - w_2 = w$ holds for all training steps?
   Answer.

   The weights for both classifiers were computed by training them on the same dataset for the same number of iterations (see table 2). By setting the learning rate of the sigmoid classifier to twice that of the softmax classifier, we can observe that the difference between the softmax weights ($\mathbf{w_2} - \mathbf{w_1}$) exactly matches the weight of the sigmoid classifier ($\mathbf{w_0}$). This confirms that the softmax classifier's weight difference can be interpreted in the same way as the sigmoid classifier's weight.

Table 2: Comparison of Sigmoid and Softmax Classifier Weights (Rounded to 2 Significant Figures)

Classifier	**Weight**	**Values**
Sigmoid	w_0	$[-0.12, 6.5, -3.2]$
Softmax	w_1	$[0.062, -3.2, 1.6]$
	w_2	$[-0.062, 3.2, -1.6]$
	$w_2 - w_1$	$[-0.12, 6.5, -3.2]$

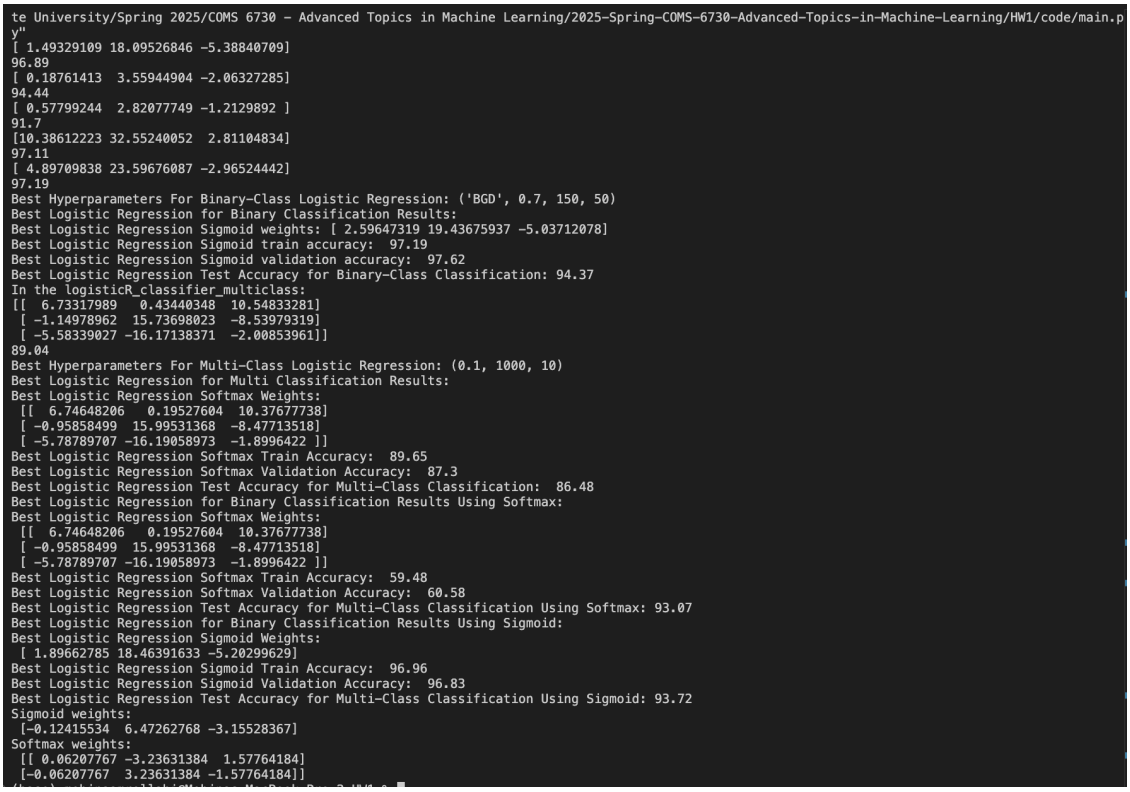


Figure 4: Hyperparameter Optimization and Performance Evaluation of Logistic Regression for Binary and Multi-Class Classification.